ÉCOLE NORMALE SUPÉRIEURE DE CACHAN
Écode Doctorale de Sciences Pratiques

T H È S E

présentée par

Nicolas LIMARE

pour l'obtention du titre de
Docteur de l'École Normale Supérieure de Cachan
spécialité mathématiques appliquées

Recherche reproductible, qualité logicielle, publication et interfaces en ligne pour le traitement d'image

—

Reproducible Research, Software Quality, Online Interfaces and Publishing for Image Processing

présentée et soutenue le XX juin 2012 à Cachan
devant le jury composé de

Président:	Xxx Xxx	-	Xxx.
Rapporteurs :	Gregory RANDALL	-	Universidad de la República, Montevideo, UY
	Guillermo SAPIRO	-	University of Minnesota, USA
	Konrad HINSEN	-	Centre de Biophysique Moléculaire, Orléans, FR
	Patrick VANDEWALLE	-	Philips Research, Eindhoven, NL
Directeurs :	Jean-Michel MOREL	-	ENS de Cachan, Cachan, FR
	Jacques FROMENT	-	Université de Bretagne Sud, Vannes, FR
	Lionel MOISAN	-	Université Paris Descartes, Paris, FR
Examinateurs :	Thierry GÉRAUD	-	EPITA, Kremlin-Bicêtre, FR
	Friedrich LEISCH	-	Universität für Bodenkultur, Wien, AT
	Gabriel PEYRÉ	-	Université Paris-Dauphine, Paris, FR

Centre de Mathématiques et de Leurs Applications — CMLA UMR 8536
École Nationale Supérieure de Cachan
61 Avenue du Président Wilson, 94235 Cachan Cedex, France

version pre-1, revision `50ff9124` (2012-04-19 17:58:20 +0200)
produced on April 19, 2012 with LaTeX and `pandoc`
more on `http://nicolas.limare.net/phd/`

Résumé

Cette thèse est basée sur une étude des problèmes de reproductibilité rencontrés dans la recherche en traitement d'image, et de manière générale dans les sciences appliquées et informatiques. Partant du constat du manque de vérifiabilité des publications scientifiques et de implémentations associées aux algorithmes publiés, nous avons conçu, créé et développé un journal scientifique, Image Processing On Line[1] (IPOL), dans lequel nous proposons un nouveau mode d'évaluation et de publication de la recherche en traitement d'image afin d'atteindre une meilleure fiabilité des logiciels et algorithmes issus de la recherche.

Les articles publiés dans IPOL incluent une implémentation complète des algorithmes décrits; cette implémentation est publiée, après validation par les rapporteurs qui s'assurent également de l'exhaustivité de la description des algorithmes. Un service web de démonstration des algorithmes publiés est joint aux articles, permettant aux utilisateurs de les tester sur des données libres avec le choix des paramètres et de consulter l'historique des expériences précédemment effectuées sur ce service. Ces extensions de la publication contribuent à plus d'échanges entre les équipes de recherche en favorisant la validation et la comparaison des résultats, la réutilisation des implémentations, et la constitution progressive d'un état de l'art vérifiable.

Pour cela, nous avons tout d'abord étudié la faisabilité d'un système de publication et d'expérimentation basé sur l'environnement web, ajoutant aux principes d'évaluation par les pairs d'un journal de recherche des contenus logiciels et des composantes interactives. Le choix de l'environnement web nous semble naturel compte tenu de l'évolution des réseaux et de leurs usages au cours des décennies précédentes, mais il limite l'interactivité qu'on peut attendre des services de démonstration des algorithmes et la précision de l'affichage des documents publiés. Il nous place de plus dans un contexte où les normes, standards et langages sont multiples et ne suffisent pas à définir complètement l'environnement logiciel dans lequel IPOL est utilisé via les navigateurs web des visiteurs.

La publication de logiciel nous a aussi amené à une tentative de synthèse du corpus législatif appliquable aux programmes informatiques, du point de vue du droit d'auteur et du droit des inventions. Il apparaît essentiellement que cette réglementation est nationale et hétérogène, et que l'association de manuscrits et logiciels en un produit unique de la recherche ne semble pas aisée tant ces deux entités disposent de statuts juridiques différents. Nous proposons dans ce contexte une politique de droits d'auteur et licences destinée à faciliter la diffusion des travaux de recherche et leur réutilisation pour des travaux ultérieurs.

La publication de logiciel est un objectif récent pour les journaux académiques, et on ne dispose pas pour le code source informatique de l'équivalent des règles qui, appliquées aux articles de recherche, définissent ce qui est publiable, et sous quelle forme. Nous proposons donc des règles en ce sens, visant à guider les rapporteurs dans leur évaluation du logiciel et contribuant à garantir qu'une implémentation sera utilisable par le plus grand nombre, qu'elle produira des résultats identiques dans divers environnements informatiques, et qu'elle est compréhensible et vérifiable, comme un manuscrit mathématique doit l'être. Ces règles d'évaluation du logiciel sont complétées par un projet de procédure automatisée de traitement des implémentations avant publication, complétant l'évaluation manuelle des codes source par des tests systématiques.

[1] `http://www.ipol.im/`

Les travaux de mise en place d'IPOL ont été menés dans trois directions: la définition des critères et des procédures de validation et publication adapté à ces nouveaux types d'articles, le développement d'outils logiciels génériques nécessaires à l'intégration des algorithmes publiés dans des services web de démonstration, et la mise en place et l'administration de ces services et outils sur une infrastructure de ressources informatiques. Après un peu plus d'une année d'activité, le projet scientifique que constitue IPOL nous apparaît très bénéfique à la recherche en traitement d'image. Malgré la charge de traitement manuel qui subsiste en raison de la nouveauté du sujet et du manque d'expérience et d'outils pour une gestion plus automatisée des articles et logiciels, plus de vingt contributions ont pu être publiées, et une cinquantaine de nouveaux articles sont en préparation. L'examen détaillé des implémentations et les tests intensifs via le service web de démonstration ont permis de publier des articles de meilleure qualité. La fréquentation d'IPOL montre par son volume, son origine internationale et la diversité des expériences faites avec le service de démonstration des algorithmes que ce journal est utile au-delà de la communauté de ses auteurs. Les auteurs sont globalement satisfaits de leur expérience et estiment que le travail supplémentaire que cette forme de publication nécessite est compensé par les avantages obtenus en terme de compréhension des algorithmes, de qualité des logiciels produits, de diffusion de leurs travaux et d'opportunités de collaboration.

À la lumière de cette expérience, on peut établir un programme de recherche ambitieux en traitement et analyse d'images adapté à cette forme de validation et diffusion de la recherche. Disposant de définitions claires des objets et méthodes, et d'implémentations validées, il devient possible de construire des chaînes complexes et fiables de traitement des images. On peut également souhaiter que d'autres journaux scientifiques et groupes de recherche adoptent des méthodologies similaires.

Abstract

This thesis is based on a study of reproducibility issues in images processing and computational research. From a constatation of the unverifiability of scientific publications and of the implementations associated with published algorithms, we designed, created and developed a scientific journal, Image Processing On Line[2] (IPOL), in which we propose a new method for the evaluation and publication of research in image processing, in order to improve the reliability of research algorithms and software.

Articles published in IPOL include a complete implementation of the algorithms. This implementation is published, after a validation by the reviewers, who shall also ensure that the description of the algorithms is complete and detailed. A demonstration web service is attached to every article, allowing users to test the algorithms on their data with their choice of parameters. The full history of the experiments previously performed with this service is also publicly available. These principles lead to further exchanges between research groups working on image processing, they promote the validation and comparison of the algorithms, the reuse of implementations, and the progressive compilation of a verifiable state of the art.

To this end, we first studied the feasibility of a publication and test system based on a web environment, adding software and interactivity to the traditional peer-review principles of research journals. The choice of the web environment seems natural given the evolution of networks and their uses in previous decades, but it also limits the interactivity of the demonstration services the rendering accuracy of the published documents, and in this context multiple norms, standards, and languages are not sufficient to define the software environment in which IPOL is used via the web browsers of visitors.

To publishing software we also reviewed the copyright and patent laws that apply to computer programs. It appears that this regulation is heterogenous, based on national laws, and that the combination of software and manuscripts in a single product of the research activity seems difficult because of the different legal status of these two items. In this contect, we propose a copyright and license policy to facilitate the dissemination of research and its reuse for subsequent works.

Publishing software is a recent goal for academic journals, and there is no equivalent for the source code to the rules governing, for research papers, what is publishable, and in what form. We therefore propose pour software guidelines, to facilitate the production and review of verifiable and usable software for reproducible research. These guidelines are completed by a proposed procedure for automated tests on a software implementation before its publication.

Establishing IPOL required some work in three directions: the definition of criteria and procedures to validate and publish these new types of articles, the development of generic software tools used for the integration of the algorithms into demonstration web services, and the setup and administration of these services and tools on an infrastructure of IT resources. After more than one year of operation, the IPOL scientific project seems very useful to research in image processing. Despite the need for manual processing that remains because of the novelty of the subject and the lack of experience and tools for an automated management of articles and software, more than twenty contributions have been published,

[2] `http://www.ipol.im/`

and fifty new articles are in preparation. Thanks to the detailed examination of the implementations and extensive testing via the demonstration web service, we published articles of better quality. The usage of IPOL shows by its volume, its international origin and the diversity of the experience with the demo service that this journal is useful beyond the community of its authors. The authors are usually satisfied with their experience and they feel that the extra work that this form of publication requires is offset by the benefits obtained in terms of understanding of the algorithms, software quality, exposure of their work and opportunities for collaboration.

In light of this experience, we can establish an ambitious research program in image processing and analysis adapted to this procedure to validate and distribute the research. With clear and reliable definitions of the objects and methods, with verified and trusted implementations, it becomes possible to build complex image processing chains. One can also hope that other journals and research groups adopt similar methodologies.

Basically, software is the specification for how the software is supposed to work. And anything less than the specification doesn't really tell you anything about how it's ultimately going to behave. And that just makes software really, really hard.

— Douglas Crockford

I don't think a program is finished until you've written some reasonable documentation. And I quite like a specification. I think it's unprofessional these people who say, "What it does? Read the code." The code shows me what it does. It doesn't show me what it's supposed to do.

— Joe Armstrong

in Peter Seibel, *Coders at Work: Reflections on the Craft of Programming*

Contents

Chapter 1

Introduction

Contents

1.1 Context and Previous Works

For computational sciences, detailed algorithms and implementations are an integral part of the research results and should therefore be published extensively. This has been continuously pointed out in the literature about reproducible research, from the seminal papers by Claerbout [80] and Buckeit and Donoho [67] to recent columns about software quality, or the lack of, in computational sciences [32, 245]. This goal is far from attained in the image and signal processing community, where only 12% of the published articles include the implementation details and 9% provide a source code [353].

Indeed, image processing and computer vision are young sciences which emerged at the end of the seventies. Their publication system is far from mature. Algorithmic exchange in this community faces serious obstacles: multiple software developing environments have grown in each research group without interoperability. Source code is subject to portability issues and depends on local tools to process or exploit the result. And software maintenance is a costly issue. A tested, reliable implementation is estimated to require three times more resources than a working prototype [61], and integrating different implementations together requires again three times more. Most labs have not the permanent workforce to finalize and maintain a software and to adapt it to the changing computing environment. Next to missing software engineering skills of scientists, software maintenance and consolidation is therefore one of the main problems in this field.

In these conditions, research groups are unable to communicate efficiently by software and are therefore limited to paper journals and conferences. Image processing libraries and development environments are not a sufficient response, because in the absence of a common base they reinforce the software fragmentation. Existing software journals, when they only publish codes, lack the precise scientific evaluation and specification of the implemented algorithms and a discussion about their properties, qualities and limits. When an implementation is provided by the authors of a research article, it is usually available *as-is* on their personal research web pages. Out of the peer-reviewed scientific publishing process, *there is no control that the provided source code exactly implements the described algorithm, there is no guarantee on the correctness and usability of this implementation, or on its long-term availability.* We have systematically observed that the proposed code differs significantly from the paper publication, that the paper publication is not enough to characterize an algorithm, and that conversely the disclosed code contains parts that are not documented or explained in the paper.

Multiple recent tentatives to address the vast problems od conservation, exchange and validation of computational science software material (code, executable programs, data and results) shows an increasing concern about the reliability of the computational science corpus. Some of them were presented in workshops related to the reproducible research methodology, such as the *Executable Paper Grand Challenge* [111, 137], *Reproducible Research: Tools and Strategies for Scientific Computing* [218], and *Rencontre de réflexion autour de la recherche reproductible* [87] in 2011 and 2012, others were independently developed and match some of our needs for a reproducible computational research methodology. We present different tools and solutions hereafter, summarized for comparison in table 1.1.

1.1.1 Web Services

The first example we found of online demos for image processing is Rolf Henkel's "*Web-based Image Processing*" [170], with 14 demos of stereo algorithms, segmentation, edge

	algorithmic description	source code available	executable program available	linked description and code	source code reviewed	local execution	server-side execution	execution archive
WEB DEMOS								
Rolf Henkel's "Web-based Image Processing"	·						●	●
EPFL Biomedical Imaging Group online demos	·		●			●		
Tomas Pajdla's "CMP SfM Web Service"		●			●			
RunMyCode	·	·		·			●	
Verifiable Computational Result		●			●			●
Flash image editors: Photoshop Express Editor			●			●		
Web image editors: Phixr							●	
LITERATE PROGRAMMING								
Sweave, Lepton	●	●		●		·		
R2	●	●		●		·	●	
STRUCTURED AND ACTIVE DOCUMENTS								
Amrita	·		●	●		●		
IODA, Planetary	·			●				
Collage Authoring Environment	●			●			●	
SOFTWARE JOURNALS								
Mathematical Programming Computation	●				●			
Insight Journal	●	●		●	·			
Open Research Computation	●	●		●	●			
PORTABLE PROGRAMS								
CDE			●					
ActivePaper	·		●			●		
SHARE, Papier Mâché	·	·	●				●	
IPOL	●	●		·	●		●	●

Table 1.1: Comparison of algorithm, software and demo publishing solutions. (●: yes, ·: sometimes or partially)

detection, texture analysis and color transformation. This set of demos was available from 1994 to 2002 and processed more than 40000 requests. The author claims they were "*among the first interactive pages available on the internet*". The archives of these web pages [171, 172] show, for each demo, a short explanation of the algorithm with examples and references to published articles, and a list of the data processed. Although our demos were developed before we knew of Rolf Henkel's previous work, the similarity of the concepts and content is striking, and we understand it as a confirmation that to be complete, an online demo project must include the documentation about the demonstrated algorithm, academic references, and an archive of demo activity.

The EPFL Biomedical Imaging Group also publishes online demos[1], developed between 2001 and 2012. These demos are organized in three categories: 12 research demos, with recent results in image processing, 9 teaching demos used to illustrate an image processing course, and 12 student demos probably designed during internships. Most of these demos are executed in the browser, as Java applets. A few ones only distribute a MATLAB or Java code, and should not be labeled as "demos". With the Java technology, we get a more interactive experience with the program run locally. On the other side, without a server-side support, everyone using these demos is isolated, and no information is available from the usage of the demos by other people. Moreover, the Java technology is still problematic, 25% of the web visitors have no Java support in their browser and this solution failed to gain a large momentum since its first release fifteen years ago.

More early online demos certainly existed. Anyone with a working knowledge about web technologies could propose interactive web pages involving some transformations of an image. However, the two aforementioned examples are the only ones with an academic research background we could trace back to more than 10 years ago, and still available online. Other examples of web demos have been used in recent years, such as Tomas Pajdla's non-interactive web service for his "structure from motion" algorithm[2] where users upload a set of several images and receive later an email with the result of the algorithm. Web interfaces are also commonly used for benchmarks and challenges in the image processing community.

RunMycode[3] [83] is a recent example of web-based online demos, focusing on the demo web service. RunMyCode allows "*people to run computer codes associated with a scientific publication (articles and working papers) using their own data and parameter values*". When they publish an article, researchers can create a "companion website" on RunMyCode, where they upload their R or MATLAB code and describe the input and output of their software. Then, people interested in these articles article will have to possibility to process their own data with the authors code. This service is provided on a best-effort basis, and the usability of the MATLAB demos in the future depends on the availability of the MATLAB computing environment for future computing system, for which the RunMyCode administrators rely on vendor's goodwill.

Another project specializes in the storage and indexation of the results of software research experiments. A Verifiable Computational Result web service[4] [139] receives, identifies and archives on demand the input and output of a program. The code itself can be archived with the data, and this system is well adapted to scripting languages like Python or MATLAB.

[1] `http://bigwww.epfl.ch/demo/`
[2] `http://ptak.felk.cvut.cz/sfmservice/`
[3] `http://www.runmycode.org/`
[4] `http://vcr.stanford.edu/`

Our project, Image Processing On Line (IPOL) [227] merges these two services with the online journal model. Web interfaces to run research codes are made available with the archive of their usage, and this is linked to the preprints and articles (manuscript, data and software) published in the journal. In IPOL, articles and demos are available online together but can be split and still be useful and relevant. They are two different online contents: not being on the same page, they are not submitted to the same editorial rules and may be developed, in collaboration, by different authors. Article and demo are published together because they support each other: the article explains the demo, and the demo illustrates the article. But they are two different objects with incompatible properties. The article is a static, immutable, reviewed document that can be distributed and archived. The demo in only an online interface to a remote software execution facility, based on the software published in the article. It only exists in one place, the demo server, collects the usage of the research software and can be updated according to the evolutions of the web technologies.

These three examples of online demo services use standard, classic web technologies. The novelty is in the application of these technologies to a scientific workflow in which the static research article is completed with dynamic software services.

We can also cite web-based image edition tools, such as Photoshop Express Editor[5], Splashup[6] or Sumopaint[7]. These services aim at replacing desktop-based image edition software, but they only provide basic and unscientific functions. Moreover, those three services use the Flash technology, which means they are executed locally, as a portable software rather than a real server-based service. Phixr[8] is another website with image edition function; this one really uses a server-side processing architecture. Finally, Tineye[9] for reverse image search and Photosynth[10] for image matching and 3D recomposition show how research algorithms can be used for full-featured web services for the general public.

1.1.2 Literate Programming Revisited

The first group of developments is a variation on literate programming, a method to write a program together with an explanation of the algorithm, by focusing on explaining the program to an human reader [200, 201]. The author writes the code and its description together in a single structured document, and uses specialized tools to transform it into two products, the software and its documentation. Donald Knuth designed the original literate programming environment, WEB, for the LaTeX and Pascal languages. The next generations added support for HTML documentation and other programming languages (C, C++, Perl, Caml, ...).

This concept was later revisited for research articles in computational science. Instead of the association of a program and its documentation, one could weave the text of a research article with the code and data used to produce the tables and figures included in the article. This procedure guarantees that the code and data are available with the article, and that the content of the article is really obtained from this code and data.

[5] `http://www.photoshop.com/tools/expresseditor`
[6] `http://www.splashup.com/`
[7] `http://www.sumopaint.com/`
[8] `http://www.phixr.com/`
[9] `http://www.tineye.com/`
[10] `http://www.photosynth.net/`

Statistics researchers use this system, Sweave, with LaTeX and the R programming language [214]; as of 2012, a new environment, Lepton, is also available with similar goals but support for more languages [222]. The last iteration of the concept, R2 [216], integrate these tools in a web service to automatically build, test and validate the articles to be published.

These tools add automation to the reproducible research principles: a research article is not complete without all the software material needed to reproduce its claims [67, 80]. Once the execution of the research code is automated on the publishing servers, we can imagine the possibility of a re-execution of this code, close to an online demo.

1.1.3 Structured Documents and Active Documents

Research articles are usually published as monolithic and static electronic documents, a PDF file reproducing a paper medium. The Sweave and Lepton tools enhance the PDF production chain by building and inserting the results of the computations, with references to the code. Other works like Amrita [292] are exploring the possibilities of interactive PDF and the use of the PDF viewer as an interactive computing environment.

An alternative direction is to publish research articles as native web documents, add interactivity, and use the result as online demos structured as articles. The first step is to model the electronic article document. The Interactive Open Document Architecture (IODA) [318] is a multi-layered representation of such document, in which each article component can be identified, queried and modified. Planetary [206] is another long-term effort to develop the representations, languages, ontologies and tools for rich and active web documents. On the viewer side, the lack of correct rendering of math content in web browsers is supplemented by JavaScript layers in JSMath or MathJax [76, 77].

The interaction in executable articles is covered by works like the Collage Authoring Environment [266], used to produce a web document an the infrastructure supporting the on-demand re-computation of the content of the articles with a web user interface and the server-side execution of code snippets.

All these projects share the idea that the research article is the natural environment for the execution of the research code.

1.1.4 Software Journals

Some research journals consider software a primary material. The Mathematical Programming Computation journal [257] includes accompanying data and software with the manuscripts. This software is evaluated and tested during the review process, and when possible the results included in the article are verified. However, there are no formal criteria for software quality, and no requirement for the software to be portable.

The Insight Journal [180] publishes software contributions in the fields of medical image processing and visualization. This journal does not follow the usual review process with a scientific committee choosing reviewers; instead, the articles are available for public review and feedback.

Finally, the Open Research Computation journal [282] publishes articles about software used by researchers. The software is required to be available under a open source license, and is reviewed and tested during the publishing process.

1.1.5 Portable Executable Programs

A program usable and valid in a given computing environment may not be usable or give the same results in other environments. The hardware (CPU instruction set, floating-point model) and software (operating system calls, libraries, compilation tool chain, interpreters, interfaces) define this environment. It varies over the computing landscape with a variety of machines and systems, and over time with new software versions and computing platforms.

There are plenty of examples illustrating these incompatibilities and obsolescence. ARM processors used on mobile computing tools [22], x86 processors on desktops and SPARC processors on supercomputers [343] have different instruction sets. Windows and Linux systems have different programming interfaces. Nvidia GPU-assisted computing does not handle floating-point numbers the way other processors do [267]. Every new version of the MATLAB environment has some incompatibilities with previous releases [182]. The programming interface of the FFTW library version 3 is not compatible with the version 2 of this library [242]. Python 3 cannot interpret some code written for Python 2 [351]. These situations can happen everyday in a computer scientist's life. And when a program includes details tied to an environments, it is not usable on others, therefore not testable and verifiable. This defeats reproducibility.

System Images and Virtual Machines

One strategy to achieve portable software execution is to save a system image of the experimental computing environment when a computational science result is obtained, shared or published, and use a system virtual machine to replay the computations or explore the computing tools. This has been used by Van Gorp *et al.* for workshops with the SHARE system [349,350] and is proposed by Brammer *et al.* for publishing the Papier Mâché system [59].

Virtual machine tools are appropriate for collaboration and archival of computing environments, but we consider they are not adapted to the publication of reproducible research, for reasons detailed later.

Process Virtual Machines

Another possibility is to use a process-level compatibility layer. A program compiled for a process virtual machine target is not tied to the operating system or hardware architecture. The binary program is made of virtual machine bytecode instead of processor instructions, and the program is not run by the operating system kernel, but by the virtual machine manager. With this abstraction layer, only the virtual machine manager needs to be adapted to the computing environment and programs are written for a generic abstract machine. The Java Virtual Machine (JVM) [229] is a popular process virtual machine, created for the Java programming language but expanded since to support other languages (Pascal, Python, Ruby, Lua, ...) and new languages created for the JVM target (Clojure, Scala, ...). The ActivePaper proposal [174] packages (with the HDF5[11] format), research programs computed for the JVM platform to avoid portability issues, with their data and documentation. This direction could be explored, to replace or complement the centralized

[11] HDF5 is a data model, library, and file format for storing and managing data (`http://www.hdfgroup.org/HDF5/`).

server-size online demo model with a decentralized, downloadable and executable system of locally executable demos.

Yet another possible direction is the automated packaging of a binary compiled program with all its dependencies, as done with CDE [161,162]. This doesn't allow real portability over different hardware,but it can solve the problems of dependencies and missing libraries and contribute to the distribution of locally executable research programs without the JVM requirement.

Finally, IPOL chose to benefit from the portability of source code and enforce a restrictive software guideline, adhere to standards and APIs, and distribute the programs in source form.

1.2 Software and Reproducibility

Xin Li's web directory of Reproducible Research in Computational Science (RRCS) [221] contains 800 entries, gathered between 2007 and 2012, linking to image processing research code and tools available online. From this list, we can estimate the potential for growth and adoption of the reproducible methodologies in the image processing research community and plans for IPOL.

1.2.1 Potential Articles and Authors

The RRCS repository is divided into 27 topics, from "image denoising" to "machine learning". IPOL already contains articles or preprints in 12 of these topics, with latest state of the art algorithms. Moreover, some topics covered in IPOL are not included in Xin Li's list (color, contrast and camera calibration), because they are not considered fashionable in the recent trends.

The directory covers most of the computational image processing. After careful evaluation of all these entries, we can count about 400 different interesting algorithms worth a full publication with online demo, about forty topics with ten important algorithm per topic.

As of early 2012, IPOL has 30 articles published or close to the publication, and 60 other papers being prepared, after little more than one year of full activity. If this cadence is maintained, we can expect to cover most of state of the art of the discipline.

The authors featured in the directory are successful and highly ranked researchers, as shown by their web pages, prizes and publications. But the inclusion of their codes in the RRCS list only means that they felt comfortable enough with their work to make it publicly available; only a fraction of these codes will be usable by other researchers and for an online demo. This shows that the requirements for reproducible research as enforced by the IPOL policies are very selective.

A research journal needs 100 to 300 regular contributors, three to ten times more than the current number of IPOL authors. The RRCS directory suggests a few hundred potential authors.

1.2.2 Long Term Perspectives

The comparison of the number of codes published in one form or another and with the number of image processing articles is shockingly low: the ratio is around 1/100 or 1/1000. The codes proposed in the RRCS repository implement a single algorithm or a small set of algorithms with their environment. Each of these algorithms are atomic, they perform one single action on an image. For the best scientific output, the research community would only need to publish these strong, reliable atoms. But for the moment the codes are only bonus supplementary materials made available at the initiative of the authors. The research community is not organized to produce implementations, it does not consider software its primary production and software is not valued in publications.

Without recognition and consolidation of these software building blocks, the researchers always start from scratch and are unable to combine the algorithmic atoms into software molecules. A few processing chains have been built, but their authors admit that they are complex kludges with no building block completely mastered.

In this perspective, the main task for IPOL is to analyze and provide solid building blocks. Researchers will be able to use these blocks to build more ambitious processing chains, and acquire good habits, such as using correct interpolation methods, always estimate the noise to adjust the parameters, etc. If this plan succeeds, new kinds of research articles should appear, focusing on the analysis of building blocks or on discussed and proven incremental improvements, then a more ambitious step with real processing chains.

1.3 Thesis Summary

In chapter 2, we begin with an overview of Image Processing On Line (IPOL), the framework project in which most of our research was implemented. IPOL is a research journal and facility. As a journal, IPOL publishes image processing and image analysis algorithms, described in accurate literary form, coupled with software implementations. The implementation is reviewed and published, like a "traditional" manuscript. Moreover, IPOL provides an online demo interface to freely try and test published algorithms on user-submitted data, with a public archive used to assess the behavior of the program on a large collection of input data and settings.

In chapter 3, after a short history of the Web from the computer science point of view, we discuss its use as an interface for image processing research software, and its technical merits and constraints. On the issue of portability, we compare two solutions, one based on source code and standard compliance, and another using a virtualization technology. We close this chapter with a discussion about the importance of software implementations in applied mathematics.

Chapter 4 focuses on software quality. We present a set of software guidelines in use for IPOL and explain them. They cover three aspects of a software as a published material: its packaging, its reliability and its documentation. These guidelines are expanded with future plans for automated quality testing and compilation.

Then, in chapter 5, we explore the legal conditions and consequences of publishing research articles with software, demos and archives. The fist part of the chapter summarizes two important but disjoint legal concepts, applied to software: copyright and patents. Then we propose a complete copyright and license policy for an online research journal with

software material, maximizing the usefulness of the publications for authors and readers. We close the chapter with a review of the regulation of the Internet in France, and its consequences on the operation of a journal as a web site.

In chapter 6, we review the image processing and computer vision terminology, and use it to draw a scientific program for IPOL. The chapter continues with the decomposition of this research program and logical groups linked to every step of the image acquisition process.

Chapter 7 contains two examples of articles published in IPOL. One is an implementation of the Retinex algorithm by means of a Poisson equation, the other is an extremely basic method to achieve a simple color balance, used for comparisons with more sophisticated methods. These two examples are completed with explanations about how the software and online demos were designed.

Finally, we show in chapter 8 the results of a survey performed on the IPOL authors after the first year of activity. This survey shows the interest of the project for authors and the priorities for future developments. Usage statistics are also collected and summarized, and they provide an estimation of the popularity and usefulness of the project for users.

The thesis ends with two documents included in annexes (the official IPOL Software Guidelines, and the questions of the author survey) and the list of references used for this work.

Chapter 2

IPOL Project Overview

Contents

Abstract

Image Processing On Line (IPOL) publishes image processing and image analysis algorithms, described in accurate literary form, coupled with code. It allows scientists to check directly the published algorithms online by providing a web execution interface on any uploaded image.

This installation acts the universality of image science. It permits to transcend the artificial segmentation of the research community in groups using this or that image software, or working on dedicated incompatible image formats. It promotes reproducible research, and the establishment of a state of the art verifiable by all, and on any image.

In this chapter, we describe the technical challenges raised by the foundation of this new kind of journal and its scientific evaluation issues. We finally analyze the first publications, to demonstrate its potential impact on the development of image science.

This chapter is adapted from the article submitted to the Elsevier Executable Paper Grand Challenge [110] and published in the proceedings of the ICCS conference 2011 [227].

2.1 Why Image Processing on Line?

The goal of the "Image Processing On Line" (IPOL) [185] initiative is to publish complete and certified implementations together with the precise algorithm description, submitted to a peer-review. This should enable performance and quality evaluations and comparisons between algorithms, a task difficult to achieve today because implementations are often missing or unreliable. Making implementations into reviewed and published material will reward the software quality, which otherwise is neglected by paper publications.

2.1.1 On-line Testing and Experiment Sharing

Despite the availability of a reliable source code, an algorithm may not be immediately usable because its compilation, installation and use is not straightforward. Many researchers are reluctant to get into a compilation procedure to check an algorithm. Most would prefer a quick test before they consider spending some more to study the article in depth. Our proposed solution is to provide an experimentat environment directly accessible over the network.

Despite various individual initiatives for web-based image processing, some as early as 1994[1], dozens of on-line photo editing and sharing services demonstrating the availability of the tools, industrial solutions being developed by large organizations for remote data analysis[2], and recent projects ensuring a reliable experimental environment with cloud infrastructure [283], experimental resources accessed over the network is a burgeoning concept in the image processing research community.

The Middlebury initiative [27] is an attempt to compare stereo vision and optical algorithms. Multiple codes process identical datasets and measures are collected from the results to compare the algorithms. The computer vision community has also made some effort to create experimental databases in "challenges" open to all researchers with no

[1] See Rolf Henkel's "Online Imageprocessing Pages" in chapter 1.

[2] The European Space Agency (ESA) develops KAOS as part of an infrastructure used for the analysis of satellite data using remote servers (`http://keo-karisma.esrin.esa.int/keo-home/KAOS.html`).

time limit: the Berkeley Segmentation Dataset and Benchmark[3] [239] and the PASCAL Visual Object Classes[4] [119], otherwise called the Pascal Object Recognition challenge, are initiatives structuring the vision research. But these initiatives do not contemplate the free interactive online execution of algorithms.

Some initiatives propose an online execution of the code, in particular in stereo vision, like Minh Nguyen's Web-Based Stereo Vision[5] [263], but they are isolated projects from one researcher and do not provide the possibility to compare algorithms and, like previously cited benchmarks and challenges, there is no peer review policy to validate the implementations as a scientific work. Out of the image processing community, the Mathematical Programming Computation journal [257] requires the authors to provide the code with their article, and technical editors will try to reproduce the results announced in the paper, but the implementation is only archived and not distributed with the article. A new journal, Open Research Computation [282], is promising: it explicitly focuses on the documentation and testing of open source research software. But this recent initiative has not released its first issue yet. Nevertheless, this is far behind other research fields such as the large on-line databases and research tools used in genetics[6].

We intend to remedy to this and the consequent lack of experiment sharing in computational sciences by providing a web-based test interface for all the algorithms published on IPOL, allowing any researcher to freely test any implementation on any data. The experimental data is archived and publicly accessible. This archive can be used to share experiments between researchers, and it is an efficient mean to assess the performances of an algorithm over a large collection of input images.

2.1.2 The IPOL Publishing Model

We devised a new model to evaluate, preserve and disseminate research. IPOL is now a solution for reproducible research, executable algorithms and experiment sharing, publicly available on `http://www.ipol.im/`:

documentation Articles are published as web pages with embedded formulas and figures, attached data and a portable minimal implementation. Their main content is the precise description of an algorithm.

implementation The implementation is used to evaluate the algorithm by careful examination of the source code and careful testing.

demonstration An on-line demonstration system is proposed to run the algorithm on a server over a web interface, with freely uploaded input data.

archive The experiments conducted with the on-line demonstration system are collected into a publicly available archive.

Similar in its form to a classic research article but focused on the implementation, the precise description of the algorithm at IPOL is the main part of the publication. It must be detailed enough to allow any specialist to implement it in their own programming

[3] `http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/`

[4] `http://pascallin.ecs.soton.ac.uk/challenges/VOC/`

[5] `http://www.cs.auckland.ac.nz/~mngu012/stereoapplication/`

[6] The National Center for Biotechnology Information (NCBI) maintains various databases and tools (`http://www.ncbi.nlm.nih.gov/`). The Science Commons project develops knowledge management systems for biomedical research. (`http://sciencecommons.org/`).

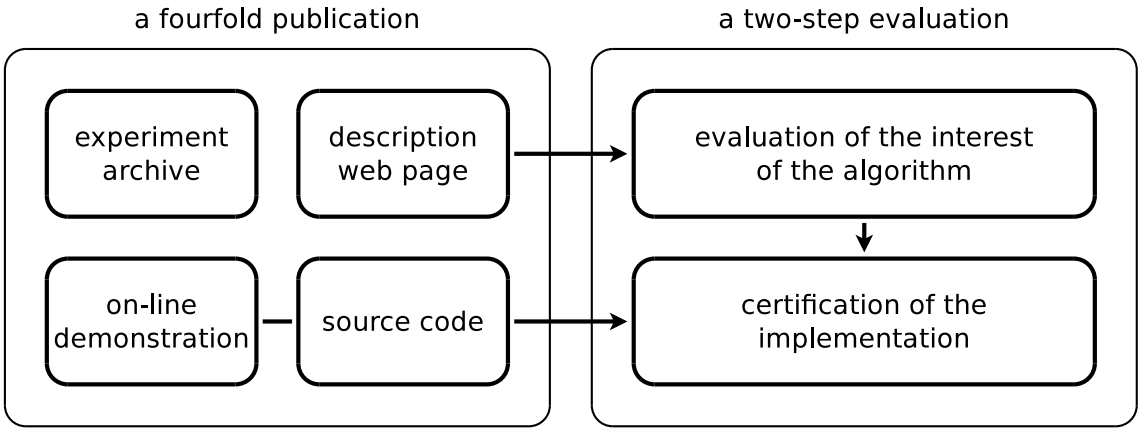


Figure 2.1: IPOL content and evaluation process.

language and environment and validate it by comparison with the IPOL code and demo. IPOL aims at publishing the most accurate descriptions of existing or new algorithms, with implementations as generic and as reliable as possible. The implementation is guaranteed by the peer-reviewed evaluation to match accurately the mathematical description of the algorithm. It runs on line to grant researchers immediate testing.

The role of referees is different from (and complementary to) to a classic journal. An IPOL article evaluation runs in two steps. At the first stage, referees only have to read the submitted web page, which must contain an accurate algorithm description and the most significant experiments. If the referees are positive the article is "conditionally accepted" and the authors are invited to submit a code, carefully commented and linked to the algorithmic description.

The "conditional acceptation" criterion is not the novelty of the algorithm: in many cases the submission develops an already published algorithm, or an algorithm simultaneously submitted to a classic paper journal. The acceptation criterion is merely the interest for the community to have the proposed algorithm specified, implemented and available on line. The second phase of the evaluation is more technical and concludes with a certification by the referees. The algorithm must be completely described, in such a way that it is reproducible. The code and online demo must implement exactly the algorithm.

The on-line demonstration system and its archive are maintained by the IPOL editorial board, and each new demonstration is created and installed by the board according to the specifications given by the algorithm author, using the exact same implementation available on the algorithm web page.

2.1.3 The Potential Impact on the Field

There are some twenty image processing and image analysis journals and an equivalent number of international conferences with more than 4000 articles every year in the community. There is therefore a desperate need for a state of the art permitting to compare and evaluate the methods. The goal of IPOL is not only to favor a fair evaluation of new methods, but also to establish a comparative state of the art by publishing in this new form all relevant algorithms of the field. Thus, all relevant algorithms will be *republished* in IPOL.

The potential impact of IPOL on future image processing and computer vision research can be summarized in the following points:

1. establish a state of the art with benchmarks comparing the major algorithms;
2. certify a correspondence between paper, algorithm and code;
3. publish benchmark data with certified properties;
4. fix the form of classic algorithms, diminishing the time wasted by researchers on reprogramming them;
5. reward the authors for this invaluable service by a peer reviewed publication.

2.2 How IPOL Works

2.2.1 Native Web Content

Most scientific publishers produce electronic versions of printed articles (generally PDF files), distributed *via* the network. This still limits the authors to the model of printed articles and implies that any non-printable content such as data and source code is handled aside, sometimes in an on-line repository provided by the publisher, often simply on the authors' personal web space.

We prefer to publish articles *in* the network as native web pages including all the material, source code, data, files and figures required for the full evaluation of an algorithm. The Word Wide Web as a publishing and documentation model [44] provides means to distribute a document with structured text, tables, figures and any attached file, without the artificial size limits imposed in printed journals. Document layout and interaction support is reliable across major browsers for all the base technologies (content, presentation, interaction, identification) and new standards are actively developed and adopted (video, math formula, vector graphics)[7]. IPOL currently uses the Ikiwiki[8] software as a document publishing system. Other solutions are possible, from generic web content management systems to industry-grade publishing chains. The document publishing system for IPOL was chosen to be a simple and immediately available solution without in-house development.

In the close future, however, we will probably transition to a process based on LaTeX and PDF documents. This orientation was needed because the authors are used to base their work on LaTeX documents and they should not need to get used to a new medium in order to access to IPOL as authors. It was also motivated by the unsatisfying quality of the typesetting and rendering of scientific documents over the web interface. Some interactivity and integration will be lost in the transition, but we hope to achieve a better publishing quality in exchange. We will try to maintain an HTML version of the articles

[7] Internet Explorer 7 (released in October 2006), Mozilla Firefox 2 (released in October 2006), Opera 9 (released in June 2006) and Safari 2 (released in April 2005) all had at least a reasonable support for the content description (HTML4), presentation (Cascaded Style Sheets — CSS) and user-side interaction (JavaScript) standards. This is expanding since with native video (HTML5 <`video`> tag), math formula (MathML), vector graphics (Scalable Vector Graphics — SVG, HTML5 <`canvas`> tag) and 3D rendering (webGL) standardized or expected soon. All this content lives in the same document space (Document Object Model — DOM) and can be interconnected for a rich interactive experience. The article and its components can be identified by unique identifiers (Digital Object Identifiers — DOI) for reliable references, citations and interlinks.

[8] Ikiwiki is a wiki compiler (`http://ikiwiki.info/`).

in the web pages of the journal and we will still publish non-text materials (code, datasets, ...) together with the article text.

2.2.2 Standard-Compliant Portable Compiled Code

Algorithms implemented for IPOL must be usable on any major computing system. Cross-platform compatibility is required at least for the Windows, Mac OS X and Linux/Unix families of operating systems, and with 32 and 64 bits variants of the x386 processor line. This is a bare minimum, and in fact we don't expect the implementations to be tied to any operating system or hardware architecture.

Process virtual machines (Java, .NET) or interpreted languages (Perl, Python) could be a mean to achieve this portability, but this solution is not realistic because it would imply sub-efficient implementations, as shown by programming language benchmarks on computationally intensive algorithms[9]. Moreover, such programming languages would be an obstacle to practical exchange and re-use of the implementations because they would not be easily merged with other code without sacrificing performance. The same compatibility issues would be faced with proprietary scientific computing environments, which in addition don't provide any assurance of the future usability of the implementations. For these reasons, we consider that the only way to ensure efficient portability is to require the algorithms to be implemented in a low-level compiled language. C, C++, Fortran are some examples of programming languages widely used in the research community, adapted to intensive numerical computations, and supported on all common computing architectures and operating systems.

Algorithms exposed in IPOL articles must still be usable many years later to ensure long-term verifiability of the scientific results. For this reason, implementations are required to follow an established standardized language specification (ANSI C89, C99, ISO C++98, Fortran 90, ...). We use static code analysis and strict compilation as a "best effort" procedure to test source code for future compatibility. The portability applies to all the code: no OS specific feature can be expected from the standard library and the programs will only interact via the only portable interface, the command line interface. For the same reasons, a perfect implementation would require no external library and include all the source code needed to produce the executable program. For practical reasons, exceptions are granted for well-known reliable and portable software libraries: IPOL currently allows the libpng and libtiff libraries to handle the image file input/output, the FFTW library for Fourier transforms, and the BLAS API and LAPACK library for linear algebra[10].

This may seem very restrictive but so far all the algorithms published or in the publishing process have been able to follow these rules or to find workarounds. Many interesting and powerful image processing algorithms can be implemented without requiring the support of external libraries, but this may not be the case in other research fields where this policy would need to be revised.

[9] The "Computer Language Benchmarks Game" is an exhaustive and constantly updated benchmark over many languages and typical algorithms (`http://shootout.alioth.debian.org/`).

[10] libpng is the reference implementation of the PNG image file format (`http://www.libpng.org/pub/png/libpng.html`). libtiff is the *de facto* reference implementation of the TIFF image file format (`http://www.remotesensing.org/libtiff/`). FFTW is a high-performance and portable library for discrete Fourier transforms (`http://www.fftw.org/`). BLAS (Basic Linear Algebra Subsystem) is the specification of elementary linear algebra libraries (`http://www.netlib.org/blas/`). LAPACK (Linear Algebra PACKage) is a library for linear algebra built on BLAS (`http://www.netlib.org/lapack/`).

2.2.3 Web Execution Interface

IPOL also provides a demonstration system for every algorithm. Client-side models were excluded because, as discussed before, the solutions for a portable executable program are currently not efficient and would not show the best performances of the algorithms[11]. We preferred server-side hosted demonstrations to avoid compatibility problems and ensure consistent optimal performance in a controlled experimental environment with high-performance hardware resources. We chose to make this system accessible over a web interface because of the ubiquity of web browsers and their use as a flexible interface to any resource over the networks, with web-based applications existing since 1993 [297].

This interface is not optimal for our needs. Among other problems, we can cite the lack of a built-in persistence mechanism in the HTTP protocol [126] to keep track of the history of the user interactions and provide an "undo" option, and the very limited feature set of XHTML controls [360] (buttons, text fields, ...) compared to what is offered from any desktop graphical user interface. On the other side, a web interface has the immense advantage over other client-server models to be immediately usable for anyone with a computer connected to the network and a browser. It also is a well tested model with a 20 years history and a large collection of tools, servers, frameworks and libraries.

The typical IPOL experiment work-flow is:

1. Upload an input image or select one from a default list. The uploaded images are converted from any common format into the file format and image type expected by the algorithm implementation.
2. Set some parameters and/or perform some pre-processing. The pre-processing tools currently available include some image editing, cropping/zooming and adding noise.
3. Execute the algorithm on the server and visualize the results. Experiments conducted with original uploaded images are archived and publicly available if agreed by the uploader.

IPOL uses native HTTP server mechanisms for data transfer, user authentication, logging and usage statistics tracking. An "experiment key" is used to identify the data set and provide continuity across the successive pages required to run the experiment. Simple XHTML form controls are sufficient for most of the demonstration interfaces and some JavaScript tools have been developed when usability improvements are needed.

We chose to use the Python language and CherryPy framework[12] to implement the server-side back-end system whose tasks are to receive and store the input images and then execute the algorithm using the implementation with some optional pre-processing dialogs. The communication between users and the server is made of HTTP requests sent by the web browser as a result of their local actions and the answers received from the server, formatted as XHTML pages. Our current system can be extended for new algorithms by reusing common building blocks from an application library. Once this young software stabilizes, it will be released to help the creation of other similar initiatives.

So far, network capacity and file size are not an issue, mainly because image files are relatively light and because a limit is imposed on the execution time. A web user is usually

[11] An increasing part of the network traffic comes now from mobile and handheld devices with a very limited processing power.

[12] CherryPy is a light framework for web applications written in Python — `http://www.cherrypy.org/`.

not ready to wait for more than 10 seconds until a page is available [265, chapter 5]. In our context, we estimate that researchers can wait for 30 seconds during the execution of an algorithm if they receive a correct feedback. In order to keep the execution time under 30 seconds, the size of the input images is limited by cropped or resized large images, depending on the kind of algorithm. This effectively limits the volume of data exchanged and the bandwidth requirements. We expect the situation to keep balanced in the future with network capacity to grow together with hardware performances and typical file sizes.

Larger data or more computationally demanding algorithms could also be processed on the server in a non-interactive processing queue, as it is currently done by some other demonstration tools:

1. upload some data or provide a way to retrieve it from the server and set the execution parameters;
2. the data is enqueued with instructions about how to process it;
3. the algorithm is run when some processing resources are available;
4. a notification email is sent to the user with the instructions needed to access the result of the algorithm.

2.2.4 Open Access and Free Licenses

IPOL subscribes to the Open Access principles and all its content is freely accessible. The copyright is kept by the authors on all their contributed content, allowing them to re-use their works at will. In accordance with Stodden's *Principle of Scientific Licensing* [326], dissemination, sharing, use, and re-use of the scientific research published by IPOL is facilitated by the use of Creative Commons licenses for the articles and Free Software licenses for the implementations when possible[13].

2.2.5 Security and Legal Context

IPOL publishes articles in the form of web pages and as such, the published content must be moderated to avoid an abuse of the service. This is achieved by identifying the authors by their name, institution and a verified contact address. The creation of any new article on IPOL requires a manual confirmation by the editorial board, and new articles are not publicly accessible until they have been reviewed and "conditionally accepted". IPOL authors are required to credit the authors republished algorithms and the origin of all the software components they re-use. Potential plagiarism issues are the same as for any scientific journal. IPOL as a web site can also be the target of external attacks, but this can be handled by generic system and network administration measures.

The demonstration system is more exposed because it provides server resources to the public by running software provided by the authors with data coming from the users. Different attacks are possible: the algorithm implementations could use all the computing resources, they could be malicious or have remotely exploitable bugs[14] and damage the

[13] Creative Commons licenses allow the licensed content to be re-used as long as the original author is credited — `http://creativecommons.org`. Free Software licenses allow unrestricted use, study and modification of a software with minimal restrictions. IPOL encourages the use of the GPL or BSD licenses.

[14] For example, a bug in the image reading library used by some demonstrations could be exploited by uploading specially crafted images to execute arbitrary code on the server.

server or use it to attack other servers. This exposure is limited by the careful examination of the source code during the review process, the restriction of the rights of the demonstration system on the server and of the total execution time a program can use, and the network isolation of this server. Virtual machines and operating system containers are considered to execute the demonstrations in an isolated environment without noticeable performance penalties.

Experiment archives could also be abused and unlawful images inserted. So far this has not been an issue, and the use of the demonstrations could be rate-limited or require a preliminary identification. The French law [3] (so far, IPOL operates from France) exempts IPOL as the on-line editor from liability for the archive content if it is explicitly not moderated and if unlawful content is promptly removed after notification.

2.3 Current Activity

The first IPOL prototypes were put on-line in December 2008 and the current version dates from December 2009 for the web publishing part and September 2010 for the demonstration server. The first article was reviewed, approved and published in July 2010. IPOL had in March 2011 a dozen algorithms published or in the final review stages and twice more in the pipeline. More than 50 international academic experts compose the scientific committee responsible for the review process, and the editorial board handling the new algorithms. A recent agreement between IPOL and the SIAM Journal of Imaging Sciences (SIIMS) [319] encourages authors to submit *simultaneously* algorithm and code to IPOL and the corresponding theory to SIIMS. As of January 2012, every month, the algorithm demonstrations are used for more than 3000 experiments. 33500 original experimental data have been collected so far in the archives.

At this point IPOL is still experimental, and faces new challenges with almost every new algorithm publication. The delicate question of how an image processing algorithm can be explained, specified and illustrated on line is the object of many trials before some satisfactory solution is reached. Once a satisfactory on line demo solution is found for a particular image processing problem, it is copy-pasted for all further submissions dealing with the same problem. Thus, the editing workload seems to be controllable. It has led to the constitution of a fifteen members editorial board to assist authors. This board is distinct from the scientific board. Exposing an algorithm to the public and allowing it to be applied on any data is not a small challenge. It often requires from the authors a serious rethinking, rewriting and many trials. This explains why the publication of even classic algorithms ends up in authentic and innovative research.

2.3.1 Published Algorithms

We briefly present hereafter some of the algorithms available in IPOL. These are only seeds of what IPOL can become if the research community subscribes to this publishing model.

Algebraic Lens Distortion Model Estimation [21] This algorithm presents a new method for correcting optical camera distortion (fig. 2.2). It is the first article evaluated and published by IPOL. The corresponding paper article was previously published in a 2009 article [209] and counts so far eight known citations as of January

Figure 2.2: Algebraic Lens Distortion Model Estimation.

Figure 2.3: Non-local Means Denoising.

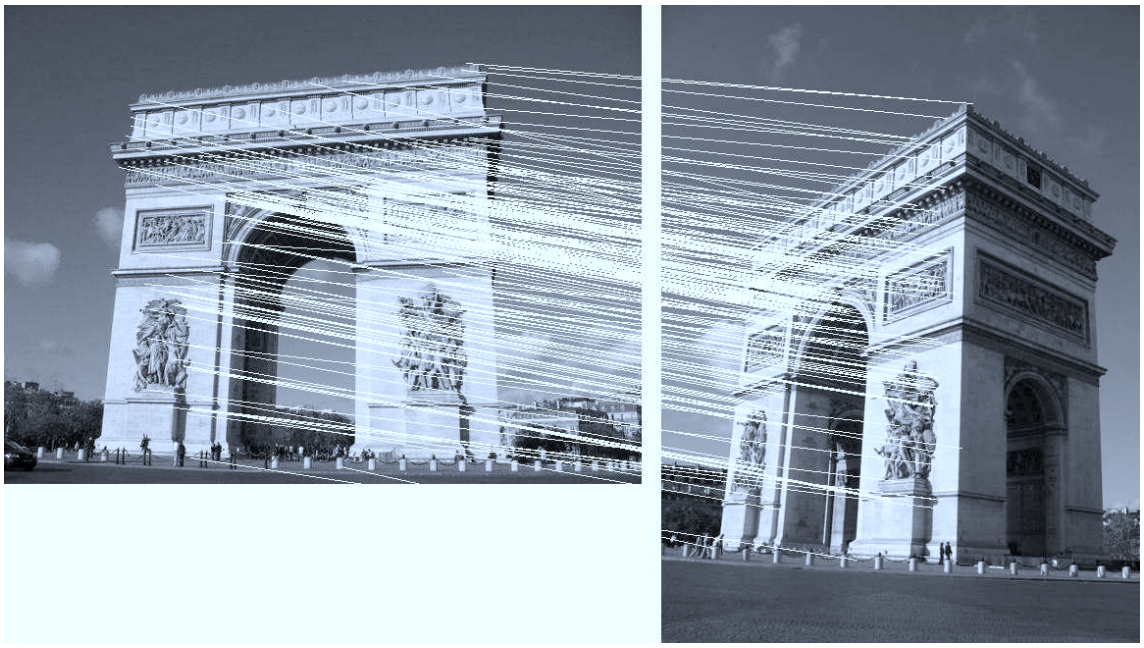


Figure 2.4: ASIFT: An Algorithm for Fully Affine Invariant Comparison.

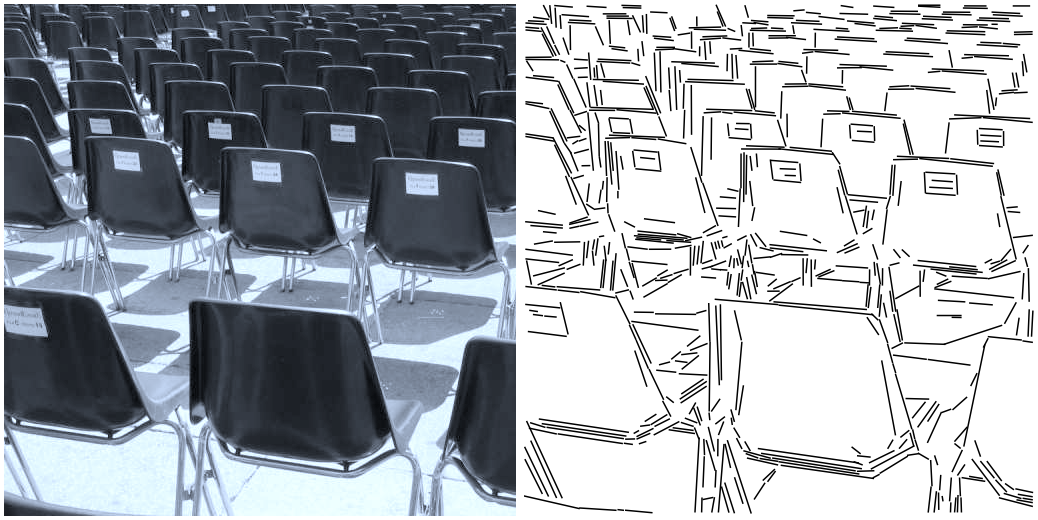


Figure 2.5: LSD: a Line Segment Detector.

2012. In contrast, its IPOL archive shows 1400 on line experiments with original data between May 2010 and January 2012. Its source code was downloaded 350 times in the same period of time.

Non-local Means Denoising [63] Any digital image taken by a camera needs a sequence of restoration operations before it is delivered in visible form. Denoising is the key operation which conditions the quality of the other ones. Non-local means (fig 2.3) was proposed in a 2005 article [18], referenced 550 times until January 2012. From its appearance in IPOL in November 2009 to January 2012, the algorithm has been tested on 3000 uploaded images and the code has been downloaded more than 550 times.

ASIFT: An Algorithm for Fully Affine Invariant Comparison [377] This on-line article is the flagship of IPOL, with 13500 on line experiments on original image pairs (fig. 2.4) as of January 2012. The reason for the interest is obvious: ASIFT is an improvement of the classic SIFT method published in 2004 [234] used to detect whether two digital images have objects in common or not. This problem is the core of the image retrieval problem, it is the first stage of many robotic and scene reconstruction projects, and it is systematically used for photo stitching. Hence the need for a standard and rigorous reference implementation. Although ASIFT only is a first approximation to that, the code has been downloaded more that 2000 times in two years between its publication and January 2012.

LSD: a Line Segment Detector [158] Probably the most puzzling publication is this one (fig. 2.5), because of its obvious impact demonstrated by 6500 on-line experiments in 18 months and more than 300 code downloads as of January 2012. In contrast, the 2009 journal paper [157] is so far cited 47 times! One explanation is that, while detecting segments or edges in an image has been a hot topic in the eighties and nineties, it is not currently considered a research subject in the computer vision community, and no new paper appears on it. Nevertheless, most scientific and technical imaging techniques require reliable segment detectors. The immense variety of the on-line experiments proves it. This also demonstrates that an on-line demo can characterize much better the impact on technology and society of a research than citation indexes do.

2.4 The Scientific Program

The hypothesis underlying IPOL is the *universality of image science*. Images currently created by mankind are incredibly diverse. They can be snapshots of current life and artworks for the wide public. Astronomers and geographers produce images of the universe, the Sun, the Earth and other planets in many wavelengths and resolutions. Material science and physics have an immense diversity of scientific samples. Last but not least medicine and biology create images of all living organisms in an almost infinite scale from organs to proteins. Human perception is prepared to this diversity, and seems to be able to adapt quickly to any new kind of image. The existence of universal visual primitives, of "perception atoms" is hypothesized in neurophysiology (where it leads to a cartography of visual areas by their geometric retinotopic primitives), in psychophysics, where it leads to experiments with abstract images exciting these visual primitives, and finally in computer vision where the goal is to emulate them numerically.

In this universal context of imaging science the existence of incompatible image formats and software is highly counterproductive. While some very specific imaging tasks might

indeed require peculiar dedicated algorithms, computer vision and image processing are highly successful in designing algorithms valid for *any kind of images*. This is the case for two of the first published algorithms in IPOL, LSD (which extracts segments in any image) and ASIFT (which can recognize any solid shape, from galaxies to logos.)

This is why IPOL will encourage researchers to publish and confront systematically existing generic image processing algorithms to new ones. In 2011–2012 at least four benchmark publications should appear in IPOL. One will deal with image denoising and will confront the state of the art methods. Other benchmarks will deal with operations as universal and basic as zooms in and out, color demosaicking, contrast adjustment, camera blur estimation. If everything goes according to plan, all researchers using images (not necessarily image processing specialists) should soon be able to test most state of the art algorithms on their own images.

Chapter 3

Online Demos and Software Journals

Contents

Abstract

This chapter starts with a recap of the evolution of computer networks, until the WorldWideWeb was invented and usable to publish a collection of interlinked and structured documents, and to query remote computer resources and services. Then we review how an usable interface to image processing research programs can be built into a Web environment, and where this environment imposes some restrictions of what can be achieved. An alternative model, based on virtual machines, is also presented with its advantages and drawbacks. We end with a discussion about how computational sciences are neither deductive nor experimental sciences, and why the availability of usable software implementations of this research is important for the validation of research results.

3.1 From Hypertext Microfilms to Web Services

Early communication networks were built and developed for the needs of state administration and business organization [176, 322], but the computer networks and applications we know today as the Internet the Web were created by and for academic and scientific uses [19, 151, 163, 311]. We briefly retrace hereafter this history of computer networks during the 20th century and their evolution as a medium for academic communication and as an interface to computing resources.

3.1.1 Hypertext

In 1945, Vannevar Bush describes what he believes the future can be for scientists [70]. He extrapolates from the technology available at this time, facsimile, photocells, television, microphotography, vocoders, early computing and information processing machines, and draws the picture of a future system to archive, index and share the scientific knowledge. This is the description of an information database with hypertext navigation in and between documents, using analog technology, mechanical actuators and physical medium. He describes this system as a possible revolution of the methods of transmitting and reviewing the results of research, a solution to the unacceptable delays imposed to scientific communication by printing technology and economy.

3.1.2 Artificial Intelligence

Fifteen years later, in 1960, the computer science and industry had expanded and research on artificial intelligence was burgeoning with high hopes. The updated vision, carried by Joseph Licklider, was to let computers help men think and model problems. In this dynamic *cooperation*, efficient but constrained computers would do the repetitive tasks and assist slow but flexible men in taking technical and scientific decisions [224].

Towards the end of the decade, in 1967, studies of the man-computer interaction assumed the use of a computer to test a model in a scientific application. The author of the studies described this interaction as a succession of tests to validate or infirm hypotheses, a process similar to the experimental method involved in testing a scientific theory [74].

In 1968, Douglas Engelbart and his team presented their *oN-Line System* collaborative computer environment [112]. With the famous first use of a mouse, the historic demo included the first working hypertext system, a word processing used to produce a conference

paper via collaborative edition. These developments aimed at enhancing the knowledge, productivity, and, ultimately, the human intellect by computer-assisted collaboration tools.

At this time, the only networked computing experience was still limited to distant operation of mainframe computers from a remote terminal. The user interface (keyboard and display) could be kilometers away from the computer and different users could use a computer simultaneously, but each mainframe/terminal set had its own idiosyncrasies and there was no machine-to-machine network.

3.1.3 Networks

The first computer interconnection tests appear in 1968[1] and the ARPANET experiments would start from 1969 [163, 164, 380]. In addition to its previous function, the computer is now a communication device. From the beginning of the transition, computer-assisted communication is more than the consultation of some distant documentation. The vision of future networks is interactive, creative information processing systems linking people to people and people to resources [225].

The ARPANET is built, and two of the first four connected nodes were research centers working on image processing and interactive graphics [163], and later five the fifteen nodes ARPANET doing research on graphics [19]. During one decade, this network expands, other networks are created with other goals, other technologies and in other countries, and from 1982 they gradually interconnect[2] and adopt a common standard, the Internet Protocol Suite. From now, there is one global worldwide computer network, the base resource required to realize knowledge society described by Bush, Engelbart and Licklider.

3.1.4 Internet and the Web

In 1988, the National Research Council report "*Towards a National Research Network*" shows how scientists use the network and how they intend to expand this usage [198]. Distant collaborators access a shared software or sensor, and work in a common experimental environment from different remote locations. Computing tasks are distributed between the local workstation front-end and the remote supercomputer for the heavy computations. Researchers expect electronic transmission of text, images, and movies to replace the distribution of scientific. And image processing researchers want to transfer large files in a short time.

The next report "*Realizing the Information Future*" confirms in 1994 the realization of these ambitions [199]. In addition to the usages already described, independent electronic preprint repositories are developed and used as an alternative to expensive journals. Interactive interfaces are available to query genetic or satellite image databases. Forecasts include distributed research teams, and computing grids. It is now clear that this is just the beginning of the revolutionary impact on knowledge societies of what Brian Kernighan will describe 15 years later as "*the universals of digital technology*": a universal binary

[1] The first packet-switching networks were probably built for the UK National Physical Laboratories and for the Société Internationale de Télécommunications Aéronautiques. By design, they were horizontal communication between peer machines, instead of the vertical client/server model of the terminal and mainframe connection.

[2] Interconnections happened before 1982, but they used different protocols and were gradually replaced by the Internet model.

representation of the information, universal (reprogrammable) information processing machines, and the universal (ubiquitous and content-agnostic) network [195].

Between those two reports, the invention of the WorldWideWeb took the Internet by storm in 1991. This invention aims at merging different kinds of information stored on a collection of machines into a single unified model and interface [42,44]. In the early catalogs and usage guides of the Internet published as books at this time [194,208], we can see long lists of network resources, such as multiple FTP servers to visit and to retrieve specialized documents or numerous telnet servers to be used for interactive services and databases, and all these servers came with their own characteristics, syntax and usage models. With an identifier (the URL), a network protocol (the Hypertext Transport Protocol, HTTP), a document language (the Hypertext Modeling Language, HTML) and a user interface (the browser), the Web "*converts every information system so that it looks like part of some imaginary information system which everyone can read*" [41]. It was the environment needed to publish and organize all the information available on the network, including all the scientific knowledge to be shared. Once the browser gained graphical abilities and a critical mass attained, the Wed started its exponential growth from 1995 [261].

3.1.5 Web User Interface

But one brick was still missing. Internet usages were still split between the retrieval and consultation of static documents, via the Web, and the usage of dynamic accessed via telnet connexions. Early definitions of the HTML language in 1993 [38, 39] only contains tags related to the document structure and provide no support for on-demand information processing.

In 1995, the first official definition of HTML includes the <`form`> tags, used to used to "*access an information service as a function of the action and method*" [40]. This, together with the Common Gateway Interface (CGI) specification [244], creates the dynamic web services.

CGI are programs used to create a Web page on demand, from the parameters provided by the visitor. Without CGI, the Web was a system to publish organized but static documents. With CGI, the web server becomes a pluggable dynamic tool to process user requests into web pages. The Web becomes a new computing environment model, where the user interface is the browser and the computing power is on a remote server. Graphical user interfaces existed before, but only for local programs. And remote services were accessed via a console, text-based telnet link. The Web User Interface (WUI) adds a cross-platform and lightweight graphical user interface to remote computing.

Since its invention, the CGI technique evolved into more efficient models like the FastCGI variant [62], scripting language, modules and frameworks[3], and the web page design standards were updated and enriched, but the principles remain the same: the users actions are HTTP requests submitted via HTML forms [40, 108] and the server answers are web pages composed from these requests. This implies three major restrictions to this programming model: the user interface is limited to the content viewable in a web browser, two network transfers and one server-side computation happen between every user request and its result, and the user-server communication uses the HTTP protocol.

[3]PHP, Coldfusion or Active Server Pages are programming languages developed for dynamic web applications. Perl, Python, and Ruby are general-purpose scripting languages popular for web applications.

An alternative to server-side processing appeared from the 2000s: client-side portable programs distributed via web pages and executed in the web browser, such as Java applet and Flash plugins, and more recently JavaScript programs. The major difference between this approach and the previous CGI model is that the performances and reliability of client-side web programs depend on the client configuration, both hardware (computer class and generation) and software (operating system and browser models and versions), notwithstanding the intrinsic performances and capacities of Adobe Flash, Java and JavaScript programs.

3.2 Online Demos

In our context, the generic concept of an online demo is a web page or a set of web pages where one can interactively specify the input data and some options and obtain the result of this input after some server-side processing.

Sometimes, the "*demo*" term is also used for examples of the action or result of an algorithm, distributed as images or video. These are just examples, chosen by the authors. These examples may be useful to communicate about the algorithm, but they are not more than advertisement and do not contribute much to the understanding of the algorithm.

3.2.1 Use Cases

Online demos, using the Web or another network technology to provide the interactive experience of a program for testing and exploration, can be useful for collaborative research, development and debugging, for journal reviews, easy access to the program, and as a reference for comparisons.

Workshops When investigating a given topic in computational research, we may want to try different algorithms with different settings and data and compare the results obtained. When this research is conducted by a geographically distributed group, the information needs to be shared through network tools. Instead of exchanging multiple versions of their source code, programs and data, researchers can use a private online demo to conduct their tests and experiments and collect the results in a single location.

Debugging Between the first working version and the public release of an algorithm, a code usually needs to be stress-tested to find and correct defects and unexpected behaviors triggered by rare data. With an online demo, a researcher can propose a program to its collaborators without needing, at this early stage, to worry about the code quality, portability or version management.

Reviews In the limited time they have for the task, academic reviewers will not always be able to get, compile, install and use the implementation provided by an author with a submitted article. With an online demo, and with the appropriate provisions for anonymity, the reviewers can validate the author's claims, reproduce the results included in the article and try counter-examples.

Easy Access If an online demo is published together with a research article, then the readers can immediately try and see the demo as a complement to the article. Nothing has to be installed, compiled or even downloaded, a demo is immediately accessible. This is useful for the readers, and informing their research community is easier for the authors.

Reference Famous algorithms can suffer from approximations and mistakes made by others in its description or implementation. With an online demo, the original authors can publish their reference implementation and have it used in comparisons and benchmarks.

3.2.2 Interface

The interface of a demo is critical, because it defines what can and cannot be achieved in term of user interaction. We can consider three models for the network access of an online demo: remote console, batch processing and client/server.

Remote Console The first possible architecture would be based on the old telnet model: with a light software console and a network link, one connects to a remote server where the online demo lives. The local console only provides an mean to access the demo, used via command-line or text mode interface.

The advantage of this system is the simple implementation and deployment, because it only requires a console access to already existing programs. However, it is rather limited: the text medium is too poor for demos on image processing, there is no file upload once we are connected to the remote console, and few people are used to the console environment.

Batch Processing Another possibility is to send some data and requests to a demo server, and receive later the results of the execution of the algorithm. This could be achieved, for example, with e-mail messages containing instructions (input, parameters) encoded in a defined syntax in the message body, and input data as attachments.

Batch processing is good to control the load of the server: a request is only handled when some processing resources are available. With this system, we can consider heavy tasks and long processing time. Batch processing is a standard procedure when the resources are scarce, for example in high-performance computing. But this model lacks interactivity. The exploration of a demo by trials and errors will not be possible with this system, and when most queries are very quickly processed, as it is the case for image processing.

Client/Server With a client-server architecture, a user runs a local program (the "client"), and this program will connect to the server, submit the input, trigger the execution of the algorithms, get their results and display them to the user. This implies a communication protocol between the client and the server; various standards exist, such as XML-RPC, Java RMI, REST or SOAP. The client can be a specific software developed for this demo system and similar to any desktop application, or a generic light network client (like the web browser), with all the interface transmitted from the server.

With this model, we get a real interactivity, and there are few limits on the flexibility of the user interface; different client software can even coexist, with different use cases and capacities, and connect to the same server. The drawback is that, unlike the previous

models, it requires more work: the client/server protocol must be defined, and the client interface and the server functions must be developed.

3.2.3 Web Demos

The client/server model seems to provide the best user experience for image processing demos in research. With a careful design, it is also possible to reuse the server side of this demo system for batch processing and use the best of these two models for different talks.

One possibility for client/server systems is to base the system on the web model : in that case, the base client is the web browser and the base server is a web server. On top of it, the interface is built into HTML documents displayed by the browser, and the demo controller is a program connected to the server, to perform the demo actions upon request (see figure 3.1 for a simplified schema).

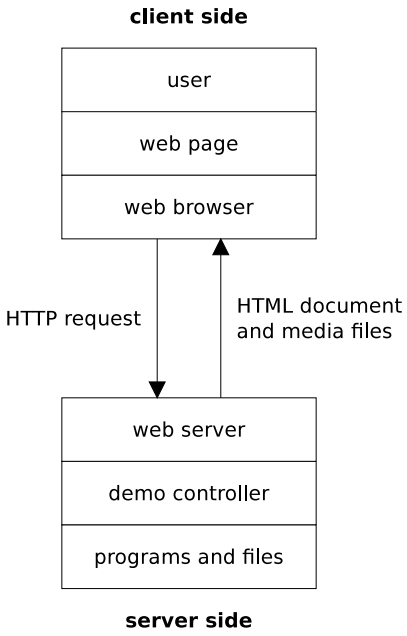


Figure 3.1: Simplified client-server schema for web applications.

The Web has the immense advantage to be widely accessible, for almost any connected user, any computer, any system. Modern web browsers are freely available for major systems (Windows, Mac OS X, Linux) and very specific targets (mobile, for example). On the other side, the Web is made of multiple layered techniques and standards with unequal level of vendor support, and what can be achieved with web browsers depends on the browser models and versions. But a base feature set is available: since a few years (October 2006, IE7), all the major browsers support the basic web layout and interaction standards (HTML 4.01, CSS 1, ECMAScript 3, DOM 1), and this compliance has improved since and gradually includes new versions of the standards.

Despite this risk of cross-browser issues, the ubiquitous HTTP/HTML server/browser seems a truly accessible system. Moreover, the development of the Web during the last 10 years stimulated the development of many languages, techniques and software usable for a demo project.

HTTP

The HTTP protocol was designed to handle data transfer for the Web. A consequence of this origin is its stateless design: the server is not supposed to retain any information about the client between two requests, the answer to each request is unrelated to any previous request, most requests are idempotent, the answer must be completely defined by all the information provided in the request and identical requests will result in identical answers [43, 126].

But most online demos involve multiple steps, which need to be connected in some way for a continuous interactive experience: once a file has been uploaded or chosen, demo users will process this file without needing to upload or select it again. A workaround is needed to connect these steps. It can be a unique identifier, maintained from the beginning to the end of the demo procedure.

HTML

The user-side display interface is a different matter. Visual web is made of a collection of techniques and standards, all interconnected via the HTML family of document description languages. We highlight hereafter some aspects of HTML relevant for online demos.

Bitmap Images HTML standards do not specify which image file format should be supported by graphical web browsers, but a consensus on the essential formats exists: JPEG, GIF and PNG.

JPEG files can contain color images encoded in 8 bits per channel, with three RGB channels. JPEG compression is efficient for photographic images but performs a lossy compression, even with the highest quality settings[4]. The compression errors would affect the precision of the demo output, so JPEG is not adapted to online demos.

GIF files are compressed without loss, but can only contain 256 different colors, a limitation usually solved by quantization and dithering. For this reason, the GIF format is not recommended for online demos.

PNG files can contain color images encoded with three RGB channel, each with up to 16 bits per pixel. Most important, PNG compression is lossless: the pixels values of an image encoded into a PNG file are perfectly restored when the file is read. This property is essential, PNG should the main image format for online demos.

So far, no floating-point image file format (such as TIFF or OpenEXR) can be used in an HTML interface, probably because HTML is made for visual display devices, and screens only handle integer precision, usually with 8 bits per channel.

Vector Images The vector formats commonly used by the scientific community, PostScript (PS) and Portable Document Format (PDF), are not natively embeddable in a web interface. A native alternative is the Scalable Vector Format (SVG), whose browser support is improving. Our tests in December 2012 showed that SVG files are displayed by

[4]Even with the maximum quality level, JPEG encoding involves a chroma downsampling and some rounding errors during the DCT compression.

almost every browser, and that an alternative PNG bitmap can be used for the few ones without SVG capability[5].

Movies, Sound, 3D Current HTML standards don't include a mechanism to embed movies, sound or 3D data. Movies and sounds are usually displayed via an external plug-in software, often Adobe Flash, sometimes Java. The VRML and X3D standards have been developed for 3D scenes, but failed to gain wide adoption. Here again, Flash or Java plug-ins are sometimes used.

It is worth noting that the next version of the HTML standard, HTML5, should solve most of these issues: it includes native audio, video and 3D media in a web page. But as of early 2012, HTML5 support is still very partial and experimental.

Forms and Actions The HTML language provides native interaction controls with the HTML forms. They can be used to upload files, submit a free text option, select between predefined choices, toggle options, and transmit some coordinates selected in an image. Their browser support is very robust and reliable. On the other side, HTML forms are passive: nothing happens until the form is submitted and a new HTML document is requested, produced by the server, transmitted, and displayed by the browser. And they are rather simple: the only action is the mouse clicks. This is a problem when we want, for example, to select an image area: we need to use multiple click, *ie* multiple client-server requests and answers.

JavaScript Adding JavaScript scripts to the web documents can enhance the HTML user interface. With this programming language, interpreted and executed in the browser, one can connect different elements of the user interface, and user input devices can include keyboard, multiple mouse buttons and mouse gestures.

3.2.4 IPOL Demos

In the IPOL journal, the image processing demo system was built on these priorities:

- simple, avoid unnecessary technology layers and software dependencies;
- flexible, we do not know yet which interface feature will be needed for future demos;
- open, integrate demo designs from authors and editors;
- accessible, no one should be barred from using demos.

The user interface is made of classic HTML documents. We use JavaScript to enrich some HTML forms, but this is essentially sugar coating and JavaScript is not needed to use IPOL, because we want to stay accessible and because we do not want to invest in more development to evaluate the impact of JavaScript on accessibility and on the long-term

[5]SVG files in an HTML <object> tag can be viewed on Windows XP systems with Firefox 3.6 and 6.0, Opera 11, Chrome 13, Chromium 8, Safari 5, Konqueror 4 and Amaya 11. On Linux systems, they were successfully tested with Chromium 11, Firefox 3.5 and Netsurf 2.7. Safari 5 and Chrome 13 on Mac OSX, and Safari on iOS, could also display the SVG image. Only these browsers needed a PNG bitmap alternative: Internet Explorer 8 on Windows XP, Internet Explorer 8 and 9 on Windows Vista, Links 2.3 and Dillo 3.0 on Linux.

maintenance of the service, as long as it is not absolutely needed. Flash and Java are not used, and we hope HTML5 adoption will be sufficient when we need to handle audio or video data.

The base workflow of an IPOL demo is always the same:

1. Users select or upload one or more images.
2. The image is preprocessed, its size is checked and it is converted to the image file format expected by the research software shown in the demo.
3. Users can modify the input images, choose some algorithm parameters, or both.
4. The input images are processed by the research software.
5. The result is shown.
6. If the input was submitted by the user, this experiment is archived. The user can go back to step 1 or 2.

Variations between demos are changes in the steps 4 (how the input data can be prepared, which options are available), 5 (how the data is processed) and 6 (how the result is displayed).

The first backend was a simple Python CGI program, we used it for our early experiments on online demos. It was rewritten as a modular web application using the Python CherryPy web framework[6], chosen for its simplicity and minimalism: we were discovering web service development, starting small was important. This version is still a work in progress, so there is not much interest in describing the implementation in details. Until a stable state is reached and documented in future research literature, the current code is available online[7].

The IPOL demo system is currently modular in the sense that it can easily accept new demos based on existing ones or with a largely different workflow. However, it is still monolithic with a single backend to build the research programs, run them on user-submitted data, provide the user interface, feed the archives and display the archive content, everything on a single server.

The next step will probably be splitting these functions into different services which could be managed by different machines and with the possibility to grow and accept more traffic by replicating the servers. This will be the opportunity to add functions missing in the current system: source code validation and strict build procedure, monitoring of the program execution and error reports, program isolation via the virtualization of the system environment, and batch processing for large input or heavy algorithms.

3.2.5 Online Demos and Software Journals

Image processing is well adapted to a web interface because most image-related algorithms are faster than those of other computational science fields, such as fluid mechanics simulation or financial analysis. This is important because a web demo interface requires an interactive user experience. One will not start a computation on Friday and retrieve the result after the week-end, this would not be an online demo but a completely different object, a batch processing service accessible over the Web.

[6] `http://www.cherrypy.org/`
[7] `http://dev.ipol.im/git/?p=nil/ipol_demo.git`

Moreover, unlike video, sound or 3D volumes, images have always been integrated in the HTML language as a native component of the web interface. Even today, the only non-text medium supported in HTML is the images. Everything else requires add-on programs and browser plugins. 2D images are the perfect match for web interfaces because the Web was conceived to be used via a computer display, an still is. This display is a flat visual human-machine interface, and there is no time dimension in a web page. Only with the future HTML5 norms will we be able to correctly manipulate audio, video and 3D data in a web page with the `audio`, `video` and `canvas` objects and WebGL API [363, 373]. Then audio, video and volume processing will become truly possible in online demos.

Compared with the distribution as a program, the outstanding benefit of an online demo is the resolution of most compatibility issues. Online demos only rely on the web standards, which are formally defined and published and whose correct implementation depends on a single (albeit complex) piece of software, the browser. There are some real issues with the respect of these standards and some cross-browser inconsistencies, but this cannot be compared with the portability problems on cross-platform GUI programming. Moreover, the algorithms can be tried and used immediately by the web users who do not need to compile or install the program, a task that some of them are not able or willing to do. Finally, when an image is processed on the server, each remote user gets the same results after the same delay, even if their local computing environments have different performances.

Another advantage of online demos is their integration in the global Web documentation system: they are better indexed, referred and exposed than files and programs distributed to everyone's individual computer, like a web page is more visible than a manuscript distributed by postal mail. This online integration can also be used to enrich the demo with two other materials: the algorithm can be described and documented as a web page, which becomes an academic article when integrated in an editorial process, and the usage of the demo can be archived and all the past experiences, input, output and parameters, made available online to explore the effect of the demo program on more data than what one person can submit.

But there are drawbacks too. The web interface is poor if compared with the graphical interfaces available on any desktop. The network architecture is another issue. A service provided from one server to multiple users is fragile since this server is a single point of failure; any problem on the server affects the availability of the service. This architecture contradicts the resilience design of the Internet [163], the peer status of machines in an end-to-end network [308] and the non-centralization of the Web [42]. This can be mitigated, but not solved, by redundancy and distribution over multiple servers, with a financial and workload cost.

Finally, online demos are obviously not usable without a fast reliable network link. This is taken for granted in the research institutions of developed countries, but the network resource is still unavailable or rare in developing countries, rural and isolated areas and for mobile users. In contrast, a downloadable program is usable in all these situations. One should not consider that an online solution "*in the cloud*" replaces an executable program.

3.3 Reproducibility by Virtual Machines

The SHARE system [350] complete system images with system virtual machines to preserve a computing environment and avoid compatibility problems. This system-level vir-

tualization uses an emulation or abstraction container for an operating system, as opposed to process-level virtualization emulation or abstraction container for a process.

Strictly speaking, a virtual machine is not needed to benefit from the saved system image. Such an image contains the whole filesystem used by the computing system, and could be copied and reused as the main filesystem on other real (non-virtual) computers. But with the recent advances in system virtualization technologies, we can easily store, transfer, duplicate and update the system images for use with various virtual computing environments via flexible distant access interfaces. These system images can also be used in combination with hardware emulation when an hardware architecture is not available.

3.3.1 Heavy Monolithic Backup

Saving the whole computing environment implies to keep a copy of everything that can be saved. Hardware cannot yet be cloned, so only the software part is saved. We usually store a copy of all the data available from the permanent storage devices and launch virtual machines from boot on this complete clone of the filesystem. It would also be possible with current virtualization technology to store a copy of the volatile memory content at a given time and launch virtual machines from this state, but this is not relevant in the reproducible research context: we want at least to be able to use the saved software after a reboot cycle, hence from fresh memory.

The saved filesystem contains everything needed to boot the virtual machine to a user session and usually a graphical desktop interface. None of the proposals for reproducible research used a text-only console access to the virtual machine. This is the base computing environment, and includes the operating system kernel, system configuration, programs and support files for all the default system services launched until the graphical user interface and for all the programs available from this interface. The disk space requirements for a fresh operating system installation is between 5 GiB and 20 GiB[8] and even if some parts of the default operating system can be removed, the size of the filesystem to save will be in the Gigabytes.

Then we need to save the program whose environment has to be preserved, and everything needed to use it. If the program is saved as an executable, we probably need some support libraries (file format and linear algebra are common examples, visualization is likely to be used too). If the program is kept as source code, we also need the compiler and runtime library. For bytecode-compiled language, we need the run-time system (like the Java Virtual Machine). Script languages (like R or Python) need an interpreter and interactive scripting languages will also require the interactive environment (MATLAB, IDL, Scilab). This programming environment is not as heavy as the base system, but still in tens to hundreds of Megabytes, or more[9].

As these numbers show, the computing environment to preserve can be many orders of magnitude larger than the program we are interested in. The issue is not the cost of storage which, as proponents of the virtual machine solution already wrote, is inexpensive and decreasing. But many GiB of data are still heavy to transfer, and this limits practical

[8] 5 Gb are required for Red Hat Enterprise Linux 6 [295] and Debian GNU/Linux 6.0 [337], 7 Gb for Mac OS X 10.7 [23] and between 16 Gb and 20 GiB for Windows 7 [248].

[9] Compressed installers for Windows 7 weight 16 MiB for Java 6.29 (32bits version), 19 MiB for Python 2.7.2 and NumPy 1.6.5, 38 MiB for R 2.13.2. The Debian 6.0 packages for the gcc–4.4 compiler and its dependencies on amd64 platforms weight 80 MiB. IDL 7.0 on Windows used more than 400MiB and a typical installation of Matlab 7.10 on Linux 64bits uses 1.6 GiB of disk space.

usage of the virtual machine disk images to a distant server accessed via a remote desktop client.

And it lacks focus. Once the virtual machine environment has been saved as a large disk image, we can refer to it as the archived content of a computer, but the interesting files are only a fragment of this computer content, not directly accessible by this reference. Referring to the preserved software in a useful way requires:

- the reference of the virtual machine image;
- an explanation about how this machine image is to be used or accessed, with the tools involved (remote desktop or virtualization monitor);
- a documentation of the graphical desktop interface provided by this virtual machine because current desktop metaphors will eventually be obsolete;
- the detailed procedure to access the software preserved in this virtual machine, to use it, and to reproduce the intended experimental results.

Finally, because the complete filesystem is saved, any update of the software to be preserved will require a whole new copy of the environment, leading to lots of duplication and redundancy. Attempts to avoid this redundancy would imply the use of de-duplicating filesystems like ZFS [51, 298], only supported by a few operating systems, or disk image incremental snapshots, tied to a specific storage abstraction layer like a logical volume manager or Copy-on-Write disk image, which may conflict with some virtualization manager requirements.

3.3.2 Quantity *vs* Quality

Chemists don't need to keep a duplicate of the full laboratory with the equipment, supplies and clones of the staff for every experiment they want to reproduce. Instead, they keep a log of their work in the lab notebook, detail the experimental process in academic publications and refer to an established nomenclature [86]. Reproducibility is composed here by the traceability of the research process, a complete documentation of the experimental setup and reliable references to common knowledge.

This could also be attained in computational sciences:

- traceability of the source code, the compiled program and the data produced by this program can be provided by source version control and automated build tools;
- documentation can be composed of the usual academic publication (what the program should do), systematic comments and explanations of every logic part of the source code (what the program does) and a user manual (how to use the program);
- the common knowledge here is the set of published and recognized standards defining the programming languages, data formats and communication protocols.

Archiving a disk image is a desperate brute force strategy : "*save all the static data available in the computing environment, keep a copy of everything, and hope it will include all the invariants needed to reproduce a result*". We think that the identification of these invariants and the conservation of all the information that defines the computing process without ambiguity, and only this information, is more rational and more interesting.

3.3.3 Portability

The proposals for reproducible science by means of a virtual machine cited the prominent current virtualization tools Xen [30, 181], VMWare [183, 331], VirtualBox [88, 362] and OpenVZ [281]. All these technologies are based on the same model: they read a disk image and use this content to define and execute a virtualized operating system.

The three major desktop operating system families in 2011 are Windows, Linux and Mac OS X[10] with usage share varying across communities. These three systems are used as platforms for computational research, so they all need to be supported by a virtualization system if it is to be used to publish and exchange research content.

But the virtualization support is limited: OpenVZ only accepts Linux as a guest operating system, Xen only supports Windows and Linux, VirtualBox and VmWare will only virtualize recent versions of Mac OS X on Apple hardware and when using Mac OS X as a host operating system, because of restrictions imposed in the end-user license agreement [24, 25]. Practically, this means that researchers using Mac OS X as their primary work environment would be excluded from publication tools based on virtualization.

And each virtual machine technology details its guest operating system compatibility per OS, version and/or distribution. This means that they provide limited support for old operating systems, which could be a concern when current systems will be considered old, 15 years later. And this compatibility list means virtualization tools do not provide a neutral hardware abstractions on which any operating system could be run. Moreover, these virtual machine tools can all read raw copies of the storage medium, but they all define their preferred and incompatible format [372] that no other virtual machine monitor can read.

In spite of some unification efforts at the API level [223], virtualization is still fragmented across technology vendors and operating systems, and this endangers any attempt to base a reproducible research publication platform on this model. All the portability issues can be solved by ad-hoc measures, conversions and migrations, but the absence of a generic solution is a serious scalability issue if virtualization had to be provided for thousands of computational science articles every hear, each of them storing gigabytes of data as seen previously.

3.3.4 Other Concerns: Licenses, System Updates and Reusability

The Microsoft or Apple End-User License Agreements for the Windows or Mac OS X only allow a limited number of instances of the operating system to be simultaneously installed and/or used. With this restriction, if these operating systems must be accepted, then storing virtual machine environments for reproducible research will require some provisions for bulk licensing or will need some sort of on-demand license accounting every time someone want to use a virtual machine archive. The same problems will arise for computing environments bound by similar license terms, like MATLAB.

Over time, defects and security issues are discovered in operating systems and software. Some of these may put the user data, user privacy, or the computing resources y exposing

[10] The exact figures are difficult to collect [371]. Wikimedia report 79% of its traffic from Windows web clients, 8% from Mac OS X, 5% from iOs and 3% from Linux [370]. The research community probably has higher Linux figures: IPOL observes 73% of its visits from Windows, 16% from Linux, 11% from Mac OS X [186].

them to unauthorized users. But these defects cannot be fixed by software updates, because this would modify the computing environment and defeat the conservation goal.

Finally, with these preserved system images, one gets a closed box. The box may be functioning, always produce the same output in a predictable way, and provide a reproducible computational result, but there are no provisions to reuse it. Scientific collaborations and developments are achieved by the exchange and recycling of successful ideas, and a software is an idea expressed in a programming language. So the correct place for this software seems to be among other published, discussed and exchanged research works, not embedded and locked in a freezed computing environment. If the program needs the preserved environment to function correctly, then it will loose its pertinence once the preserved environment is obsolete, and its scientific interest is rather poor. On the other hand, if the program can function correctly out of this preserved environment, then the virtual machine infrastructure is not needed to achieve reproducibility and harms reusability.

3.3.5 An Experimental Tool

In conclusion, virtual machines technologies are great for experiment and collaboration, but they are not a good solution for publishing and reusing. Instead, we propose with IPOL the conservation of the minimum set of readable information defining the computation: the source code, expressed in a unambiguous standardized language syntax; external computing tools can be used by means of a public stable API, or included with the program as source code; some attached data files, subject to the same standard unambiguous formatting requirements. Then virtual machines can still be used as a backup, a solution to archive a computing environment and keep a software somehow accessible when everything it depends on is obsolete.

3.4 Implementations and the Scientific Method

Online demos only differ from standard implementations by their distributed, network-based properties, and their compared advantages and drawbacks have been presented. But most of the research articles in signal processing are released without code [353], so the interest of research programs needs to be explained, to justify the development of online demos.

Theoretical sciences are based on logic and deduction and do not require implementations. For example, a typical math article will focus on demonstrating a new theorem. The scientific content of the paper is the demonstration of the theorem, the set of logically connected assertions such that if the conditions are met and the axioms accepted, then the conclusion is right. The subject matter of this science is the abstract concepts and properties represented by the mathematical notations. No program is needed there, and established standards of the mathematics community ensure that a proof is replicable, well defined and verifiable by every researcher of the field.

Experimental sciences are based on hypothesis and deduction, and do not need implementations either. A biology article will propose an hypothesis and a method to test it, in the form of an experiment. This experiment will be described with enough details for other researchers of the field to independently reproduce it and verify the published conclusions. Here, the subject matter is the hypothesis and the procedure to test it, and they are

accepted until a new experiment brings a contradiction or a new hypothesis refines the previous one.

Theoretical or experimental sciences may use software, but this software is a research tool. If it is the subject of the research, then this research is not theoretical nor experimental anymore.

Depending on the local academic culture, image processing is categorized as applied mathematics, computer science or electrical engineering[11]. Some image processing research qualifies as theoretical science when only abstract objects and properties are manipulated, but these are not the works we are interested in. Image processing can also be involved in other sciences like biology, astronomy or geography, but this does not imply that image processing belongs to these domains, not more than database management, often needed for computational genetic research, is a branch of genetics. For a large proportion of digital image processing research, the subject matter is algorithms, *ie* numerical procedures applied on images to achieve a goal expressed in term of image properties.

Some algorithms can be demonstrated: one can prove that a sequence of manipulations on a numerical array will sort the array, or perform a discrete Fourier transform, or invert a matrix. This is theoretical, deductive research, and while these algorithms may be useful for image processing, the elaboration and study of these algorithms is not image processing as long as the algorithms operate on abstract numerical data.

Image processing algorithms are not software, but they can be expressed as software and they need to, to be applied to digital images. This unavoidable presence of software in applied digital image processing makes it a branch of the computational sciences branch, neither theoretical (computers are not abstract concepts) nor experimental (software do not test hypotheses on the laws of nature).

Application cannot be avoided in image processing when we claim to solve a problem with an algorithm. In that case, the subject matter of the research is not the properties of the algorithm, but its usefulness, performance and robustness. These are not well-defined properties when an algorithm is claimed to "*match features*", "*remove the noise*", "*correct the color balance*" or "*interpolate an image*" because we are interested in algorithms solving these problems on natural images, not on any abstract numerical array. We do not have theoretical tools to demonstrate that, for example, for any digital image obtained from a natural scene, our algorithm corrects the color better than any other color balance algorithm, and there are no reliable automatic quality assessment measures for images. Thus, readers and users of the research litterature should judge by themselves the results of algorithms.

In the absence of deductive methods, one solution is to build trustworthy implementations

[11] According to the 2009 IEEE Taxonomy (`http://www.ieee.org/documents/2009Taxonomy_v101.pdf`) and 2009 IEEE Thesaurus (`http://www.ieee.org/documents/pdfieeethes05nov09.pdf`), *image processing* is a branch of *computers and information processing*, but biomedical image processing is a sub-branch of *engineering in medicine and biology* and stereo image processing is a sub-branch of *imaging*. The 2010 Mathematics Subject Classification of the AMS (http://www.ams.org/mathscinet/msc/msc2010.html) places *image processing* in the *computer science / computing methodologies and applications* and *information and communication, circuits / communication, information* categories and the *computer graphics, image analysis, and computational geometry* in *numerical analysis / numerical approximation and computational geometry*. The 1998 ACM Computing Classification System (`http://www.acm.org/about/class/1998`) considers *image processing and computer vision* as a branch of *computing methodologies*. In the Dewey Decimal Classification (`http://www.oclc.org/dewey/resources/summaries/`), *image processing* can be found in the *000 – Computer science, information and general works*, *500 – Science (including mathematics)* and *600 – Technology and applied science* classes.

of the algorithm, use them to process images and be convinced, after thorough exploration of the range of typical images and possible special cases, that the algorithm performs as expected, until a better one is proposed or failure cases are discovered[12].

In experimental sciences, the experiments are built to test an hypothesis on the laws of nature. In computational sciences, experiments are conducted to test an hypothesis on the properties of an algorithm. Image processing software are engineering tools to process images according to an algorithm, and image processing science is involved in the design of the algorithms and their software realization and in the empirical verification of the qualities of the algorithms via those of their implementations.

We can see now why research articles do not contribute to the field if the authors do not provide a publicly available implementation or a complete, very detailed description of the algorithm sufficient to write a software implementation. Online demos are tools for the scientific method: they simplify the use of the implementations and they collect in their archives all the intelligent testing effort of the community to validate the algorithm properties.

[12]This is obvious when we look at the series of contributions to a research domain, like "*intelligent image resizing*" for example, where every yearly conference has a paper whose authors claim to solve the question and provide a video with a set of chosen examples, until the next authors exhibit other images where all the previous methods failed badly.

Chapter 4

Software for Reproducible Research

Contents

Abstract

Publishing some software in a scholarly journal implies that this material is reviewed, and that the journal readers can expect some level of quality standard for the software as well as for other kinds of research papers. Such a software should be usable and have predictable effects across a reasonable diversity of present and future computing environments. The source code should be available — otherwise no one can say what the software is really doing — and it must be readable and documented, just like the math demonstrations in a paper are readable and explained. We propose to establish these quality expectations as a Software Guidelines document composed of requirements and recommendations, similar to the Article Guidelines frequently used in text-only journals. We also propose an automated testing procedure to help the reviewers verify the quality of the code. In addition, these guidelines are a good place to mention the copyright, license and patent policies in place for the software publications.

4.1 The Need for Software Quality

To publish a research article in mathematics, the author's working notes are rewritten into a form suitable for a research journal. During this rewriting the author's mnemonic shortcuts are replaced by standard and widely used notations. The text is edited for grammatical correctness and clear style [330]. The work is structured according to recognized composition rules [274, 345], and assembled into a printable matter with the help of a typesetting environment [202, 210].

All this work in rewriting makes the research readable, understandable, useful and unambiguous. The scientific content of a paper ready for publication is arguably not more than the original ideas scribbled on the author's blackboard, but the presentation efforts made by the author in writing a research paper will help convey the ideas expressed in the text to its multiple readers.

Scholarly journals provide their "Author Guidelines" about how articles should be written. We think that some software guidelines are also needed when a journal expects to receive, review and publish a computer program. Code is read much more often than it is written, and the source code is read directly, without being processed by typesetting solution, so its readability is worth some efforts from the authors. Moreover, a source code is not only expected to be read but also reused, so care has to be taken for the usability of the code in addition to its clarity. Conversely, compiling and running a program is not sufficient if we cannot understand what is done with the program.

Various coding style guides have been written and used to enforce a unified visual style and improve the quality of the code since the original *Indian Hill Recommended C Style and Coding Standards*[1] and the *Elements of Programming Style* [197].

The guidelines hereafter were developed as an attempt to guide the authors of implementations submitted to IPOL. Their goal is to increase the readability and usability of the programs published in the journal, with a set of requirements and recommendations for an article to be accepted.

[1] Many C and C++ style guides have been collected and archived by Chris Lott (`http://www.maultech.com/chrislott/resources/cstyle/`). An example of recent and exhaustive coding style reference is used for C99 code by the EPITA computer engineering school (`http://tsunanet.net/~tsuna/codingstyle/codingstyle.html`).

They do not cover topics like the logical correctness of the implementations or the architecture of the software, which are expected to be evaluated directly by the reviewers. Another document could provide some advice on these topics, but the matter is wide and requires hundreds of pages to be completely covered [196, 243], not counting the additional matter of security risks in software and how to avoid unintended abuse of the program [310, 367]. The current guidelines only focus on the priority: readable and usable programs.

We believe that these guidelines are relevant for other computational science research communities, after some adaptation of the domain-specific items like data file formats, common software libraries, or essential languages.

4.2 Software Guidelines 1.00

The first version of the IPOL Software Guidelines was adopted in December 2011. Abridged guidelines are reproduced and commented hereafter; the full text is available in annex 9 and on the IPOL web site [187]. They will probably be revised in future versions with the experience collected from their usage.

As a normative document, the guidelines need to be expressed without ambiguity. The vocabulary described in IETF RFC2119 [58] was chosen for its concise yet clear expression of the requirements, recommendations and options. For reference, the guidelines are publicly available online [187] and every guideline item is numbered.

4.2.1 Packaging and Content

The first set of guidelines defines how a program is distributed and what it contains. Their goal is to ensure that the program can be easily identified, transmitted and evaluated, and that it can be manipulated by everyone.

1.1. Compressed Archive

> "*An IPOL program must be packaged as a compressed archive file, either a single volume .ZIP compressed archive or a GZIP compressed tar archive. The size of the compressed archive file should be less than 2 MB.*"

Programs are a set of files, but manipulating them as a single atomic item is more convenient, so we require them be merged in a single archive file. And because it must be handled via an archive tool, we can also compress it for easier exchange. Zip is the default archiving format on Windows systems and tar/gzip the default in Linux, so both formats are allowed.

These archive formats are formally defined[2] but examples of programs usually available or easy to obtain for most systems (`tar`, `gzip`, `zip`, `7zip`) are provided to guide the authors.

Large and multiple data files, such as large uncompressed images or video, could lead to arbitrarily large archives, difficult to store and exchange with little or no benefit. This is

[2]The .ZIP compressed archive format is defined by the PKZIP APPNOTE documentation [193], the tar archive format is defined by the POSIX.1 ustar standard [340], and the GZIP compression is defined by the IETF RFC1952 [99].

avoided by setting a size limit on the program packaged as a compressed archive to 2 MB. This arbitrary size was chosen because it fits all the software already published in the platform before the adoption of the guidelines.

1.2. Archive Name, Program Name and Version

> "*The compressed archive file of an IPOL program must be named according to the* `name_version.extension` *pattern, where:* `name` *and* `version` *must consist only of lower case letters, digits, minus and period signs,* `name` *must be at least two characters long and start with a letter,* `version` *must start with a digit, and* `extension` *is* `zip` *for zip archives and* `tgz` *for tar/gz archives .*"

The program needs to be identified and this identification must allow the distinction between successive revisions of the program, *ie* a program must have a name and a version number. To access this information without decompressing the archive package of the program, the name and version is encoded in the file name of the archive and defines it. It requires a separator, "_", and a restricted set of characters compatible with all common filesystems and text processing utilities. The name should be pronounceable, and start with a letter; the version number should be sortable, and starts with a number.

Unique program names and strictly increasing version numbers are desirable, but this is left for later versions of the guidelines because it requires some tools to be developed and maintained for this purpose.

1.3. File and Folder Names

> "*All the files and folders extracted from the compressed archive must be located inside a base folder named* `name_version`*, where* `name` *and* `version` *are identical to those used for the compressed archive file name.*
> *The name of all files and folders composing the IPOL program must consist only of lower or upper case letters, digits, minus, underscore and period signs.*"

The extraction of the content of an archive may overwrite existing files, mix them with other files in the working directory or create them in unusual locations. To avoid this undesirable effect, all the content is required to be located, after extraction, in a folder named after the compressed archive file. The archive is a single compressed folder containing all the files included in the program.

The files extracted from the program archive must be usable and comfortable on all systems, filesystems and localizations, so the file names are required to be written in the ASCII character set without any path separator (slash, backslash, colon), command separator (space, comma, semicolon), wildcard sign (asterisk, question mark), comment marker (pound sign, percent sign),...

1.4. Hidden and Useless Files

> "*An IPOL program should not include hidden files or folders or by-products of the tools used by the authors, such as (but not limited to) files inserted by file managers, folders inserted by version control managers, backup versions.*

The program should not be distributed with files not useful to build, use or study the implementation of the algorithm published in IPOL."

If a file is not useful, it has no reason to be included and distributed with it. This rule reflects the desire to distribute the final versions of programs, not working drafts.

4.2.2 Implementation

The second group of guidelines takes care of the portability of the software. No guideline can guarantee than a program will be usable on any present and future computing system, but known sources of incompatibilities can be avoided.

2.1. Source Code

"*An IPOL program must include all the material necessary to build one or more executable program files implementing the algorithm published in IPOL. This material must be provided in human-readable source code form.*"

The first source of incompatibilities is the binary executable format. When a program has been compiled for a computing system and hardware platform, it is not executable in other environments[3]. The distribution of the program in source code format preserves the possibility to recompile it into executable form on other present and future environments.

Binary versions of the program may be useful for some users and can be distributed by the authors in a software journal, but we know that these versions will eventually be obsolete. For this reason, they must be clearly distinguished from the reference implementations of the published algorithms, and not distributed via the compressed archive of the program.

The source code format also allows the review, validation and documentation of the implementation; these aspects are covered by later guidelines.

2.2. Programming Language

"*The source code of an IPOL program must follow the published standard syntax of one or more compiled programming languages. IPOL can currently only process C89 (ANSI C), C99, and C++98 (ISO C++).*
The source code may use the OpenMP 3.0 API for shared multiprocessing programming but it must also compile and provide the same results without OpenMP."

To retain the possibility to reuse the software, the language used to express it must be understandable on various present and future systems. Nothing can guarantee that a language will be usable forever, but some languages and dialects are known to be only usable on a limited set of computing environments or to have a limited lifetime, and should be avoided. Languages implemented by a single vendor are always menaced by possible issues with the vendor company and discontinuation of the support for this language.

[3] Hardware emulators and operating system compatibility layers can be a solution to execute a program built for one system and platform in another, but these partial solutions imply a performance penalty and may not be maintained in the future.

The MATLAB, IDL and similar programming languages lack a public and formal specification and are tied to a single vendor, and we cannot hope implementations expressed in these languages to be usable for a long time[4].

The Java, Python or Ruby languages are well defined with public specifications, but we chose to refuse implementations in interpreted languages because they usually (but not always) have worse performances than compiled ones. Another issue with languages like Java, Python and Ruby is that the execution environment of these languages must be available to use these programs. Java needs a Java Virtual Machine [229] to execute the Java bytecode. Python and Ruby need the language interpreter to be available at run-time to execute the scripts. Programs written in a compiled language only need to be processed by a compiler once, at build time; then the compiled programs contain instructions directly understandable by the processor and at run time and they are directly processed by the operating system without the need for any external virtual machine or interpreter. Moreover, more combinations and code reuse is possible between compiled programs than between interpreted ones[5].

Our solution is to only accept compiled languages defined by a publicly available specification, such as C89, C99 and C++98 [84,130,280]. Other languages, such as Fortran 90, are well defined and could be accepted, but a survey among our early authors suggests that this language is not very represented in the image processing community; C and C++ are the main or only programming languages of almost all our current authors. Of course, the situation would probably be very different for numerical analysis and simulation research. An alternative could be to publish MATLAB, Java, Python or Ruby codes and expect the authors to maintain their implementations if the language changes. These codes written in a high-level language would be more compact than C/C++ codes, more readable and closer to a pseudo-code.

We are not aware of tools to strictly validate the conformance of a source code with the standard definition of a language. And even if such tools exist, they are not sufficient because we would also need to verify that the code is correctly interpreted by the compiler. Instead of pursuing formal purity goals, we prefer a "best effort" pragmatic approach: the code should be tested by the authors and reviewers with strict compilation options[6], but this can not be a strict requirement as long as different compilers or different compiler versions produce different warnings. Automated testing tools are planned with compilations with different compilers in a controlled minimal environment and tests with static and dynamic check tools (such as Splint, Clang, or Valgrind[7]).

[4]The MATLAB language is only defined *de facto* as the language implemented by the Matlab computing environment. New versions of this environment are released every six months, and with every release the syntax or effect of some functions are modified and some functions are added or removed [182].

[5]C, C++, Fortran and Ada source code and object code (the compiled programs) can be combined on the basis of the C application binary interface. They are all compiled using the same instruction set and data primitives, into binary program files following the same format. In contrast, Java, Python and Ruby sources are translated into three different and incompatible representations of a program (Java, Python and Ruby bytecode), then these hardware-independent representations are processed by three different run-time environment. In-depth explanations can be found on the Wikipedia pages about interpreters (`http://en.wikipedia.org/wiki/Interpreter_(computing)`) and process virtual machines (`http://en.wikipedia.org/wiki/Virtual_machine`).

[6]As a preliminary test, authors are instructed to test their implementation with the GCC compiler using the `gcc -std=xxx -Wall -Wextra -Werror` options where `xxx` is `c89`, `c99` or `c++98`.

[7]Splint is a static code checking tool (`http://splint.org/`), Clang is a C/C++ frontend to the LLVM compiler with code analysis features (`http://clang.llvm.org/`) Valgrind is a framework of dynamic analysis tools with a memory error detector (`http://valgrind.org/`).

Parallel computing is needed to achieve the best possible performances on modern architectures, but this form of computation is not covered by the C and C++ language standards. We chose to allow the authors to use the OpenMP model[8] because it is standardized, portable and available with every major compiler. OpenMP only implements shared-memory parallel computing, but this model is sufficient and we see no need for distributed computing or cluster architectures in our image processing environment in the coming years. But we also recognize that OpenMP is not available neither relevant on all computing environments. For this reason, the programs must also be usable without parallel computing and provide the same results, albeit possibly slower.

OpenCL[9] is another framework for parallel processing, with support for GPU hardware and heterogenous systems. This open and cross-platform standard could be allowed too, once it is supported by enough compilers.

2.3. Portability

> "*The source code of an IPOL program must not require any extension of the language or its standard library, or any resource specific to a hardware environment, operating system or compiler. These extensions and resources may be used to achieve better performances if they are available but their availability must be detected during the compilation or execution and an alternative portable implementation must be used in their absence.*"

This section enhances the requirement for a standard-compliant source code. Implementations targeting a specific compiler, operating system or machine do not benefit users of other computing environments, and become completely useless once the target environment is obsolete (and every piece of hardware and software eventually becomes obsolete[10]).

The goal of reproducible research is not to obtain implementations for one or a few systems[11], but implementations that should be usable on any real or virtual computing environment implementing the aforementioned system-agnostic language standards. This means that language dialects specific to a compiler, standard library functions specific to an implementation, assembler code or "intrinsic functions" mapped to a processor instruction, code tied to a specific hardware component (GPU), operating system calls, file system locations and code specific to a memory model must be avoided.

This guideline does not imply that a program following these guidelines will be usable in the standard default environment of Windows, Mac OS X and Linux systems. For example C99 code cannot be compiled with the default Microsoft Compiler [47]. Installing and using an appropriate compiler, for example, can be needed. But using another operating system must not be required, and any compiler correctly implementing the published language standard should be sufficient.

[8] `http://openmp.org/`

[9] `http://www.khronos.org/opencl/`

[10] The Windows 32bits and Intel x86 backward compatibility history is an exception in the computer world. Apple systems went through two architecture and one major operating system change. Architectures once prominent in high-performance computing, such as DEC Alpha or PA-RISC, are now extinct. ARM is now the architecture deployed on the largest number of computing units and used on almost every mobile platform.

[11] An implementation with variants for Win32 and POSIX systems is not sufficient, because operating systems are not limited to this alternative.

We recognize that the performance of the implementation can benefit from the use of some features specific to a hardware or system environment, such as data parallelism with vector instructions. These features can be used in the implementation for the benefit of the users, but the code must not depend on it and alternative implementations must be available, with the same computational results.

2.4. Dependencies

> "*An IPOL program must not use external software components except for the libraries and APIs listed hereafter:* `libtiff`*,* `libpng`*,* `libjpeg`*,* `zlib`*,* `fftw`*,* `cblas` *and* `clapack`*.*"

According to the previous guidelines, a good program should include the complete implementation of the published algorithms. This is theoretically feasible but neither realistic nor appropriate because it would imply lots of work for every algorithm, with some duplicated and sub-optimal implementations. Moreover, file management, numerical analysis, optimization and linear algebra solvers (among others) are not in the area of expertise of image processing researchers and they should not have to spend time on these tasks.

We need to set attainable goals and assist the authors in their efforts, so we allow so far the use of a few external libraries: `libtiff` and `libpng` to read and write image files, with their dependencies `libjpeg` and `zlib`[12] and `libfftw`, `blas` and `lapack`[13] for Fourier transforms and linear algebra operations. These libraries were selected on four criteria: they are useful, widely used, portable (at least Linux, Mac OS X and Win32 systems) and have a stable programming interface[14]. The GNU Scientific Library[15] has been examined, but its usability on Windows systems needs to be confirmed before it is allowed as an external dependency. Moreover, we provide some code samples and tools[16] to help authors start with their implementation and access external libraries via simplified interfaces.

This restriction only applies to software components used by the program but not distributed in source form with the program. The program may include some code from other software projects, programs and libraries, and this is encouraged when it helps improving the quality of the implementation, as long as all the source code follows the same guidelines, regardless of its origin.

A common objection is that no serious implementation is possible without external libraries. Our answers are that many interesting image processing algorithms do not require any complex software component, that essential building-blocks are available in the C++

[12]These image libraries are available at `http://www.remotesensing.org/libtiff/`, `http://libpng.org/pub/png/libpng.html`, `http://www.ijg.org/` and `http://zlib.net/`.

[13]These numerical libraries are available at `http://www.fftw.org/`, `http://www.netlib.org/blas/`, `http://www.netlib.org/lapack/` and `http://www.gnu.org/software/gsl/`.

[14]libfftw 3.0 was released in 2003, libfftw 2.0 was released in 1998 and this branch is still maintained. libpng 1.2 was released in 2001 and is still maintained, in parallel with the recent 1.4 and 1.5 branches. LAPACK 3.0 was released in 2000 and is still in development. The BLAS interface has not changed since its release thirty years ago.

[15]These numerical libraries are available at `http://www.fftw.org/`, `http://www.netlib.org/blas/`, `http://www.netlib.org/lapack/` and `http://www.gnu.org/software/gsl/`.

[16]The IPOL wiki has simplified interfaces to `libpng`, `libtiff` and a portable high-quality random number generator (`http://tools.ipol.im/wiki/author/code/tools/`) and a collection of contributions from authors willing to share their work (`http://tools.ipol.im/wiki/author/code/hatchery/`).

standard template library and reusable software collections[17], and that is always possible to include a library in source form with the implementation of an algorithm. If such a library is not available in source form, or if its compilation process is too complex, or if it is not made of standard and portable source code, then one cannot expect it to be usable in the long term. And finally, one can greatly reduce the library needs by focusing on the essential, the algorithm; we do not need to support many image file formats, to have a graphical user interface or numerous options and variants to implement and demonstrate an algorithm. Simpler implementations are smaller, have less bugs, are easier to use, test, analyze and reuse.

2.5. Compilation

> "*An IPOL program must be compiled by an automated non-interactive build procedure with* `make` *or* `cmake`*. This build tool must not be configured to use any special compiler. The default build procedure must use standard compiler options only.*"

The automated compilation tools provide a unified interface and syntax to build any program, and the users do not need to know any detail of the implementation to be able to compile it. Moreover, the Make and CMake tools are so common that every programmer knows how to use them when they notice a `Makefile` or `CMakeLists.txt` file. It also prepares future developments for an automated build and test tool.

Here again, we want to maximize the chances of a portable implementation, including its build procedure. We do not want the name of the compiler to be hardcoded in the build configuration (Make should use the generic `$(CC)` and `$(CXX)` variables for the C and C++ compilers) and we only want to use standard compiler options.

The closest thing we could find to "standard compiler options" are the options defined by the the POSIX `c99` specification [341] which happen to be valid options for all the C compilers we know on POSIX systems. The Microsoft Visual C compiler understands most of these options [247], with the Windows `/x` syntax instead of the UNIX `-x`. The only POSIX-specific options are `-g`, `-l`, `-o`, `-s`. the first two ones can be ignored because debugging symbols and stripping are not crucial for a default compilation. However, `-o` and `-l` cannot be avoided, and the latter is closely tied to the UNIX shared library model, very different from the Win32 DLL system. A `Makefile` with only POSIX compiler options should work with any compiler on a POSIX system, and the true portability is attained with CMake, which determines the compiler options once the compiler is known.

This guideline shouldn't be interpreted as the interdiction to use any compiler-specific compilation option, such as optimized compilation flags. These options can be useful for some users and generate programs with better performances. However, they must not be active in the default build procedure, and should explicitly be invoked[18].

[17] Even if it is not yet allowed as an external library in these guidelines, the GNU Scientific Library is designed in a modular way allowing for extraction and reuse of only the needed parts. Daniel Atkinson's `ccmath` (`http://freecode.com/projects/ccmath`) is another example of a math library in which a single component can be isolated and reused.

[18] `make optimized` or `make -f Makefile.gcc` can be two means to explicitly invoke non-standard build procedures, by choosing a specific build target or configuration file.

2.6. Usage and Input/Output

> "*An IPOL program should be minimal and only perform the algorithm published in IPOL. It must be usable from the command line environment without any user interaction, taking all its parameters from the command line.*
> *An IPOL program must be able to read the input data and write the final output data in at least one of these formats: PNG, TIFF or PNM for raster images, EPS or SVG for vector images, VRML or PLY for meshes, and plain text for other data.*"

Any feature of the program that is not needed to implement the algorithm imply more source code, more software bugs, more need for documentation, more effort to read and review the code, more effort to isolate and reuse the algorithm for further reuse. We particularly do not want the algorithms to be implemented as one module or function in a large application or library.

The command-line interface is the universal interactive user environment, available on any interactive computing system[19], the only one that needs to be provided by any published program. It is natively supported by the standard C and C++ libraries and completely portable. Its support is built in the operating systems[20]. The demo interface combines the command-line execution of the programs with a web front-end.

The input/output part of this guideline item ensures that anyone can prepare some input data for the program and read and understand the output data via a list of well-defined, well-known and largely supported file formats[21]. Other formats can be added later to this list when the need arises. And we would like to see the end of the multitude of half-baked, ill-defined, poorly documented and useless file formats used for every new local image processing project while other ones, carefully designed and implemented, could be chosen.

2.7. Computing Resources

> "*In the demo environment, the program must not need more than 30 seconds to process typical data. For slow algorithms, this limit may be achieved with parallel processing or a limit on input size. An IPOL program should not use more than 1 GB of memory and must not use more than 8 MB of stack memory space.*"

It is a consistency requirement that only one version of the implementation of an algorithm is reviewed, distributed and used for the demo. Without that, one cannot know if

[19] A historic and cultural account of the importance of the command-line can be found in "*In the Beginning... was the Command Line*" [325]. It predates the apparition of mobile computing devices (iPhone, iPad) where the command-line interface has been deliberately hidden and is there, but locked out, inaccessible to the users by a questionable design choice.

[20] Operating systems can launch any program with command-line options with system calls like the POSIX `execvp()` without any terminal or command shell interpreter.

[21] PNG is defined by the IETF RFC2083, TIFF is defined by the Adobe TIFF 6.0 Specification, PNM (PBM, PGM and PPM) is defined by the netpbm documentation, EPS is defined by the Adobe Encapsulated PostScript 3.0 Specification, SVG is defined by the W3C Scalable Vector Graphics 1.0 Specification and VRML is defined by the ISO/IEC Virtual Reality Modeling Language Specification [34, 55, 125, 290, 333, 334]. There is no formal published specification of PLY, but this simple format introduced by the Stanford 3D Scanning Repository is documented on Paul Bourke's site [54].

objections made to the reviewed code are relevant for the people who download the code, or if the results observable in the demo are really achieved by the implementation of the algorithm as described.

This also means that the time and resource constraints of the demo environment have to be taken into account for the implementation.
To maintain an interactive feeling, the web users should not wait for more than 30 seconds to get the results of an algorithm. Slow algorithms can be accelerated by parallel processing. If this is not enough the input size can be limited.

The memory limits are here to preserve the coexistence of several demonstrations invoked simultaneously on the same machine, and to inform the authors of technical limits on the demo server.

4.2.3 Copyright, License and Patents

The third category of guidelines ensures that the rights of the authors, contributors, inventors, readers and users are clearly mentioned and respected. These are the legal guidelines.

3.1. Copyright Attribution

> "*Every source code file in an IPOL program must mention its authors in a copyright attribution line at the top of the file. This mention may be omitted in very simple files such as header code.*
> *Every person whose contribution to this file is not trivial and implies some creative work must be credited.*"

According to international copyright conventions, software has the status of literary work and its copyright automatically belongs to the authors[22]. Mentioning this information in every file clearly conveys the copyright information, the minimum credit due to the software authors. The obligation to mention it in every file avoids omissions and code of obscure origin.

The copyright attribution must include the years of production of the work, the full name and an e-mail address for contributor. It may also include other relevant information such as the employer, affiliation or a web site.

The copyrights are attributed to every author whose creative work was involved in the source code file, so successive authors are added while the code evolves, as shown in figure 4.1.

```
Copyright (C) 1998-2003, Taro Yamada <taro.yamada@example.jp>
Copyright (C) 2005-2011, Juan Perez <juan.perez@example.es>
Copyright (C) 2011,      Marie Untel, ENS Cachan
                         <marie.untel@ens-cachan.fr>
```

Figure 4.1: Copyright attribution for multiple authors.

[22]Different countries have different local regulations, and may handle the economic rights in different manners, but the moral rights (exclusive right to claim authorship of a work) are consistently attributed to the authors.

3.2. Patent Warning

> "*When the authors are aware or suspect that a source code file implements an algorithm which might be linked to a patent (the main algorithm published on IPOL or another algorithm used for this implementation), a patent warning must be inserted after the copyright attribution, in every file potentially linked to this patent.*"

In some jurisdictions, patents on inventions involving a software component can result in the interdiction to distribute, compile or use an implementation of the patented invention. This patent restriction depends on local laws and changes as the laws change and the patents expire. The editor of a software journal cannot determine the rights of the software users in these various situations and cannot anticipate which patents will be enforced and which patents will stand in court.

Due to the nature of a software publication as a mean for experimentation and research, we consider that any algorithm can be distributed as source code. When the authors are aware of the existence of patents, they are required to inform the readers by the insertion of a patent warning. The exact determination of the rights of the readers and users is their responsibility, and any other rights conveyed to the users by subsequent software licenses are conditioned to the absence of conflicting patents. This recommended wording for this warning is shown in figure 4.2.

```
This file implements an algorithm possibly linked to the patent
<REFERENCE OF THE PATENT>.
This file is made available for the exclusive aim of serving as
scientific tool to verify the soundness and completeness of the
algorithm description. Compilation, execution and redistribution
of this file may violate patents rights in certain countries.
The situation being different for every country and changing
over time, it is your responsibility to determine which patent
rights restrictions apply to you before you compile, use,
modify, or redistribute this file. A patent lawyer is qualified
to make this determination.
If and only if they don't conflict with any patent terms, you
can benefit from the following license terms attached to this
file.
```

Figure 4.2: Recommended wording for the patent warning.

3.3. License

> "*Every source code file must mention a usage and redistribution license after the copyright attribution (and patent warning for algorithms potentially linked to a patent): GPL/LGPL/AGPL or BSD when no patent risk is known, BSD when a patent is registered and the code authors are not the patent inventors, 'for research and education only' when a patent is registered by the code authors.*"

In the absence of a software license, the authors of the source code retain all the rights attached to the program. One may suppose that, because the source program is available

online, everyone is allowed to download it, but modification and redistribution in source or compiled form are forbidden unless explicitly allowed.

The source code is published for the benefit of the readers and users, who should be allowed to download, read, use, modify, reuse and redistribute it. These rights match the Free Software principles [134, 338], so to guarantee these right for the users the authors are required to distribute their code under a free software license.

To avoid license proliferation and keep things simple, we propose the GPL/LGPL/AGPL and BSD licenses, which are expected to cover all the use cases with various levels of restrictions and freedoms. GPL licenses are known to conflict with patent rights [382], so in case of a patented algorithm, we require the implementation to be distributed under the BSD license. A special case is when the patent inventors are the copyright holders of the implementation: in that case, they cannot simultaneously assert their patent rights to be the exclusive distributors of the invention and distribute the code under a free software license granting unlimited redistribution rights. We consider this exception and allow the code, in that case, to be distributed under a "research and education only license" because this matches the patent exceptions in most jurisdictions. The recommended wording for these license terms is shown in figure 4.3; The full license terms should be provided in a separate file.

```
This program is free software: you can use, modify and/or
redistribute it under the terms of the GNU General Public
License as published by the Free Software Foundation, either
version 3 of the License, or (at your option) any later
version. You should have received a copy of this license along
this program. If not, see <http://www.gnu.org/licenses/>.

This program is free software: you can use, modify and/or
redistribute it under the terms of the simplified BSD
License. You should have received a copy of this license along
this program. If not, see
<http://www.opensource.org/licenses/bsd-license.html>.

This program is provided for research and education only: you can
use and/or modify it for these purposes, but you are not allowed
to redistribute this work or derivative works in source or
executable form. A license must be obtained from the patent right
holders for any other use.
```

Figure 4.3: Three recommended wordings for the license information.

Finally, as exact copyright and license regulations depend on the countries and professional environments, the authors are encouraged to verify that these copyright and license terms are adapted to their personal situation.

4.2.4 Documentation

The last set of guidelines covers the documentation, and has provisions to guarantee that the reviewers and readers of the published implementations will have all the information they would need about the software. This includes generic information such as references

to the article and usage instructions. But the source code itself is considered a published material and is expected to be read, so some rules are added to guarantee that the implementation is readable and understandable.

4.1. README.txt

> "*Every IPOL program must provide a file named* `README.txt` *in the base folder and written in plain text and in English. This* `README.txt` *file must include the following essential information, in any order: name and brief description of the program, reference to the IPOL article, authors and contact information, version number and release date, location of future releases and updates, copyright, patent and license information, tools and libraries needed to compile and use the program, compilation instructions, usage instructions and example, changes in the program since it was first published in IPOL*"

The `README.txt` is the basic information file traditionally distributed with programs. In simple text format, it is universally readable and does not require any special software to be read. Some authors may wish to provide extensive documentation in a visual format like PDF or HTML, but such documentation will complement the simple `README.txt` file, not replace it.

Software packages will be downloaded, stored and archived by the readers, and might be redistributed. It is important for the completeness of this package to maintain the link with the original article, and that is why the articles and authors must be referenced.

A single version of the software is published, validated by the peer-review process. Future updates and improvements will not be hosted and distributed by the journal, so the authors must mention the location of future revisions.

The copyright, patent and license information provided in this file is an overview of the legal status of the code; it should be completed by reading the header of each source code file. An example for a complex case (multiple authors, patents and licenses) is shown in figure 4.4.

```
This program is written by Taro Yamada <taro.yamada@example.jp>
and Juan Perez <juan.perez@example.es> with contributions from
Marie Untel, ENS Cachan <marie.untel@ens-cachan.fr>.
- mmatch.c and rot_tree.c may be linked to the pending EU patent
  123.456 by Taro Yamada and Juan Perez and are provided for
  scientific and education only.
- demoz.c may be linked to the US patent 65.43.21 by Jane Doe;
  see the file for license terms.
- eizo.c and linalg_lib.c are distributed under the terms
  of the BSD license.
- All the other files are distributed under the terms of the
  LGPLv3 license.
```

Figure 4.4: Example of copyright, patent and license information.

4.2. Readability

> "*The authors must take care of the clarity of their program. It must be consistently indented and spaced. Lines should be limited to 80 characters and should not end with blank characters (spaces, tabs, ...). Files should not have more than 1000 lines. The line terminations should be the same (DOS/Windows CR+LF or UNIX CR style) for all the files of the program.*
> *Functions should be grouped by abstraction level in different source code files: the* `main()` *function, command-line processing and input/output calls in one file, the implementation of the algorithm described and reviewed in the IPOL article in one or more other files, and the implementation of auxiliary and external routines in one or more other files.*"

The source code is a text, both understandable by human readers and software compilers. It is the intermediate between the human-readable high-level description of an algorithm and the machine-readable compiled instructions.

The source code is written once and read many times. The original authors will re-read their code when they need to update it; the reviewers will read the code to analyze it; multiple other researchers will read the code when they need to use it, understand the algorithm internal details or modify it to produce another program. Because of this "written once, read many times" asymmetry, the authors must take care of the comfort of the readers and the time and effort they will spend to improve the visual quality of their code will be balanced by the time and effort saved by multiple readers.

In a published program, the source code is a primary material for the publication. It will be reviewed, published and read like any other part of the article. The text part of an article is generated by a text processing system into a visually clear and attractive document, and one usually only reads the final document, in PDF or HTML format. But the source code is read directly, the source code is the final document, so good layout rules must be applied to the code and it is the responsibility of the authors to take care of its clarity and readability.

The visual comfort one experiences when reading a source code is directly related to the indentation and spacing, and this aspect of the code layout directly influences how easily the code can be understood. The indentation of the instruction blocks visually conveys the high-level structure and control flow of the program [197]. Different indentation styles exist, are a matter of personal taste and are equally acceptable [324, "indent style" entry]; one can adapt to any indentation style chosen by the author of a code, but inconsistent styles are visually disturbing and uncomfortable.

Dense code without spaces is visually correct but difficult to read because we are used to read alphabetic languages by text segmentation, identifying words separated by spaces.

The 80 characters per line limit is not only a historic limit based on old console sizes, it is also good for the reading comfort. Modern consoles, editors and screens can display longer lines, but the ideal line length is based on the reader experience, not the ever-expanding screen capacities. Long lines are uncomfortable because the reader needs to move the eye or head to follow the line and looses the track of the current line when switching to the next line. That is why journals (newspapers and scientific journals) use a multi-column layout instead of long lines on the whole paper width. The "ideal line length" has 12 words

for web designers[23] and 66 characters for typographers[24]. This matches the standard 80 characters limit for source code with reserves for the indentation, as enforced by many major software projects and conventions[25].

The 1000 lines per file is another recommendation for the comfort of the reader. When a file is too long, scrolling back and forth becomes cumbersome and some time is needed to find the desired location in a file if it is too far from the current one. That is why web newspaper articles are split in many pages. Moreover, long files contain too many first-level objects for someone (other than the author) to keep everything in memory and have an understanding of what this code unit is about[26]. The 1000 lines limit encourages the reorganization of the code with more granularity.

But the line length or file length limits should never be enforced if they don't make sense programmatically. The priority is the quality and readability of the code, not some artificial metrics.

Maintaining a good and consistent indentation and spacing policy can be achieved with few efforts by automated code beautifiers, such as the Indent, Uncrustify or Astyle programs[ˆindenttools].

4.3. Implementation and Comments

> "*The source code of an IPOL program must be commented precisely and exhaustively. Authors should target the '1/8 comment/instruction ratio', but the quality of the comments is more important than the quantity. The source code must be written in English, including all variables, functions names and comments.*
> *Authors must ensure that the code is understandable, to the satisfaction of the editor and reviewers, so that consistency between the description of the algorithm and its implementation can be verified.*
> *Authors should apply simpler implementations when available, follow the conventions of the programming language, and use comments to explain implementation choices and every complicated or subtle point in the program. Clarity is more important than virtuosity.*"

The authors have an global understanding of the program because they wrote it and spent many hours in its development. For them, at least while they are developing it or shortly

[23] "*The ideal line length for text layout is based on the physiology of the human eye... At normal reading distance the arc of the visual field is only a few inches – about the width of a well-designed column of text, or about 12 words per line. Research shows that reading slows and retention rates fall as line length begins to exceed the ideal width, because the reader then needs to use the muscles of the eye and neck to track from the end of one line to the beginning of the next line. If the eye must traverse great distances on the page, the reader is easily lost and must hunt for the beginning of the next line. Quantitative studies show that moderate line lengths significantly increase the legibility of text.*" in *Web Style Guide* [236].

[24] "*Anything from 45 to 75 characters is widely-regarded as a satisfactory length of line for a single-column page set in a serifed text face in a text size. The 66-character line (counting both letters and spaces) is widely regarded as ideal.*" in *The Elements of Typographic Style* [60].

[25] This 80 characters line length is mentioned in the *Linux kernel coding style* [233] , the *Google C++ Style Guide* [365], the *Code Conventions for the Java Programming Language* [332] and the *Style Guide for Python Code* [352].

[26] Cognitive psychology experiments suggest that one cannot efficiently process more than seven new information items at a time [249]. Recent research suggests even less capacity with three or four items efficiently handled in the general non-trained case [121].

after, the code is always clear and easy to understand. But readers explore it one file, one function at a time and nothing is obvious in the code for them. They need to be guided by abundant comments explaining what the program is doing and how this is achieved.

Variables and functions, are as important as the comments because they refer to the concepts, mathematical or computational components of the algorithm, and this link should be clearly conveyed by their names. One-letter names mapped to mathematical variable names are only meaningful when one has the mathematical text in mind, and should be avoided in the implementation, or at least explained.

One of the goals of the peer-review process in a software journal is to certify that the algorithm implementation matches its description, that the same parameters are used, that the same processing steps are involved. To facilitate this manual review, the authors need to explicitly mention the correspondence between each step of the implementation and its description.

We also require every function to be documented with at least one line explaining what the function is doing, and the meaning of its parameters and return value. This is sometimes superfluous on trivial functions, but this general obligation is simpler to enforce than ad-hoc considerations about the simplicity of a function. With this rule, all the code can be prepared for documentation generators like Doxygen[27].

Tools like Cloc, Ohcount and Sloccount[28] can be used to count the comments, instructions and blank lines and evaluate the comment/instruction ratio, but this ratio should be a hint, not a rule. Simple comment metrics are artificial and can be abused. A long line could be split in five short lines without any added value. A ten line comment can be obscure while a single line may clearly provide all the information one needs. A wrong comment is worse than no comment at all. The "*1/8 comment/instruction ratio*" is only an order of magnitude of how much comment is expected. Less comments, good and useful, are preferred to more bad and useless ones.

Finally, our goal in publishing reproducible research is not to collect high-performance code but useful implementations understandable by other researchers. When the quest for better efficiency results in more complexity and obfuscation of the algorithm, it defeats the purpose of a source level publication. When a choice has to be done between efficiency and clarity, the authors should always choose clarity because it will benefit the other researchers. However, high-performance is welcome if it is well commented and achieved without obscuring the code. The distinction is subjective and left to the appreciation of the reviewers, who are the first ones who need to understand the implementation.

4.4. Example Data

> "*The authors should provide an example of input file to test the IPOL program and the result to expect when this input file is processed by the program.*"

Example input and output data serve three purposes. They are a concrete example of what the algorithm does, and contribute to its documentation. They are a mean to verify that the program performs as expected by the authors and is not affected by some properties

[27] `http://www.doxygen.org/`

[28] Cloc (`http://cloc.sourceforge.net/`), Ohcount (`http://ohcount.sourceforge.net/`) and Sloccount (`http://sloccount.sourceforge.net/`) compute metrics from source code files.

of the local environment. And they provide an input data suitable to the algorithm for the first tests performed by the users.

4.3 Examples and Online Test Service

The notion of "standard and documented implementation" was not sufficient to guide the authors, editors and reviewers. The adoption of these guidelines establishes a clear and shared understanding of what is expected from the authors. But their application depends on how they can be understood and verified. To help authors, editors and reviewers, a set of simple algorithms implemented by following these guidelines are provided as examples.

A test tool has also been developed to check a program against some of these guidelines. It is available via a web interface[29] and a downloadable command-line tool[30]. Authors and reviewers can use it to perform all the automatic tests on the file content and metrics, and focus on high-level analysis of the code once this filter is passed.

4.4 Automated Processing

All the previous guidelines items have been expressed for IPOL needs, but can be followed for any software distributed in source format to improve its usability. We can extend these guidelines to an automated identification, build and test procedure, needed for a software journal where multiple programs are regularly submitted and updated. With an automated procedure, the demonstration tools and the code documentation can be kept updated with minimum effort, and the reviewers can be assisted in verifying the code quality.

As of January 2012, the items detailed hereafter are only preliminary drafts and have not been proposed for adoption yet. We describe both sides of the automation: the configuration and support files needed in the program package, and the principles of automated building and testing used for the IPOL services.

The automated processing of IPOL packages is inspired from the `deb` format and tools of the Debian package management system [299, 339], which address similar automation needs. Our design is simpler because some usages of the `deb` packaging format, such as dependency handling and the notion of binary packages, are not relevant for a software journal like IPOL. When it made sense, we tried to maintain the Debian vocabulary and keywords.

These guideline items are only relevant for programs packaged for IPOL or similar systems. Unlike the previous guideline items, all the files supporting automated software build and test are designed to be machine-readable and processed by programs. They are stored in a special subfolder of the base `name_version` folder: the `ipol` folder. This folder contains two files, and nothing else: `ipol/control` and `ipol/rules`.

[29] `http://tools.ipol.im/pkg/`

[30] `http://dev.ipol.im/git/?p=nil/ipol_pkg.git`

4.4.1 Automated Identification

The `ipol/control` file contains some global information about the program. It is an ASCII text file composed of a series of fields expressed in one line as the field name, a colon (`:`) and the field value. White space (spaces or tabs) can be inserted immediately after the colon; a single space here increases the readability of the file. Every field name detailed hereafter must occur exactly once in the control file:

- `Standards-Version`
  The version of the guidelines the package follows.
- `Maintainer`
  The name of the person who prepared the package, *ie* who wrote the content of the `ipol` folder, followed by their email address in angle brackets (<>). The maintainer will usually be the main author of the IPOL article, but we can see situations when this will not be the case, for example when an editor assists the authors in packaging their program.
- `Package`
  The name of the software package, as already defined in section 1.2 of the guidelines.
- `Version`
  The version of the software package, as already defined in section 1.2 of the guidelines.
- `Homepage`
  The address of the reference web page for this package, where the current version can be downloaded. It is a single URL without any surrounding character, and should be the DOI[31] URL for published articles.

The `Standards-Version` field is needed to process the package according to the correct specification, which could change in the future. The `Maintainer` information provides a contact in case of issues with this package. And the `Package`, `Version` and `Homepage` fields can be used to display accurate information about the algorithm implementation used in the online demonstrations and their archives. A possible `ipol/control` file for the `axpb` package example is proposed in figure 4.5. Parsing these name-value pairs is an easy task with every scripting language. For example, the Python one-liner in figure 4.6 will read the `ipol/control` file into a dictionary structure.

```
Standards-Version: 1.00
Maintainer: Nicolas Limare <nicolas.limare@cmla.ens-cachan.fr>
Package: axpb
Version: 1.3
Homepage: http://tools.ipol.im/pkg/examples/
```

Figure 4.5: A possible `ipol/control` file for the `axpb` package.

[31] A Digital Object Identifier (DOI) is a globally unique identifier attributed to a work available in digital form. An URL, associated to this DOI, is a persistent network address usable to retrieve the work. Among other things, these identifiers are used to index and track research articles and to provide long-term means to access them (`http://www.doi.org/`).

```
dict([(name.lower(), value.strip())
      for (name, value) in [field.split(":", 1)
                            for field in open("ipol/control")]])
```

Figure 4.6: Python code to read an `ipol/control` file into a dictionary structure.

4.4.2 Automated Build

The building script is `ipol/rules`: a Makefile containing all the instructions to build the executable programs and documentation from the source. This Makefile is different from the one that may be provided with the source code, as mentioned in section 2.5 of the guidelines. The `ipol/rules` is only used for automated build and test of the program, and is not expected to be called manually or by normal users of the program. It is marked as executable and starts with the special line `#!/usr/bin/make -f`, so that it can be called directly by `./ipol/rules <target>`. This Makefile implements the following targets, executed by a non-privileged user from the base package folder:

- `default`
  required — This target must be the first one and its invocation must result in no action.
- `bin`
  required — This target compiles the package and places the resulting executable files in the `ipol/tmp/bin` folder, created first if needed. It should use the compilation procedure provided with the source code, as already mentioned in section 2.5 of the guidelines with standard cross-platform and cross-compiler options.
- `bin-amd64-linux-gcc`
  optional — This target compiles the package with options optimized for the GCC compiler in a Linux operating system running on an amd64 hardware platform.
- `doc`
  required — This target prepares the package documentation and places the resulting files in the `ipol/tmp/doc` folder, created first if needed.
- `clean`
  required — This target deletes every file and folder created by other Makefile targets.

The `bin` target aims at maximum future-proof portability. It can only invoke the POSIX shell and utilities[32], and standard compiler flags (this excludes OpenMP support, whose syntax varies across compilers). If a build-time test is available, it should be run in this target to verify that correct programs are produced.

With less portability, the `bin-amd64-linux-gcc` achieves better performances allowed by compiler optimizations and other tuning, including parallel processing with OpenMP. The demo system will first try to build the executable programs with this target, and fall back to the generic `bin` target if an error occurs. As the name of the target suggests, the same pattern may be used in the future with other hardware platforms, operating systems and compilers.

[32]The 2008 version of the POSIX specification includes a list of the standard shell and utilities and their expected behaviour (`http://pubs.opengroup.org/onlinepubs/9699919799/utilities/contents.html`).

The documentation produced by the `doc` target includes, at least,
the `README.txt` file mentioned in section 4.1 of the guidelines, and can also include a source documentation generated with Doxygen. HTML documentation should be in an `ipol/tmp/doc/html` subfolder. This target will be used to produce the documentation provided with the source code, on the IPOL article web pages.

All these Makefile targets must be non-interactive and multiple invocations must not trigger errors. As showed by the files proposed for the `axpb` and `imgdiff` package examples in Annex B, `ipol/rules` should use the Make or CMake build procedure already available with the source code. Other Makefile targets and custom variables are allowed, but must start with an underscore _; this name space is reserved to avoid conflicts between freely defined targets and future updates of the guidelines.

The Makefile syntax is used for the build rules because, in this restricted context, `ipol/rules` is only a list of shell commands to be run for each target, plus the possibility of local definitions and variables. Any other solution would either be more complex or introduce a new *ad hoc* syntax.

Build Environment

The `ipol/rules` file is intended for use with a chroot tree, a set of files and folders replicating the file hierarchy of a computing environment. With a wrapper like the `chroot`, `fakechroot` or `lxc` commands[33] on Linux systems, programs are executed in this view of the filesystem, called a chroot environment and cannot access other files.

A chroot environment has three advantages for us. Its file tree is a stable reference filesystem which can be replicated on different computers to achieve reproducible code compilation. It restricts the local resources available to programs executed inside the environment and ensures that only explicitly allowed dependencies are used to compile and execute a program. And it protects the host system against resource abuses and accidental or hostile actions that may happen during the build procedure.

A chroot environment is not comparable to the virtual machine model used by some reproducible research initiatives to distribute, execute and archive a program. In our context, it will only be used to build the packages from source in a controlled environment, and the chroot content will be continuously updated.

The standard chroot tree for IPOL packages contains all the basic POSIX utilities[ˆposixutils] used to build a program, a C/C++ compiler and the libraries whose use is allowed. In addition, the Doxygen and dot programs are available to compile the source documentation. Everything is in a FHS-compliant[34] layout. No use of this chroot environment is expected to require administrator privileges, so all the files can be owned by non-privileged users, which means that a standard chroot tree can be archived, transferred and deployed without requiring administrator rights.

Packages will be built as follows:

[33] The `chroot` command (`http://www.freebsd.org/cgi/man.cgi?query=chroot`) and system call are available in UNIX systems to restrict a process to a subset of the filesystem. The `fakechroot` command (`https://github.com/fakechroot`) adds the possibility for a non-privileged user to run the chroot command, with less isolation. The `lxc` virtualization layer for Linux (`http://lxc.sourceforge.net/`) adds some processor, memory and network resource restrictions to the basic filesystem containment.

[34] The File Hierarchy Standard is the reference specification of the location of file resources in the filesystem of Linux (and some other UNIX) systems (`https://wiki.linuxfoundation.org/en/FHS`).

1. a pristine standard chroot tree is deployed and the package archive is extracted inside this tree;
2. via chroot or a similar wrapper, the `ipol/rules` are used to build the binary programs or the documentation; the `bin-amd64-linux-gcc` target is invoked first and if it fails, the `bin` target is used instead;
3. the `ipol/tmp` subfolders are copied to their destination, *ie* the location of the demo executable programs or the online documentation pages;
4. the chroot tree is discarded.

With all the compilers, libraries and build tools needed to compile an IPOL program in addition to the standard management and system tools, a base Debian system image weights 350MB[35] (75MB compressed). This includes many unnecessary files, such as the system and network tools, a Perl interpreter with libraries, localization support files and all the documentation. After trimming, a lighter chroot tree based on this Debian version can be obtained around 200MB (40MB compressed), and most of this size is used by the gcc compiler and linear algebra libraries. Further downsizing could be achieved by using lightweight alternatives, such as BusyBox[36] for most of the command-line utilities, if not all of them, in 500KB, or alternative compilers. The relatively small size of a chroot tree means that it can be published, regularly updated and distributed as a reference computing environment for a programming community, such as the IPOL authors. But as the guidelines already stated, this computing environment should not be mistaken for a portability standard; portable implementations must be usable with other hardware, operating systems and compilers.

The goal of the implementation guidelines detailed before is to help the authors avoid portability issues with their implementations. A reference environment has a different function: for IPOL, it provides a precise definition of how the program will be compiled and helps the authors get the best performances for their online demo. Moreover, the demonstration platform being similar to the typical computers used by many researchers (amd64 hardware, Linux operating system, GCC compiler), achieving good performances on the reference environment will usually mean that these performances are available for the general user.

Example `ipol/rules` files

The `ipol/rules` files in figures 4.7 and 4.8 are adapted to the `axpb` and `imgdiff` example packages[37]. They use Make and CMake to compile the implementations of the algorithms.

4.4.3 Towards Automated Tests

A software journal like IPOL asks that the published source code meets some quality criteria, chosen to maximize the present and future usefulness of the code. Some of these

[35] This size was obtained from a standard Debian 6.0 Squeeze distribution with the `cdebootstrap` installer, with the additional packages `binutils`, `gcc`, `g++`, `make`, `cmake`, `libpng12-dev`, `libtiff4-dev`, `libfftw3-dev`, `libatlas-dev`, `liblapack-dev` and their dependencies. The chroot tree was compressed with `xzip`.

[36] BusyBox combines tiny versions of many common UNIX utilities into a single small executable for use in resource-constrained computing environments (`http://www.busybox.net/`).

[37] `http://tools.ipol.im/pkg/examples/`

```
#!/usr/bin/make -f

# folder locations
TMPDIR  = ipol/tmp
BINDIR  = $(TMPDIR)/bin
DOCDIR  = $(TMPDIR)/doc

# build the program
bin :
    make axpb
    mkdir -p $(BINDIR)
    cp axpb $(BINDIR)

# compilation options for amd64/linux/gcc
CFLAGS  = -O3 -ffast-math -march=native -funroll-loops -fopenmp

# build optimized programs
bin-amd64-linux-gcc :
    make CFLAGS="$(CFLAGS)" axpb
    mkdir -p $(BINDIR)
    cp axpb $(BINDIR)

# build the documentation
doc :
    make doc
    mkdir -p $(DOCDIR)
    cp README.txt $(DOCDIR)
    cp -a doc/html $(DOCDIR)

# cleanup the build files
clean   :
    make distclean
    rm -rf $(TMPDIR)

.PHONY  : default bin bin-amd64-linux-gcc doc clean
```

Figure 4.7: A possible `ipol/rules` for the `axpb` package.

```
#!/usr/bin/make -f

# folder locations
TMPDIR  = ipol/tmp
BINDIR  = $(TMPDIR)/bin
DOCDIR  = $(TMPDIR)/doc
BUILDDIR    = build

# build the program
bin :
    mkdir -p $(BUILDDIR)
    cd $(BUILDDIR); cmake ../
    make -C $(BUILDDIR)
    mkdir -p $(BINDIR)
    cp $(BUILDDIR)/imgdiff $(BINDIR)

# compilation options for amd64/linux/gcc
CFLAGS  = -O3 -ffast-math -march=native -funroll-loops -fopenmp

# optimized build for the imgdiff package, using cmake
bin-amd64-linux-gcc :
    mkdir -p $(BUILDDIR)
    cd $(BUILDDIR); CFLAGS="$(CFLAGS)" cmake ../
    make -C $(BUILDDIR)
    mkdir -p $(BINDIR)
    cp $(BUILDDIR)/imgdiff $(BINDIR)

# build the documentation
doc :
    mkdir -p $(DOCDIR)
    cp README.txt $(DOCDIR)

# cleanup the build files
clean   :
    rm -rf $(BUILDDIR)
    rm -rf $(TMPDIR)

.PHONY  : default bin bin-amd64-linux-gcc doc clean
```

Figure 4.8: A possible `ipol/rules` for the `imgdiff` package.

criteria are a matter of software design and documentation and we believe it can only be evaluated by the reviewers. Another criterion is the exact implementation of the algorithm as defined mathematically in the article. Such a property could be formally proved with the help of proof assistant systems[38] [150], but this requires some understanding of lambda-calculus and formal proof tools, unlikely to be available in the image processing community. Moreover, authors are asked to provide a correct implementation, but not yet to prove that the implementation is correct.

A less ambitious but more realistic goal is the correctness and robustness of the code, roughly defined as following the programming language grammar, syntax and conventions, and producing the same results on any compliant computing environment. We cannot prove the code correctness without the formal proof model already mentioned, but we can submit the code to a suite of tests to detect some implementation errors. We present hereafter a collection of possible test strategies to detect the presence of known bugs.

Compilation Test

The compilation is the first test of the code. If it doesn't compile, or if the compiler throws some error or warning messages, then something is wrong with the implementation. With the automated compilation rules previously introduced, one can verify that every version of every program package can be compiled in the reference environment.

This build environment can be customized to override the compiler, for example by replacing it by a wrapper script, in order to use another compiler. Different compilers have different warning options and perform different tests on the code. Trying to compile the code with different compilers helps detect bugs which would not have been spotted otherwise.

If a build-time execution test is provided with the building script, we can verify that, at least with some input data and parameters, the implementation produces a correct program with every compiler tested.

Every individual source code file can also be compiled into object code, with options restricting the compiler to a programming language standard version. For such file-level compilation, one needs to know the compilation flags needed (inclusion path, preprocessor variable) and the code standard claimed to be followed by the authors (C89, C99, C++98). This information could be provided as a tag in the source code or an `ipol/fileinfo` file, or maybe collected from the cmake build environment. The method is yet to be decided and refined.

These cross-compiler strict compilation tests can be completed with a cross-platform battery. With different build environments, one tests the code compilation in other operating systems or variants of the same operating system. Then these environments can be used on other machines with different hardware platforms or emulated architectures to verify the portability of the code.

[38] Coq (`http://coq.inria.fr/`) and Isabelle (`http://isabelle.in.tum.de/`) are two possible tools to achieve the formal proof of the correspondence between the specification and the implementation of a program.

Static Analysis

Static analysis is another category of tests, performed without compiling the program. The source code is parsed and the result is used to detect common error patterns. In addition to commercial products and services of the software industry, a collection of static analysis tools are freely available[39]. These tools differ by which programming language they can process and which errors can be found:

- `splint` checks C programs for security risks and coding errors: invalid pointers, undefined values, type mismatch, memory management, inconsistent function interfaces, infinite loops, dead code, and buffer overflows;
- `cppcheck` performs static analysis on C and C++ code: it looks for array boundary errors, memory leaks, and uninitialized variables;
- `clang` is a C/C++ compiler with static analysis: it checks the correctness of the standard function calls, the consistency of interface functions, divisions by zero, NULL pointer dereferences, and uninitialized values;
- `uno` is a simple C code analyzer, focusing the three most common errors: uninitialized variable, invalid pointers, array boundary violation;
- `lint`, distributed with the SunStudio C/C++ compiler, detects patterns in a C code that are likely to be bugs, non-portable, or wasteful: type mismatch and overflow, dead code, unused or uninitialized variables, flawed conditions, inconsistent function calls and return values, unused inclusions, usage of deallocated memory or deallocation of non-allocated memory;
- `rats` and `flawfinder` look for dangerous code patterns and potential security risks in C/C++ code.

Each one of these tools comes with its own usage model, call syntax and output format and we cannot expect every contributor to use them. Instead, a service can be provided to authors and reviewers, where the code is tested with the aforementioned tools and their reports are combined to automatically produce an assessment of the code quality. This service would help improve the quality of the implementation before it is submitted and perform some bug screening during the review. Like the compilation tests described before, accurate static analysis may require file-level information such as the programming language standard and compilation flags.

Runtime Tests

After the compilation and static analysis, a third kind of tests can take place during the execution of the compiled program. With some tools like Valgrind[40] or Electric Fence[41],

[39] Splint (`http://splint.org/`), Cppcheck (`http://cppcheck.sourceforge.net/`) Clang (`http://clang.llvm.org/`), Uno (http://spinroot.com/uno/), RATS (`https://www.fortify.com/ssa-elements/threat-intelligence/rats.html`) ans Flawfinder (`http://www.dwheeler.com/flawfinder/`) are or include static source code analysis tools. The SunStudio has been renamed Oracle Solaris Studio, and is distributed free of charge from the Oracle web site (`http://www.oracle.com/`).

[40] Valgrind (`http://valgrind.org/`) is a framework for dynamic analysis tools. The tools available include call, cache and heap profilers and memory and thread debuggers.

[41] Electric Fence (`http://perens.com/FreeSoftware/ElectricFence/`) and its variant DUMA (`http://duma.sourceforge.net/`) redefine the memory management functions, to ease the detection of wrong memory access.

one can detect memory management problems such as uninitialized or unreleased memory and access to wrong memory locations. The advantage of tools like Valgrind over other memory debuggers[42] is that no modification of the source code is required, and the program is compiled as usual, but executed under the supervision of the memory debugger, which reports all the problems detected.

Such debuggers will not find all the possible memory management errors in a program. They will only spot the errors happening to a given execution of the program. If some errors are only triggered when some parameters are used or some unusual data is processed, they will probably not be found in automated tests. Nevertheless, finding common error bugs happening during a typical execution of the program is an improvement and should not be overlooked.

To perform such runtime tests, we need the authors to provide some examples of the usage of their program. This could be combined with the build-time execution tests that may be provided in the automated build procedure.

Finally, if the program can be used via a public web interface with free input, this can be another way to test it processing unexpected and unusual data submitted by random users of this service. Unfortunately, the performance cost of a memory debugger like Valgrind is currently too prohibitive ($\times 10$) to be used in a realistic web demonstration interface. But with a simple monitoring of the exit status of the program, and archive of the input, parameters and output, and the debugging information provided by the operating system on memory segmentation errors, we can collect a fairly good assessment, over time, of the robustness of the program.

On Tests

Software testing is a rich discipline, with plenty of free and commercial tools and many techniques. In addition to the compilation, static analysis and runtime tests, we could for example cite unit tests, whose function is to ensure that every function in the code performs the expected actions, fuzzing, which is used to assess the robustness of the program against invalid or unexpected data, or the systematic declaration and verification of invariant properties expected on the data at the function boundaries.

As Dijkstra famously remarked, "*testing can be used to show the presence of bugs but never to show their absence*" [105]; the software testing task is endless and we can always add another layer of tests to catch rare but possible programming errors. Trying to find and much bugs as possible is like pursuing the myth of an absolutely safe program. But the essential question is not how to test but what to test, and it defines how much effort one wants to invest in the testing procedures and how intrusive in can be on developer's work.

The three categories of tests presented before can be used to build a flexible and extensible testing procedure. First, the compilation tests only require an automated compilation procedure, which is already required as soon as we want to automatically build the programs, for an online demo system for example. And with some infrastructure work, these compilation test can catch most portability issues. Then, if the authors provide some information about how each source file is to be compiled, static analysis can detect erroneous or suspicious constructs dangerous in the code. A static analysis service can start with

[42] Dmalloc is a popular memory debugging library (`http://dmalloc.com/`). Mtrace is another memory debugger, built in the GNU C library. Both require some modification of the source code, at least to activate the debugging functions.

one analysis tool, and be gradually expanded to a collection of these tools. And finally, if the author provide some usage examples for their code, memory errors can be found with dynamic analysis tools in real usage cases.

We can hope that, in the coming decades, computer scientists will be required to prove the correctness of their programs, like mathematicians today are expected to provide a proof of their theorems. Until adequate tools and abstractions are available to assist us in these requirements, software tests can help us detect and avoid common programming pitfalls.

Chapter 5

Copyright, Patents, Licenses and Network Laws

Contents

Abstract

In this chapter, we try to collect and summarize how copyright and patents apply to computer programs in a computational science research context. Copyright and patents are governed by distinct laws, treaties and jurisprudence, with national and international regulations. We only explored the situation in the United States, the European Union and France.

Then we propose a policy for journals publishing articles with software and datasets, based on the Open Access, Open Source and Open Data guidelines. Unlike the usual policies in use in the academic publishing industry, this one is designed to facilitate the dissemination of the research works and the developments based on this research.

The chapter ends with a review of the laws and regulations applicable to the operation of a journal as a web site in the French jurisdiction, including the consequences of an online demo service with user-submitted content and archives.

5.1 Software Copyright and Patents

Copyright and patents are two distinct legal protections systems: the former attributes moral and economic rights to the author of a creative work and the latter provides an exclusive exploitation right in exchange for the publication of an invention. Both can be involved and may conflict in the implementation of an algorithm, and these legal considerations must be understood to publish a software.

This is not the work of a lawyer. It only tries to build a usable overview of this topic from a practical researcher perspective. Further details on the complex question of software copyright and patents can be found in jurist reviews[1] [26, 189, 359, 361].

5.1.1 Copyright

The Berne Convention for the Protection of Literary and Artistic Works [37] protects "every production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression" since the beginning of the 20th century [36]. Computer programs were explicitly included to the list of protected works during the 80s and 90s in most national laws, with the development of the computer industry.

United States courts [268] and laws [347] attributed to software the copyright status of literary works. In the European Union, the Computer Programs Directive [118] states that a computer program can be protected as literary and artistic work. The French Intellectual Property Code (*Code de la propriété intellectuelle*) [133, article 112-2] includes computer software in the list of intellectual works (*œuvres de l'esprit*) submitted to copyright regulations.

All of these laws have similar provisions: a program can be copyrighted if it is the author's own creation. All the forms are protected, including the source code (human and machine readable form) and object code (machine readable form). The copyrighted materials include the documentation, and preparatory and design documents. The ideas and principles underlying the program are not copyrighted.

[1] The works of Lawence Lessig [217] Richard Stallman [321] also provide a detailed critical analysis of the copyright and patent system at the Internet era.

Moral and Economic Rights

As defined by the Berne convention, the author of a copyrighted work owns two kinds of rights: moral rights and economic rights. Moral rights on a computer program include the right to claim authorship and to protect the reputation of the author against prejudicial modifications of the program. Economic rights include the rights of distribution and reproduction of the program, and licensing its usage. In the European Union countries, these two categories are governed by the same body of laws. In the United States, moral rights are covered by defamation laws rather than copyright laws, but this technical distinction has no impact on the regulation and protection of these rights.

In France, a computer program is a special exception in the Intellectual Property Code [133, article 113-9]: the economic rights are not owned by the author of the software, but automatically transferred to the employer. The author-employee only retains the moral rights. Thus, the economic right on research software developed by researchers in the course of their work belongs to the research institution [81, 152, 153].

We are not aware of a similar exception for software in United States laws, but computer programs can be considered "work made for hire" [347] and, as such, belong to the employer. Research software and all the associated rights would then belong to the research institution.

Article *vs* Software in France

Research articles are qualified for copyright protection as intellectual work and the authors own moral and economic rights attached to their articles[2]. For this reason, the publication of research articles is often subject to a copyright agreement between the authors and the editor.

But the nature of software is hybrid. It can be considered as an intellectual work, and as an industrial and commercial product. The exceptions and provisions for computer software in copyright laws are adapted to the requirements of the software industry but create a copyright dissonance for computational research: researchers own all the the rights of the articles they write to describe their programs, but not the rights of the programs they write to implement, illustrate, verify or prove their article [154].

This dissonance becomes obvious when a pair of research articles and software, by the same researcher, include exactly the content written twice, one in human-readable natural language and one in human and computer-readable programming language. These two different presentations of the same concepts can even be created and distributed as a single document with the article and the software texts interleaved in a continuous and intelligible document, a method known as literate programming [201] and widely used in the statistics research community [215]. And the distinction between article and software is blurred when the editorial policy of a journal is to integrate computer programs in the articles they publish [185, 282]. Deciding who owns the economic rights of a work which is a research article *and* a research software would probably be a difficult case, which has not hit courts yet.

[2]This was confirmed for French public agents, in an exception for science and technology institutions by the DADVSI law [11]. See also *Veni, Vidi, Libri : Les contraintes contractuelles / L'auteur fonctionnaire* [fr] (`http://www.venividilibri.org/fr/Les+contraintes+contractuelles#L-auteur_fonctionnaire`).

Other problems hinder the acceptability of research software as a product owned by the employer, such as the questionable qualification of students and interns as employees [81, 152]. This leads to the conclusion that the software industry needs for computer program copyright are not adapted to research. In our opinion, copyright laws should be revised to exclude research software from the software exceptions. A research software is a by-product of the research activity among others, like data sets, algorithms, articles and books, and all these intellectual works should be governed by the same unified copyright regime.

5.1.2 Patents

Patents are exclusive exploitation rights granted to inventors on their inventions during a limited period of time. This exclusivity includes the rights to make, have make, sell, offer or license their invention. The patent system was created to stimulate the exchange of ideas and creativity, as an incentive to publicly disclose inventions in exchange for this temporary exclusivity. To be patentable, an invention must be new, useful, non-obvious and susceptible of an industrial application.

And anyone competing with the exclusive rights, with or without a commercial activity, is infringing the patent regulation. Encouraging or providing assistance to the infringement of a patent is also a source of patent infringement liability.

National Regulations and Software Patents

The Patent Cooperation Treaty was an international agreement on patent applications. The recent Patent Law Treaty was an harmonization of the procedures. But patents are a national prerogative, delivered by national patent offices under the control of national courts. There is no international patent, nor international patent jurisdiction. The exclusive exploitation of a patent is only granted to an inventor in the countries where the patent was delivered, and this exclusive right is defended or challenged in national courts.

Moreover, there is no international agreement on what is patentable. There is a general agreement on the conditions of patentability (novelty, originality, utility and applicability) and on the exclusive right granted to the inventor. It is also generally recognized that an idea is not patentable, neither are theories and mathematics, and that patented inventions must provide a technical mean to achieve an effect. But the meaning of these words is debated, and there is no agreement on the patentable subject matter. The patentability of computer programs is a hot question and different courts, laws, patent offices, industries and communities have different opinions.

In the European Union Since the European Patent Convention [116], patent applications in Europe follow a unified framework and are managed by the European Patent Office. But the unification only covers the procedure to deliver a portfolio of national patents, and it does not replace national patent offices. Recent attempts to reform the European Patent Convention and create a common concept of patents backfired, and have been rejected by the European Parliament, in part because of the issue of software patents.

In the current situation, the European Patent Convention describes a framework used by each member country and the European Patent Office to define their patent policy. But patents remain a national prerogative, with different interpretations across the European

countries. The European Patent Office and national courts sometimes disagree on what is patentable. In that case, the European Patent Office is not binding on national courts and only the national court rulings matter because in the absence of an European patent, inventions are protected and challenged at the national level.

On the patentability of computer programs in computational sciences, the European Patent Convention, article 52, says that patents shall be granted to new, inventive and applicable inventions, but not for discoveries, scientific theories, mathematics or computer programs *as such*[3]. This "as such" and its interpretation are the source of most of the debate. A program cannot be patented but it may be possible for inventions involving software, such as computer-controlled devices and processes.

The central question, still not resolved, is "when is a software the subject-matter of the patent and when is it only a part of the invention?" In the context of computational sciences, the difficulty may be to find a place for the definition of "a computer program" between the mathematics (not patentable, excluded by the EPC) and the implementation (copyrightable but not patentable either). One could argue that there is a continuum between the description of an algorithm as a mathematical method and its expression in a computer program, a continuum of successive refinements from a high-level description in natural language to a low-level detailed expression in a form understandable by a computer.

In France, as early as 1968, "programs or series of instructions for the operations of a calculating machine"[4] were explicitly not to be considered as industrial invention in the Law 68–1 on inventions and patents [17, article 7]. The current Intellectual Property Code (*Code de la propriété intellectuelle*)[5] [133, article 611-10] is a close translation of the European Patent Convention article 52.

In 2003, French courts denied the patentability of processes only involving computer programs in the *SAGEM* case [89]. The United Kingdom Patent Office took similar decisions in 1997 in *Fujitsu's application* [113] and later in 2006 in *Aerotel v. Telco and Macrossan* [114], as well as the German Patent and Trade Mark Office in 2004 with *Rentabilitätsermittlung* [68] and 2007 in *Informationsübermittlungsverfahren* [69].

We conclude from this jurisprudence that, notwithstanding the European Patent Office position on patentability of some computer programs, software patents are currently not recognized by the European national jurisdictions. But the multiple legal initiatives in the recent years calling for unification and enlarged patentability at the European (European Patent Litigation Agreement, Unitary Patent) and international (Trade-Related Aspects of Intellectual Property Rights - TRIPs) levels suggest this situation may be challenged. Meanwhile, the software development and distribution by computational science researchers seems safe from patent infringement in Europe.

[3] "1- European patents shall be granted for any inventions which are susceptible of industrial application, which are new and which involve an inventive step. 2- The following in particular shall not be regarded as inventions within the meaning of paragraph 1: discoveries, scientific theories and mathematical methods [...] and programs for computers [...]. 3- The provisions of paragraph 2 shall exclude patentability of the subject-matter or activities referred to in that provision only to the extent to which a European patent application or European patent relates to such subject-matter or activities as such."

[4] "*les programmes ou séries d'instructions pour le déroulement des opérations d'une machine calculatrice*"

[5] "1- *Sont brevetables les inventions nouvelles impliquant une activité inventive et susceptibles d'application industrielle.* 2- *Ne sont pas considérées comme des inventions au sens du premier alinéa du présent article notamment [...] les découvertes ainsi que les théories scientifiques et les méthodes mathématiques [...], ainsi que les programmes d'ordinateurs*; 3- *Les dispositions du 2 du présent article n'excluent la brevetabilité des éléments énumérés auxdites dispositions que dans la mesure où la demande de brevet ou le brevet ne concerne que l'un de ces éléments considéré en tant que tel.*"

In the United States In 1952, the United States Patent Act stated that "*any new and useful process, machine, manufacture, or composition of matter*" [348] could be patented. In the United States legal system and tradition, the Patent Act only provides a general principle, to be refined by court decisions in case law.

In 1994, a patent for using a smoothing algorithm in a measure and display device was approved in *In re Alappat* [269] by the Unites States Court of Appeals. This court also ruled in 1998 that an invention only needs to produce an useful, concrete and tangible result to be patentable in *State Street v. Signature* [271]. These decisions prompted a sudden increase of software patents claims, and the United States Patent and Trademark Office adjusted its guidelines: mathematical theories and algorithms are confirmed not to be patentable, but their particular practical use in an invention can be patented [285].

However, recent decisions of the Court of Appeals are reverting the tendency, and restricting the patentable subject-matter:

- In 2008, the court in *In re Bilski* [272], confirmed by the Supreme Court in *Bilski v. Kappos* [279], refused a patent for a method to process and transform abstract information (financial data). However, inventions to process signals representing physical objects were not excluded.
- In 2011, the court ruled in *Cybersource v. Retail Decisions* [273] that if an invention is based on simple calculations, it is not more than a mental process and cannot be patented [213].

Three United States Supreme Court decisions are now considered the canon on which future decisions are to be built:

- In 1972, the court ruled in *Gottschalk v. Benson* [276] that a numerical algorithm was not patentable and confirmed a Supreme Court decision from 1852 in *Le Roy v. Tatham* [275] that abstract ideas and scientific principles were not patentable because they were universal truth.However, this decision did not prevent any computer program to be patented.
- In 1978, the court ruled in *Parker v. Flook* [277] that an invention using a mathematical algorithm is not patentable if its only innovation is the use of the algorithm.
- In 1981, the court ruled in *Diamond v. Diehr* [278] that a patent on an invention with a physical effect could not refused on the sole reason that a computer program was involved in the process. Meanwhile, the impossibility to patent mathematical formulas was confirmed.

Recent decisions confirm a new tendency to restrict the patentable subject-matter, but software patents are still currently delivered in the United States. Under the hypothesis that computer programs and algorithm could be patented, what is a patent infringement? The patent holder has the exclusive right to produce distribute or license the invention, so any non-licensed usage of a compiled program implementing the patented algorithm could be an infraction, because it involves a prejudice to the commercial exploitation of the invention. Implementing the patented algorithm in source code from existing documentation of the invention, including the patent description, is allowed, but if this implementation is distributed, even in source code form, it can be considered inducing patent infringement by others, by distribution of means.

And even the definition of a computer program by United States courts is not clear enough to consider safely that a binary machine code executable file is a program and a source code file is not [91]. However, the United States Court of Appeals ruled in *Bernstein v. the United States* [270] that source code of computer programs are free speech and protected by the First Amendment. This could be sufficient to allow safe distribution of the implementation of patented algorithms in source code form, but this has not yet been tested in a patent litigation.

Exceptions for Research

The original goal of the patent system was to stimulate innovation and the circulation of ideas, by providing an incentive for inventors to disclose their inventions, in exchange for a limited exclusivity. For this reason, patents are publicly available and patent laws have an exception: it is allowed to create, test and use the patented inventions as long as it is only for "research and experimentation" and does not conflict with the commercial exploitation of the invention.

In the French Intellectual Property Code [133, article L613-5], the exclusivity granted to the inventor does not include experimentation on the invention[6]. A similar provision in the German patent law is used for the development and distribution of a source code implementing sound decoding algorithms [335] in spite of patents pending on this algorithm [56] and enforced by the patent holders [179]. And as early as 1813, in *Whittemore v. Cutter* [240], an United States court decision stated that the intent of the patent system was not to punish someone who infringes "merely for [scientific] experiments, or for the purpose of ascertaining the sufficiency of the machine to produce its described effects."

5.1.3 Conclusion

We have seen that copyright and patents are two distinct, legal concepts. These two may sometimes be conflicting, for example if the distribution license of a software claims to allow the usage of a program for any purpose, while a patent attributes this exclusive right to the inventor. Moreover, the exact definitions and consequences of software parents are not the same in every country, and changing with the public understanding of software matters. Researchers are used to universal, perpetual science truth, not national regulations with changing interpretations.

The scientific project of the IPOL journal requires the public availability of computer programs for every algorithm published, in order to enable the validation of the scientific results and reuse of this work to build new algorithms. We address these goals and the complexity of patent and copyright regulation as follows:

- If a patent is known by the authors to be related to the implemented algorithm, then a patent warning is inserted in the source code. Neither IPOL nor the authors claim that the program implements a patented algorithm, as only a lawyer may be qualified. Moreover, the validity of such a statement and its consequences on the usage of the program depend on the jurisdiction, so IPOL and the authors only provide a warning, to the best of their knowledge.

[6] "*Les droits conférés par le brevet ne s'étendent pas [...] aux actes accomplis à titre expérimental qui portent sur l'objet de l'invention brevetée*"

- No copyright agreement requires the transfer of the economic rights from the authors to the editor. The authors keep free to re-distribute or re-license their software. IPOL always distributes the implementations in source code form. If the authors of the algorithm are the patent holders for the algorithm implemented, then the source code is distributed "for research only". Otherwise, a free software license is used.

Two possible objections are that the web interface to the execution of the algorithm provided by IPOL competes with the commercial exploitation of the patent, and that the distribution of the source code of patented algorithms helps others infringe on the patent.

For the first point, only functions accessible via the web interfaces of a journal like IPOL are the execution of a mathematical algorithm. But the case law has a tendency to protect the industrial applications of inventions and deny the patent protection for "pure algorithms". The online tools provided by a journal only have a demonstration value and do not compete with complex industrial systems. They can only process data of a modest size and do not compose a complex processing chain or compete with a commercial image processing software.

For the second point, IPOL does not allow any use of the source code to circumvent a patent regulation: the patent warning clearly mentions that the free license terms are only valid if they do not conflict with patent regulations in the local jurisdiction. Moreover, we consider that the implementation of an algorithm is the only complete description of such an algorithm. The patent system already requires the complete description of an invention to be publicly available in the patent documents, so one can argue that a third-party implementation does not provide any information not already available in the patent record. Finally, the vocation of a computational science journal is the study of algorithms, so we expect IPOL works to be covered by the patent infringement exception for research. The patent system was created as an incentive for public disclosure of inventions, and that disclosure is practiced by the public research community and its scientific journals.

5.2 Copyright and License Policies

The rights retained by the author of a scholarly journal, those transferred to the publisher, and those exercisable by the readers of the journal, are defined by the copyright and license policy of the journal, agreed by the author and publisher.

5.2.1 An Open Policy

We based our policy on the following established successful practices for manuscripts, software and datasets:

- Open Access publishing allows unrestricted access and redistribution of a work. This policy has been adopted by 7500 journals, registered in the Directory of Open Access Journals [106] in early 2012. According to the Scholarly Publishing and Academic Resources Coalition — Europe (SPARC Europe), over 10% of the research journals have an open access policy [320], and some open access publishing initiatives such as the Public Library of Science (PLoS) and BioMed Central proved successful with

highly regarded content. 58% of the computational science journals have an Open Access policy[7].

- Open Source licensing allows anyone to use, study, redistribute and modify a software. These licenses have revolutionized the software development and distribution since the first efforts led by Richard Stallman and the Free Software Foundation from 1985. With the advent of the Internet, Free Software is a fundamental building block of vibrant, innovative and successful development communities and a major vector of technical innovation in leading commercial or research projects[8] [175, 368, 369].
- Open Data means that anyone can use and redistribute the data data without financial or legal restrictions. This has been a common practice for a long time, from the Evaluated Nuclear Data File (ENDF), GENBANK and Cambridge Crystallographic Database in the 1980s [198], to the genetic databases currently managed by the National Center for Biotechnology Information (NCBI) [129]. Many datasets in image processing are freely shared, like the Kodak video benchmark dataset [374] or the SFU color constancy dataset [31]. Our policy only adds an explicit license to clarify the usage and redistribution rights.

In addition, we avoid the unnecessary transfer of the copyright to the publisher and we add some measures to avoid exploitation of the research work by third parties without benefits for the research community. The result is very close to the Reproducible Research Standard[9] legal framework proposed by Victoria Stodden [326, 327].

With a permissive copyright and license management, this policy removes any unnecessary barrier to creative research collaboration, yet confirms the authorship, requires the attribution of the works to their authors, and make these works accessible to the research community.

5.2.2 Copyright and License Agreement

As an example of possible copyright and license agreement between an academic journal publishing software and datasets and its authors, we propose the following terms. Written for the IPOL journal, they define what the authors and the publisher are allowed to with the published works.

> *Authors who publish with this journal agree to the following terms:*
> *- **manuscript**. Authors retain copyright and grant the journal right of publication with the manuscript simultaneously licensed under a Creative Commons*

[7] Unpublished study by Stodden and Guo, preliminary results [329].

[8] The SourceForge service (`http://sourceforge.net/`) hosts more than 300000 open source software projects; the Ohloh software directory (`http://www.ohloh.net/`) references more than 500000 open source projects (circa February 2012). Firefox, OpenOffice, VLC are well-known free software products for the general public. GCC, R, OpenCV, Scilab, TeX, FFTW are other free software tools used by scientists.

[9] "*The Reproducible Research Standard is a way for scientists to publicly mark their work as reproducible, meaning that certain conditions are satisfied:*
1. The full compendium is available on the Internet,
2. The media components, including the original selection and arrangement of the data, are licensed under CC BY or released to the public domain under CC0,
3. The code components are licensed under one of Apache 2.0, the MIT License, or the Modified BSD license, or released to the public domain under CC0,
4. The data have been released into the public domain according to the Science Commons Open Data Protocol."

Attribution Noncommercial Share Alike License. This license allows others to share and reuse the manuscript for noncommercial purposes with an acknowledgment of the work's authorship and initial publication in this journal.
*- **software**. Authors retain copyright and grant the journal right of publication with the software licensed under a Free Software license as specified by the Software Guidelines. This license allows others to use, study, redistribute and modify the software. Some conditions can be added depending on the license chosen.*
*- **dataset**. Authors retain copyright and grant the journal right of publication with the dataset licensed under a Creative Commons Attribution License. This license allows others to use, redistribute and modify the dataset with an acknowledgment of the work's authorship and initial publication in this journal.*
- Authors are able to enter into separate, additional contractual arrangements for the non-exclusive distribution of the journal's published version of the work (e.g., post it to an institutional repository or publish it in a book), with an acknowledgment of its publication in this journal. They are also able to redistribute their code under other licenses at their will, including via commercial contracts and partnerships.
- Authors are permitted and encouraged to post their work online (e.g., in institutional repositories or on their website) prior to, during and after the submission process, with an acknowledgment of its publication in this journal.

With this agreement, and unlike usual publishing terms[10], the journal receives the bare minimum rights necessary to perform its function, *i.e.* to receive, review, validate and publish the research articles. Our goal is to enhance the rights of the authors and readers, in order to ease collaborative research. Similar agreements are already used by some other journals[11], and this shows the viability of this model. We do not know, however, of a single copyright and license policy covering manuscript, software and dataset.

5.2.3 Copyright

Every article distributed by IPOL is the creative work of its authors, and as such covered by the copyright laws. By default, the authors own the exclusive moral rights (the right to claim authorship) and patrimonial rights (rights to a commercial exploitation) attached to their works. Most academic journals require the authors to transfer the patrimonial

[10] Three major publishing organizations in computational science and applied mathematics have similar restrictive policies: the IEEE "*requires that prior to publication all authors or their employers must transfer to the IEEE in writing any copyright they hold for their individual papers*" (`http://www.ieee.org/publications_standards/publications/rights/copyrightmain.html`), the ACM "*requires authors to assign their copyrights to ACM as a condition of publishing the work*" (`http://www.acm.org/publications/policies/copyright_policy#Requirement`), and Elsevier requires the authors to transfer the copyright in "*the manuscript [...] and any supplemental tables, illustrations or other information submitted therewith that are intended for publication*" (`http://www.elsevier.com/copyright`).

[11] *The EURASIP Journal on Image and Video Processing* (`http://jivp.eurasipjournals.com/`) has an Open Access Policy; under the SpringerOpen copyright and license agreement (`http://www.springeropen.com/authors/license`, the authors retain the copyright of their work. Open Research Computation (`http://www.openresearchcomputation.com/`) is a new Open Access journal with a software focus; it requires the software to be available under on Open source License. The Public Library of Science published seven peer-reviewed journals on life and health sciences under a Creative Commons Attribution License (`http://www.plos.org/about/open-access/`). Some data published by BioMed Central journals are released under an Open Data license (`http://www.biomedcentral.com/about/access`).

rights to the publisher. Others, like the current example, let the authors retain their full copyright and only require them to grant a non-exclusive right to publish their works, *i.e.* the manuscript, implementations and datasets.

This agreement secures the right for the journal to distribute the article. The authors keep the right to any other usage of their work: publication in other venues, like other journals and books, with other agreements with other publishers, redistribution of the software with other licenses, negotiation of the commercial exploitation of their inventions, etc. The only restriction is that future usage of their work in other conditions cannot obliterate the distribution rights granted to the journal, which cannot be barred from publishing the article under the agreed licenses. This means that the authors cannot grant an exclusive publishing right for their article or an exclusive usage right for their software or data to another party. They can, however, enter such agreements on future and updated versions, as long as it does not affect the rights granted to the first published for the version published in this journal.

The authors must comply with copyright rules too. Every material to be published with the article, including all the images, text and source code, must either be the work of the article authors or be distributed by their creators under a license compatible with the journal policy. The origin of all these external materials must be explicitly mentioned in the article. This requirement is detailed in the Software Guidelines for the source code, and should be mentioned in the Manuscript Guidelines, yet to be written, for other materials reused in the manuscript.

Finally, this agreement reminds the authors of their right to self-archiving, and encourages this practice. The right to self-archive the preprint version of an article is a matter of journal policy, not of copyright regulation because it happens before any copyright transfer can exist [117]. This is the base of the successful model of preprint repositories widely used with large repositories like arXiv, RePEc or HAL[12]. Our agreement also reminds the authors that, because no copyright transfer occurs and the journal claims no copyright on the value added to the article at the edition and layout step, they are allowed to archive the final version of the article. Contrarily to a preprint, the final version uses the same pagination as the one distributed by the journal and mentions the copyright, the license, and the information needed to properly refer to this article.

5.2.4 License

The proposed agreement is between authors and publishers, and states their rights and mutual obligations involved in the publication. This agreement also mentions the licenses attached to the published works. These licenses establish the rights of a third party: the recipients of the works, the journal readers. Article manuscript, software and dataset have different properties and usage and need different licenses. They have been chosen to allow the research community to reuse and build on the published material and to acknowledge the paternity of the authors on their work in the form of standard academic

[12]In 1994, the "*Realizing the Information Future*" report from the National Academy of Science on the future of the Internet [199] mentions that "*physicist [have] developed and used electronic archives in areas such as high-energy physics [...] as an alternative to buying journals that cost hundred to thousands of dollars per year for subscriptions*". Circa February 2012, the arXiv repository (`http://arxiv.org/`), operated by the Cornell University Library, hosts 750000 preprint papers in mathematics, physics, and biology; RePEc (`http://repec.org/`) has almost 700000 articles in economy; HAL (`http://hal.archives-ouvertes.fr/`), a CNRS initiative, hosts almost 200000 science papers.

citations. Together, they compose the public policy of the journal, summarized by the "Open Access, Open Source, Open Data" trilogy and detailed later in this chapter.

One can also see the licenses as a balancing factor in the author-publisher relation; they both relinquish some of their exclusive rights, for the benefit of the research community:

- The authors allow the publisher to distribute their work if and only if the readers have some right to access and use the published materials. The publisher cannot distribute the works without mentioning the rights granted to the journal readers via the licenses.
- The publisher lets the authors keep the patrimonial rights of their works if and only if the authors allow others to use, modify and redistribute the published works.

Without such licenses, the usual scheme is that the publisher has exclusive rights on the manuscript and the authors maintain their exclusive rights on the software and dataset, while the readers are only allowed to access and read the article and attached material.

Manuscript

The manuscript is published under a "Creative Commons Attribution Noncommercial Share Alike" (CC-BY-NC-SA) license. A summary of this license[13] is:

> *You are free:*
> *- to Share — to copy, distribute and transmit the work*
> *- to Remix — to adapt the work*
> *Under the following conditions:*
> *- Attribution — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).*
> *- Noncommercial — You may not use this work for commercial purposes.*
> *- Share Alike — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.*

Like every Creative Commons license, the CC-BY-NC-SA license maintains the moral rights of the copyright holder but allows the free distribution of the copyrighted works. We expect this to facilitate the work of the research community by disseminating the manuscripts. Reaching more potential readers, we hope to increase the visibility, impact and usefulness of the article, and this should benefit ultimately the authors with more citations[14].

The attribution condition of the license is detailed in the journal by requiring any usage of this work for a scientific publication to cite the article with the standard citation metadata provided with the article. To ensure a stable and reliable access to the cited articles, the journal uses Digital Object Identifiers (DOIs). These identifiers are persistent identifiers

[13]The full text of the license is written and promoted by the Creative Commons organization (`http://creativecommons.org/licenses/by-nc-sa/3.0/`).

[14]The works of Lawence Lessig [217] Richard Stallman [321] also provide a detailed critical analysis of the copyright and patent system at the Internet era.

of materials published online. They help guarantee the validity of a citation and access to the article over time, and they associate an unique identifier to every published article, which simplifies the automated processing of scientific literature databases. With this standardized citation model coupled to the attribution requirements of the license, we follow the practice of the research community and integrate in global bibliometric systems and practices.

The free redistribution model associated with a Creative Commons license implies an unrestricted access to the article: any restriction on accessing the article would be circumvented by a single reader lawfully making the article available in a location not controlled by the publisher. This places the journal in the category of Open Access journals [49], who make their content "*freely available without charge to the users or their institutions*". The articles are immediately and permanently available online and users are "*allowed to read, download, copy, distribute, print, search, or link to the full texts of the articles in this journal without asking prior permission from the publisher or the authors*". This policy is expected to be more adapted to the collaborative nature of the research than expensive and restricted access to the articles as is common with mainstream commercial publishers. Moreover, this should also benefit the authors: some studies showed that articles published in Open Access get more citations from other researchers [93, 165].

Moreover, other researchers are not only allowed to access and redistribute the manuscript. Reusing it to elaborate one's own research is also explicitly allowed. This is the common academic practice, but rarely mentioned in the copyright statements of other journals.

In the competitive academic edition market, one wants to avoid other publishers to benefit from the selection, edition, peer review and proofreading work invested before the first publication of the articles at the expense of the original publisher, for example by reusing the articles in a compilation book. For this reason, the commercial exploitation of the work is forbidden unless explicitly negotiated with the author. We expect that the original publisher will be in the best position to develop commercial re-edition projects with the authors if it is their intention.

Software

If the article includes a software part, typically the implementation of the algorithm presented in the manuscript, the software must be published under one of the Free Software licenses listed in the Software Guidelines. The 1.00 version of these guidelines mentions the "GNU General Public License" (GPL)[15], the "GNU Lesser General Public License" (LGPL)[16], the "Affero General Public License" (AGPL)[17] and the "BSD License" (BSD)[18].

All these licenses are Free Software licenses[19]. They guarantee the same four essential rights to the recipients of the software:

- the right to run the program;

[15] `http://www.gnu.org/licenses/gpl.html`
[16] `http://www.gnu.org/licenses/lgpl.html`
[17] `http://www.gnu.org/licenses/agpl.html`
[18] `http://www.opensource.org/licenses/bsd-license.php`
[19] Similar definitions of Free Software are given by the Free Software Foundation [131], the Open Source Initiative [184] and the Debian project [291].

- the right to study how the program works;
- the right to redistribute copies of the program;
- the right to modify the program, and redistribute the modified versions.

These licenses differ in the additional rights the authors want to guarantee to the users of the program and its modified versions and how it can be combined with non-free software. They are adapted to different kinds of software and different strategies of their authors, as shown in table 5.1.

Many other Free Software licenses exist, but the short list in the Software Guidelines aims at simplifying the choice of a license and reducing the complexity of the license management in a journal, while being adapted to all major use cases. To be complete, the list should also include a very permissive license such as the "GNU All-Permissive License" (APL)[20], suitable for trivial works (the similar Public Domain terms are not recognized in European jurisdictions).

	GPL	LGPL	AGPL	APL	BSD
four essential rights for recipients of the code	•	•	•	•	•
authorship must be acknowledged in source distribution	•	•	•	•	•
authorship must be acknowledged in binary distribution	•	•	•		•
modified versions must be released under the same license	•	•	•		
source must be distributed with compiled versions	•	•	•		
must only be linked with codes under a compatible license	•		•		
source must be provided with application services using it			•		

Table 5.1: Brief comparison of five software licenses: GPL, LGPL, AGPL, APL, BSD.

We chose this free software license policy because we consider it promotes collaborative research. Free software is the programming equivalent of the publication of research papers[21]. Like the article manuscript, the source code is expected to be read, used and tested by other researchers. They will build their own research publications on the results exposed in the article, and they will build their software tools by reusing parts of the source code published in the article. And the license terms require the copyright attribution to be conserved in software works derived from the published code, to acknowledge the work's authorship. This is similar to the requirement for a correct citation of the manuscript in academic journals, but adapted to the habits of software development and collaboration.

Unlike the manuscript, however, we do not consider that a restriction to noncommercial use is adapted to research software because commercial and industrial applications will want to reuse the ideas, inventions and algorithm, not the experimental implementation published in the journal. The copyright and software licenses only cover the source code, not the algorithm expressed in the source code. Instead, inventions are covered by trade secrets and patents. As the current patent regulation excludes the mathematics and scientific discoveries from patentability, it seems that an algorithm could only be "protected" by secrecy, which is not applicable when the inventors choose to publish their algorithm in a scientific journal. For these reasons, there is no justification to restricting the software part of an article to non-commercial use only[22].

[20] `http://www.gnu.org/licenses/license-list.html#GNUAllPermissive`

[21] This affirmation is taken from a chapter about "software for rent" and the benefits and free software in "*Confessions d'un voleur — Internet : la liberté confisquée*" [79].

[22] In fact, some patent jurisdictions, principally the United States one, recognize some patents on inven-

Finally, it is worth repeating that the free software license used to distribute a source code via the journal does not prevent the authors to use other licenses when they redistribute their works, or enter commercial agreements to sell a right to use their software or future versions to a third party. This will especially be useful when an industrial partner does not want to be bound to the GPL license terms. These terms require anyone reusing a GPL licensed software in their own program to release the whole source code under GPL-compatible license terms. Most industrial partners would prefer to negotiate a commercial arrangement with the authors for a non-GPL license to reuse their code.

Dataset

If the article includes a dataset part, this dataset must be published under a "Creative Commons Attribution" (CC-BY) license. A summary of this license[23] is:

> *You are free:*
> *- to Share — to copy, distribute and transmit the work*
> *- to Remix — to adapt the work*
> *Under the following conditions:*
> *- Attribution — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).*

Like every Creative Commons license, the CC-BY license maintains the moral rights of the copyright holder. Like for the manuscript part of the article, the attribution condition of the license is detailed in the journal by requiring any usage of this work for a scientific publication to cite the article with the standard citation metadata provided with the article, including the Digital Object Identifier reference.

The datasets are freely available on the Internet permitting any user to download, copy, redistribute, analyze, process, or use them for any other purpose without financial, legal, or technical barriers. We think that a very permissive license like this one is adapted to datasets because such works gain value by being reused and integrated in comparisons and benchmarks. Without a free access to the data, one cannot reproduce the experiences and compare the published algorithm with another one. This license policy is compatible with the Open Data definition [132]: "Anyone is free to use, reuse, and redistribute it — subject only, at most, to the requirement to attribute and share-alike.␣"

The only condition is to mention the authors of the work and refer to its original publication as the non-ambiguous location where this dataset is available. A reliable and unique location is especially important for datasets. In its absence, one may refer to different data by the same name and variations of the same initial data would result in biased measures. This happens, for example, with the well-known "Lenna" image, available from different places on the Internet with different resolutions and alterations and no clear way to know which is *the* standard Lenna picture [342].

---

tions involving the use of a program, and these patents cover the algorithm implemented by the program as a part of the wider invention. To accommodate for this situation, we require a patent warning to be included in the source code when the algorithm is covered by such patents, and we accept *ad-hoc* "research only" license terms when the source code author is the patent inventor.

[23] The full text of the license is written and promoted by the Creative Commons organization (`http://creativecommons.org/licenses/by/3.0/`).

Reliable indexation and storage of such scientific datasets is necessary. Research cannot rely on lab websites for the availability and identification of important datasets. Among the few initiatives to tackle this important issue, ce can cite 3TU.Datacentrum [16], a joint project by the universities of Delft, Eindhoven and Twente, and DANS [92], an institute of the Royal Netherlands Academy of Arts and Sciences and the Dutch Organization for Scientific Research. More efforts are needed, especially out of the biomedical firelds, probably to be led by research agencies at the national or international levels.

Finally, we must say that a dataset, even published under a permissive license, is useless if the data format is not documented or if the software tools needed to read and use the data are not available. To publish useful datasets, one must use open and standardized formats. This should be covered by appropriate Dataset Guidelines, yet to be written.

5.3 Online Publishing and the Law

5.3.1 Online Demos in Research Journals

Online research demos provide a web interface to process user-submitted data with a research programs, in order to try and compare different algorithms. They are developed and proposed by individual researchers, research groups or laboratories, or journals.

One major difference between such demos and articles, software and datasets is in their nature: online demos are a service, not a product. They may be reviewed like articles, but instead of being edited, published and distributed, demos are maintained and continuously kept available online. Moreover, if these demos include a public archive of the experiments, their online content includes the data submitted by the users, not by authors or editors.

This difference in nature has legal consequences. A demo is another program, with a web user interface and networked input and output, and as such the creative code expressed in its code, like any program code, is copyrighted. But the copyright does not applies to the service provided with this code. When visitors use a demo, they do not receive the demo, they only interact with it. The demo program stays out of their reach, on the demo server.

Instead of the copyright regulation, one may attempt to invoke patent protection on the demo service. But such patents on web service are only recognized in few jurisdictions, and the idea of an interaction with distant computers via a web service is at least as old as the Common Gateway Interface (CGI) introduction in 1993.

On the other side, the data processed by a demo has an author, a copyright protection, and its availability in the demo archive is a redistribution of this material. But the author and copyright status of most of this data is unknown to the demo administrators. We may also wonder who is responsible for the use and possible abuse of this archive.

The answer is to be found in the law. After an overview of the legal coverage of publishing and communications, we will detail how an online journal, including online demos with archives, can be operated and used. This analysis only covers French law because of the location of our IPOL project. The situation would be different in other countries with other local laws on press regulation, copyright protection and free speech.

5.3.2 Retrospective on French Law and Internet

Press and Freedom

Introduced in 1881, the *Law on the freedom of the press* [1] is the foundation of the freedom of the press and freedom of expression in France and defines the regulation of publications and media. This law went through several amendments but remains in force today and one century later, its framework governs electronic publishing on the Internet.

For every publication, the publisher ("*directeur de publication*") and printer ("*imprimeur*") must be publicly identified. This obligation has been maintained and adapted for electronic publications in recent laws. We detail later their respective responsibilities.

The publisher is responsible for all the published material and may be condemned for libel, insult and defamation. The 1881 law also introduces heavy punishments for inducing discrimination, hate or violence against races, nations and religions. These "press offenses" are still enforced today on Internet publications, from the web version of paper newspapers to personal blogs.

Computers and Data

In 1978, before the advent of personal computers and while networking was still in a pre-Internet state, the *Law on the computers, databases and liberties* [2] is the first regulation of the possible privacy issues foreseen in automatic data processing. This law creates an independent administrative authority ("*Commission Nationale Informatique et Libertés*" — CNIL), mandated to monitor collection, storage and use of personal data. Since 1978, any collection of personal data has to be declared to the CNIL with details on what is collected and how it is processed, and users have the right to access and request modification of their information. This obligation naturally extended to the Internet; it applies for example to the user accounts or web usage statistics and tracking implemented in a website.

Electronic Communications and Crime

In 1982 with the *Law on the audiovisual communication* [3] and in 1986 with the *Law on the communication freedom* [4], the laws on press are transposed to the context of electronic communication. With the radio and television regulation, the Internet is introduced in the French law with other "means of communications not using hertzian waves". These laws state that electronic communication is free and its usage by the citizen a privacy matter. They also require a publisher to be identified for every electronic communication service, a parallel with the press laws. The equivalent of the printer role, however, will only be defined 10 years later.

In 1988, the *Law on the computer fraud* [5] adds to these regulations of electronic communications as mass media the first legal definition of computer crimes and punishes the unauthorized access to an automated data processing system — generic definition of a computer — and the deletion or alteration of data on this system. This is completed in 2000 with the recognition of electronic documents as valid legal material and acceptable evidence, at par with paper documents, in the *Law on the evidences, information technologies and electronic signature* [6].

Trust and Security

With the 21st century, Internet becomes a major communication medium. In 2004, the *Law on the trust in digital economy* (LCEN law, from the French acronym) [10] details the regulations on Internet communications that were missing from previous early texts.

It defines the roles and responsibilities of the publisher ("*éditeur*") and hosting provider ("*hébergeur*") of an electronic publication, inherited from the paper press world. Publishers have editorial control of the published content; they order, select, correct and approve every piece of information distributed via their media. As such, they are responsible of this content and its compliance with the laws. On the other side, the hosting providers are only technical service operators; they have no *a priori* knowledge of the information they make available online. They are not responsible and do not have to monitor it, but must promptly take it offline when notified of an illegal content[24].

The law recognizes that every website does not necessarily have a clearly defined editor. Most of the web content is published by their author rather than as a publication. Some websites are also open to free contributions, like wiki sites, forums, and blog comments. All these situations form the majority of the Internet publication. This ability to broadcast one's message to the world was the novelty of the Web after centuries of printed mass communication. For all these situations the hosting providers, which can usually be identified by technical means, are required to keep a record sufficient to identify the authors of every content they store and distribute via the networks[25].

During the same decade, the legislators adopt five laws including electronic communications in texts focusing on security issues: the *Law on the daily security* [7] and *Framework law on the internal security* [8] in 2003, the *Law on the internal security* [9] in 2003, the *Law on the terrorism* [12] in 2006, and finally the *Law on the performance of internal security*" [15] in 2011. These texts detail how identification databases, wiretaps and filtering can be used by the police and justice administration. They also refine the definitions of the responsibilities involved in electronic communications, which identification data has to be kept, how long, and who can access this information.

Copyright

At the end of the decade, the latest set of laws so far deal with copyright regulation and file sharing, in 2006 with the *Law on copyright in the information society* [11] and 2009 with the *Law on creation and Internet* [13] completed by the *Law on the copyright protection and Internet* [14]. These texts punish the unauthorized publication of copyrighted works, with an exception when this is solely done for education and research without commercial

[24] "*Les personnes physiques ou morales qui assurent [...] le stockage de signaux, d'écrits, d'images, de sons ou de messages de toute nature fournis par des destinataires de ces services ne peuvent pas voir leur responsabilité civile engagée du fait des activités ou des informations stockées à la demande d'un destinataire de ces services si elles n'avaient pas effectivement connaissance de leur caractère illicite ou de faits et circonstances faisant apparaître ce caractère ou si, dès le moment où elles en ont eu cette connaissance, elles ont agi promptement pour retirer ces données ou en rendre l'accès impossible. [Elles] ne sont pas soumises à une obligation générale de surveiller les informations qu'elles transmettent ou stockent, ni à une obligation générale de rechercher des faits ou des circonstances révélant des activités illicites.*" — LCEN Art. 6

[25] "*Les [hébergeurs] détiennent et conservent les données de nature à permettre l'identification de quiconque a contribué à la création du contenu ou de l'un des contenus des services dont [ils] sont prestataires.*" — LCEN Art. 6

use. They also introduce various measures to monitor network communications, detect copyright infringement and suspend the offenders' Internet connection.

5.3.3 Consequences for Online Journals

From this timeline, we can identify three categories of legal obligations on electronic publishing:

- the generic regulation applying to anyone managing a website: identify the people involved in the service, respect the privacy of the users
- regulations on web publishers and authors: do not put illegal material online
- regulations on web hosting providers: identify the authors, remove illegal content on notice

We will see hereafter how this translates for an online research journal with demos and archives.

General Website Operation

Like any website, an online research journal is required to identify its publisher and hosting provider. For self-published independent journals, the publisher will be the person who chooses, selects or writes the content included in the journal or oversees this process, *ie* the editor in chief. The publisher is also usually identified when the journal requests an ISSN registration.

The definition of the hosting provider is more subtle, because this function may be decomposed into different layers: who administers the server, who owns the machine and who controls its connection to the Internet. If these roles are handled by different entities, the simple solution is to mention them all.

Finally, users must be informed of the data collected from their usage of the service. Even if there is no user account and no user is identified by their name, a simple web usage statistics tool collects personal information: IP addresses of the visitors, with the time and date of every page they accessed. This usage of an online communication service is a private matter, so such usage monitoring must be declared to the CNIL, mentioned on the website and the visitors must have the possibility to access and correct this information.

Edited Material: Manuscripts, Software and Datasets

The journal editors propose, select, review and validate every article to appear in the journal issues. This is unchanged from the traditional press and publisher model to the electronic medium transition. The publisher — usually the editor in chief — has the legal responsibility to ensure that every published material is legal.

Due to the nature of a research journal, few articles, if any, present the risk to include unlawful content such as libel, defamation and hateful messages. The major concern is copyright infringement, in any of its forms: abusive copy/paste of external sources and other articles, plagiarism, disregarding the copyright status and restrictions of data

included in an article, such as images and figures, of redistributing a software without being allowed to do so or without crediting the original authors.

But the editor in chief cannot perform a complete inquiry for every article to be published. Instead, in the IPOL journal, the rules are simply mentioned to the authors via a copyright agreement and some author guidelines. Researchers contributing to the journal are asked to mention the origin and respect the authorship of every non-original content included in their submission. These agreement and guidelines are acknowledged by the authors when they propose a new article, and unless a misconduct is signaled the editor relies on their integrity.

Contributed Material: Demo Archives

The public archives of an online demo are another matter. First, the mere volume of such a service (more than 50 new images per day for a small journal like IPOL) means that monitoring the content of the archives is no small effort. Then, if the archives are moderated, every image publicly available in the archives has been approved by a moderator, and the publisher publisher is liable for every possible illegal content, any identifiable face, any recycled image, and so on.

Another possibility, chosen for IPOL, is to automatically publish this content without *a priori* filtering. If the archives are not moderated, the journal is only hosting content contributed by the demo users. The publisher does not condone this content and the journal only provides a technical mean to put it online, and acts as a mere hosting provider. This approach is based on the similar situation of non-moderated web forums, whose administrators have been qualified as hosting providers, not editors, by case law from 2002 [71, 96, 115, 256].

In order to support this qualification of simple host, the online demo users and visitors must be clearly informed of the non-moderation of the archives, and all the information necessary to notify illegal or suspicious content must be available in the archive. Moreover, the origin of every material inserted in the demo and redistributed via the public archives must be recorded and kept for one year, as required by the LCEN decree[26]. This is easily achieved via the IP address and file identifiers found in the default web server logs.

One may wonder if such online archives would not be flooded by illegal material. After two years operating the IPOL services, our answer is no. There has been some abuses, but very few: about one per month, less than 0.1% of the archive volume. And these abuses were always with questionable and obviously not scientific content such as nudity pictures, but never with obviously illegal material. The explanation may be that such archives are not very convenient: inserting the images in the system takes some time to go through the processing chain, and their size is limited. There probably are lots of more efficient channels to distribute unlawful content.

Acceptable Input The archives are not moderated, but the users are informed of which input data they should not feed in the system, to avoid copyright infringement, privacy violation and press offenses.

[26] LCEN application decree, 24 February 2011 (`http://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000023646013`)

Copyright The copyright status of the images processed by the online service and publicly archived must not conflict with this usage; the copyright owners must allow this processing and the redistribution of their works via the archive. To avoid copyright infringement, online users must only upload images for which no copyright applies (public domain), or the copyright is theirs (personal photos), or when the copyright owner explicitly allows derivative works and redistribution for free without any restriction.

This implies that images "found on the Internet" cannot be inserted in the public archive of a demo, because unless stated differently all the rights, including the right to redistribute the works and to modify them, are reserved to the authors. But copyleft image databases provide a useful and rich alternative. Via the Creative Commons search portal[27], one can search for images, video or sounds in the Flickr, Google, Jamendo, Wikimedia, Youtube or Europeana databases and restrict the search results to materials we can reuse and redistribute.

Privacy Photos redistributed via the demo archives should not infringe on the right to self image and privacy. No person should be recognizable on these photos unless they have given their consent for publishing. Precisely, online users must not upload images of human beings if the persons displayed in the images did not explicitly give their written consent or are not mentally healthy (and not able to give an informed consent); for children, the parents must have given this consent.

To avoid unnecessary paperwork, we suggest the online demo users to use photos of themselves if faces are useful in the processed images. For photographs of persons included in published journal articles, an authorization letter is required and kept at the office of the editor.

Other Illegal Content All other generic press offenses and illegal material per free speech regulation are obviously forbidden too. This includes encouraging hate, discrimination and racism, child pornography, attack against human dignity, counterfeiting, apology of crimes against humanity, apology and incitement to terrorism and negationism.

Output Usage Conditions Finally, no copyright policy can be established on the output of the online demos because the copyright status of the input images is unknown to the demo administrators. In particular, for their own articles, researchers should only reuse the output obtained after processing data they submitted themselves to the algorithms and for which they know the origin and the modification and distribution terms. They are also asked to cite the article linked to the demo if they use this demo in their works, but this can only be a courtesy matter, not a license condition.

[27] `http://search.creativecommons.org/`

Chapter 6

A Short Survey of Image Processing and Computer Vision

Contents

Abstract

This chapter starts with a review of image processing and computer vision terminology extracted from well frequented web sources. The goal of this review is to make sure that few aspects of the subject as it exists escape the discussion. The ultimate goal is to be able to sketch a scientific and algorithmic program adapted to IPOL, this program being thought in terms of algorithms, or of algorithmic questions. The chapter continues with the proposition to deduce a sizeable part of the necessary algorithms from the imaging system itself. Cameras and the other imaging systems have a number of fixed mathematical and physical characteristics that literally define the image structure and its potential defects. Other algorithms are rendered necessary by the structure of the visual world, and by the structure of our perception. The chapter starts by establishing lists of algorithms that are required for any image processing or any image analysis project. Although it would be ridiculous to pretend that such lists could be complete, there is some return on investment for this effort. For example, the list of envisaged algorithms, while being rather detailed, does not exceed a couple of hundreds. It includes of course many of the algorithms that have been already acclaimed, but also several unnoticed ones, despite their obvious importance. The moderate number of necessary algorithms raises the hope that a journal like IPOL can reach soon a usable size, where a majority of the essential algorithms or questions will be represented by at least one first article. This would enable IPOL to establish the basis for a new sort of algorithm dialog, where each new algorithm could be confronted to the others by its results on any image, and by its detailed description and comparison to others, in the spirit of Hermann Hesse's *Glassperlenspiel* [173].

6.1 The Universality of Image Processing

The existence of a fully autonomous image and movie processing chain is a biological fact. Our visual system is completely adaptable and robust to the physical changes of the environment and of the optical system itself. It is able to synthesize a better quality view from several ones. The binocular fusion of two retina images into a *cyclopean vision*[1] not only recreates depth, but also significantly enhances the visual acuity of each eye.

Since imaging is invading all crucial aspects of our all-day life, and in particular of our *scientific life*, it is crucial for image science to deliver universal image analysis and processing algorithms to the immense group of users.

Furthermore, most researchers, engineers, doctors, and other users have no easy access to the internal parameters of their own imaging tools. In consequence, image processing labs are overwhelmed by a growing demand from biologists, doctors, physicists or physicians who request them to make the best of their observations, be it for noise, blur, color, contrast, stability, image comparison, image registration, and so on.

The creation of universal image and video processing algorithms is so much more important that, *contrarily to intuition, most high level image analysis tasks essentially rely on low level automatic mathematical primitives.* This fact was discovered and illustrated in the past century by the Gestalt school, from the founding paper by Wertheimer [366] to the books of Metzger [246] and Kanizsa [192] [191]. Metzger's book summarized in 1975 fifty years of Gestalt theory research demonstrating that the visual perception of 3D information, color, texture, shadows, geometric structure, perspective, motion, and causality *is a low level process*, automatic, universal and definable in geometric terms analogous to physical principles.

[1] Term due to the psychophysicist Bela Julesz

However, every image process is not a single algorithm, but rather a complex chain of atomic algorithms, each one requiring the presence of the others to make the result visible. A crucial question for an image processing journal is whether universal image processing atomic algorithms can be made available to all. They must generally be combined with others to make them testable on line. Thus, the difficulty is that such algorithmic chains should integrate very diverse interlaced tools, each requiring a specific mathematical theory and its algorithms. So far the knowledge in image processing, when existing, is partial. Many case-dependent competing algorithms are proposed. The main image processing and analysis journals (IEEE Image Processing, International Journal of Computer Vision, Journal of Mathematical Imaging and Vision,...) reflect this babel situation by publishing large numbers of papers on each given problem without disposing of a proper comparison methodology.

The other main difficulty is that in the literature most algorithms depend on several supervised parameters. Eliminating these parameters is a crucial requirement to have algorithms usable by all, and demonstrable on line. The main difficulty is probably the variety of mathematical and computational theories that have to be adapted or developed for each one of the algorithms. *The fact that our vision integrates smoothly all aspects of image formation gives a sense of easiness that is fully illusory.* To give a few examples, sampling theory relies on harmonic analysis with a subtle interlacing of discrete and continuous aspects. Color theory relies on PDE's. Denoising relies on nonlocal mathematical processes of a new form, unknown until they were introduced for that scope. Image matching relies on a multiscale theory, scale space, that is also specific of image analysis and computes scale invariants in a very clever way.

In this chapter we shall try to recast the typical image formation chain and therefore to deduce the typical image algorithmic atoms and chains, as they should be elaborated in IPOL. We shall start with an attempt of synopsis of the whole field, obtained by concatenating in the right order the 2000 most frequent key words or expressions of computer vision and image processing. This synthesis in chapter 6.2 will show the overwhelming variety of the goals of Computer Vision and Image Processing. Nevertheless, we shall discuss in chapter 6.3 whether there is an underlying general model for this material, and how a systematic scientific and reproducible methodology might cover it. We shall see that, in fact, most questions of Image Processing and many problems of Computer Vision stem directly from the image formation model. This will allow us to end with a development program for the development of the discipline, and therefore orientations for the IPOL journal, which must expand on the subjects which seem to be crucial.

Of course, this synthetic attempt permits a broad coverage, but does not bring in the necessary analysis. Thus in a second section we shall try to deduce as much as possible of the Computer Vision and Imaqge Processing program from the mere model of cameras. This deduction works well for image processing and the parts of computer vision which are close to image processing, in particular all questions related to the 3D reconstruction of the environment.

As for the question of Computer Vision that are related to our human world and our human quests in it (mostly related to object recognition and scene understanding), the research program is more focused on Machine Learning than on vision itself, so that a rational deduction of the main topics would request a thorough discussion of the Machine learning program. This goes beyond our scope. Furthermore, we see so far no way to actually execute efficiently online learning algorithms, which request the upload of very large databases and a sophisticated benchmark method. Thus only *ad hoc* computer vision

tasks will be discussed and the main learning algorithms will nevertheless be mentioned.

Needless to be said, research creates constantly unexpected new fields of investigation. Thus, the present discussion only reflects an ephemeral view on the state of the art of the discipline.

6.2 A Rewriting of 2000 Keywords

Computer Vision (CV) and Image Processing (IP) are two twin disciplines (CVIP) dealing with the processing and analysis of digital images. They interact directly with Imaging, Neurobiology, Psychophysics, Robotics, and Photogrammetry. Indeed, they handle the digital part (algorithms and data) of these five disciplines. This section presents a synthesis of the list of 2000 most frequent key words in Computer Vision, taken from The Microsoft Academic Search database[2]. This recent list is completely unstructured and sometimes redundant. In the following text, we commit ourselves to eliminate only the redundant or synonym terms and to organize *all* of the other ones. The goal is to get a faithful and complete coverage of the discipline in its current state. It seemed to us that the 2000 key words could be at first classified in three "robust" categories:

End-goals those dealing with the end goals or applications of CVIP, such as image compression, video surveillance, image retrieval, urban modeling;

Concepts of CVIP those which represent a specific conceptualization, of a more philosophical nature. They characterize the lines of thoughts and emerging theories: for example learning, multiscale analysis, pattern formation, visual grouping, or image representation;

Mathematical tools and algorithms: the most numerous terms depict the mathematical theories and techniques used (and changed) by CVIP. They stem from projective geometry, topology, Fourier analysis, parametric and nonparametric statistics, probability, optimization, partial differential equations and variational methods.

6.2.1 Main goals of Computer Vision and Image Processing

According to the exhaustive list of 2000 key words which we shall simply reorder in the ensuing text, the main goals of Image Processing and Computer Vision can be classified in not less than sixteen categories. Because they are not our main focus, the goals related to the conception of imaging systems will be listed at the end. For the rest, the logic leads us to go progressively from "low level" to "high level", namely from image formation, to the "medium level" extraction of cues and reconstruction of a 3D environment, and finally to the high level image and video interpretation tasks. We give now the list of the CVIP goals:

Compression to encode and compress by lossy or lossless coding or sparse representation images, video, and 3D point set data;

[2] `http://academic.research.microsoft.com/`

Single image processing to define, control and improve the quality of digital images and video by developing mathematical theories and algorithms for raw image sampling, demosaicking, interpolation, denoising, deblurring, super-resolution, color balance, contrast enhancement, histogram manipulations, motion compensation, motion deblurring;

Multiimage processing to perform the same operations when several images of the same scene are available, in which case the correspondence problem, also called image matching, or image alignment, or image registration, or non rigid registration must be solved first, followed by a joint demosaicking, denoising and super-resolution strategies, a joint histogram equalization, followed by the same color balance or contrast enhancement. This joint restoration and equalization can end up in the fusion of the images, for a higher resolution image, an image mosaic, or a panoramic image;

Video processing to perform the same operations as above from video, in which case the *motion compensation* and optical flow techniques replace the registration;

Camera calibration and pose to deduce a complete mathematical model for each camera from photographs taken by the camera of specific patterns, or of multiple solid scenes: this permits to give accurate models for the charged coupled device (CCD), for the point spread function (modeling camera blur), the lens optical distortion, the chromatic aberration and the vignetting effects. Likewise from multiple views or video the whole camera geometric model including its focal length and pose or motion can also be estimated. This calibration can be extended to camera networks surveying a scene. The precision of these measurements is a key to digital photogrammetry;

Illumination to model, detect or compensate the effects of light on the photographed scenes such as shadows, reflectance, highlights, specular reflections, structured light. In particular certain algorithms tend to minimize the effects of changing light on images by manipulating the color histograms, and applying color constancy strategies;

Image and video low level analysis to extract cues, otherwise called features which are universal and can be a common denominator to any image and video processing and analysis: color and texture segmentation, feature grouping, extraction of interest points or key points, shape descriptors, detection of the main aspect graph features (triple junctions, corners, edges), line detection, saliency maps, Harris points, similarity invariant features, bags of visual words. In video, this feature set is complemented by motion estimation, motion segmentation and feature tracking;

3D recovery: to recover the 3D shape of objects from one, from two, from several images, or from a video stream. Already with one image partial depth information can be recovered by shape from shading, shape from silhouette, shape from texture, depth from focus, geometric perspective (vanishing points), atmospheric perspective and local occlusion cues such as T-junctions. Recovering the 3D depth map from two or more images is the classic *stereo vision* problem, and recovering 3D from video is made by the structure from motion method. Nevertheless, the creation of 3D scenes from images or video brings on the 3D point sets similar problems as for images: first to triangulate or mesh the 3D point cloud, to analyze it, segment it, extract its meaningful features, recognize and align several meshes, etc. The urban scenes and digital elevation models are sometimes also obtained directly from 3D imaging system like range images or sometimes by remote sensing (radar or optical aerial and satellite images);

Object modeling, detecting, tracking, and recognition: to model, detect, track, align, and recognize in images or video any kind of solid or moving or deformable object, or event, or action, and to determine their pose. This recognition problem applies to the images and videos themselves as object to be recognized. But also to shapes, patterns, and more specifically, for example in video surveillance and remote sensing, to foreground and background, buildings, roads, communication networks, targets, vehicles, human motion, human activity, human actions, events. Also to people, characters, faces, heads, facial emotions, facial expressions, facial features (lips, skin colors, eyes, eye directions, eye movements), gait, gestures, hands motions. In images of printed material the question is to recognize and read text, handwriting, logos, line drawings. In industrial material, to detect and classify cracks, defaults, anomalies, fibers, surface roughness, etc.

Scene understanding and control still more generally, to understand and classify filmed scenes, with such goals as quality control by automated visual inspection, video surveillance of building or public places, road traffic control or intelligent video conferencing;

Content based data organization and mining to organize databases of images and video by their visual content information, thus permitting to index them, to annotate them, to summarize them, to search them by content (data mining) basing this search on a query. In particular, to make these operations on the web, organizing global photo or video sharing;

Man machine interaction to devise new camera based man-machine interactive environment using eye and gaze tracking, gestural interface, pose estimation;

Medical diagnosis and action to explore the human body and perform image based medical actions such as the exploration of anatomical variation, or the brain mapping. To perform computer aided diagnosis of (*e.g.*) breast cancer by mammography, of cardiovascular diseases by vessel segmentation. To make surgical planning and perform image guided surgery. To explore all layers of life by Biomedical Imaging from proteins, cells to organs. To obtain cell genealogies from videos of embryos;

Computer Graphics to create artificial images and video and virtual environment for electronic games and the movie industry and to develop editing and augmented reality techniques for images and video, including for example texture synthesis and mapping, ray tracing, inpainting, view interpolation, and 3D animation of artificial scenes and humans. These operations represent a complete fusion of image processing, computer vision and computer graphics techniques. They aim at a faithful scene reconstruction that recreates new images and video from acquired images and video in which artificial objects can be inserted.

Robotics to conceive vision tools for intelligent autonomous vehicles or robots endowed with navigation, path planning, obstacle detection and avoidance, detection and tracking capabilities;

Imaging systems last but not least, to conceive imaging systems specially conceived to model, detect, extract, observe, track or count certain objects of interest. Thus, there is a loop between imaging systems and the image processing and analysis tools. The list of imaging techniques is constantly expanding. The list here is far from exhaustive. Optical cameras based on the pinhole principle can be panoramic, fish eye, thermal, infrared, multi-spectral, push-broom (in scanners or satellites). They can have high dynamic range or an augmented light system, and can have any size, up to telescopes. Active imaging systems emit and catch reflected or absorbed waves:

structured light scanner, laser range scanner, laser triangulation scanner, synthetic aperture or laser radar, sonar, ultrasound. To this we must the add the whole series of X-ray based imagery, X-ray tomography, electrical impedance tomography, emission tomography, angular resolution diffusion imaging, positron emission tomography (PET) magnetic resonance imaging (MRI), functional MRI, diffusion tensor MRI, scanning electron microscopes, single photon emission tomography, etc.

6.2.2 The Concepts of Image Processing and Computer Vision

IP and CV use relatively traditional applied mathematics : probability, statistics, topology, linear transforms, variational methods and partial differential equations. Nonetheless, they have introduced a series of new concepts which perfuse the whole discipline. Several of these concepts refer to a global apprehension of the end goals, according to which computer vision aims at building up vision systems, ending with shape, image and scene perceptions and representations. Certain authors such as Grenander [156] or Mumford and Desolneux [258] go as far as to consider that to recognize patterns, shapes, objects, scenes, one must be able to simulate them. Thus they envision a fusion of pattern classification and pattern formation theories.

Is CVIP ill-posed or well-posed? There is a rift between those who envision computer vision as an ill posed problem, and those for which it can become well posed by collecting enough information from redundant data, obtained for example by an active vision strategy.

Nevertheless there are uncertainty principles due to the necessary presence of noise and blur. *A priori* performance bounds for every vision task depending on the signal to noise ratio are unavoidable. Notorious examples of an uncertainty requiring some sort of *a priori* model are the ambiguity of apparent motion in video, also called "aperture problem", and the illusory contours discovered by the Gestalt school, which demonstrate the necessity of top down strategies for detecting certain low level features. Likewise, the segmentation problem is generally formulated as a variational problem with a regularity term signaling its ill-posed character. Therefore there are recent tendencies to reformulate segmentation as a man-computer interactive process.

For those who consider computer vision an ill-posed problem, the curse of dimensionality is often invoked. It is the difficulty of learning high dimensional probability densities from very few samples. The missing information is recovered by supervised Machine Learning strategies, using as much *a priori* knowledge as possible, complemented by a ground truth. The *a priori knowledge* can also be endowed in a parametric or neural model of the objects, like in the Bayesian models. *A priori* models can also be obtained from natural image statistics of indoor and outdoor scenes.

Multiscale representation There is a general agreement since the founding Marr book [238] that the visual representation resulting from an image or video analysis must be hierarchical, from local features to higher level groups. This hierarchy is built by a multi-scale analysis that can be linear (steerable filters, wavelets, Gabor functions), non-linear local (PDE's, scale space), or non local like in Gestalt theory, which sustains the existence of perceptual nonlocal feature grouping mechanisms. The gestalt perceptual grouping, when emulated in computer vision, is based on similar color, similar texture,

similar disparity, similar motion, parallelism, similar shape, and more generally on what gestaltists call "common destiny law" (*Gesetz des gemeinsamen Schicksals*). The gestalt grouping principle gives the framework for bottom up strategies starting from local cues up to perceptual groups, complemented by top down inference. Technically, as we shall see, it boils down to a clustering in the "feature space". Some researchers have also attempted to formalize hierarchical grouping as some sort of shape grammar. The simplest and earliest such grouping strategy seems to the be the Hough transform, grouping aligned points. The SIFT transform (scale invariant feature transform) has been the first method to give an achieved and efficient scale invariant representation. It computes features that are local, translation, rotation and scale invariant, and insensitive to illumination changes.

6.2.3 Mathematical Tools of CVIP

It is more and more apparent that computer vision and image processing are leading to a thorough reformulation and many innovations in all fields of applied mathematics.

Linear transforms Linear transforms, Hilbert bases, in particular Fourier bases, wavelet bases, Gabor frames, the cosine transform, steerable filters and their fast versions (DCT, FFT, DWT) have received strong backing from neurophysiology since the Hubel-Wiesel discovery of retino-topic orientation and scale selective neurons in the V1 area of cats [178]. These transforms are used to compress images and video, and to build redundant multi-scale representations for textures and images, inspired from the V1 architecture. The Wiener filter, attenuating transform coefficients caused by noise, has been adapted successfully to DCT and various wavelet transforms for image denoising and deblurring.

Projective geometry Projective geometry has been completely renewed by the multi-view geometry [123], [168], namely the interaction of multiple cameras and therefore of multiple perspective structures, with the introduction of geometric concepts such as the fundamental and essential matrix, the epipolar geometry, the extensive use of homographies to model camera rotations, the importance of detecting vanishing points in images, and the relevance of affine invariance to model projective local deformation of flat surface patches.

Information theory Shannon's entropy [316] and the minimum description length model [296] are the main attempts to measure the amount of information contained in an image. As a consequence of Shannon's theory, mutual information, relative entropy and the Kullback-Leibler distance have been used as natural measurements of how much difference there is between two images. Likewise, the maximum entropy principle is used to select the best solution in variational formulations.

Topology Being functions, images have a natural topology given by their topographic map defined as the inclusion tree set of level lines [252], [29], (or alternatively the tree of upper and lower level sets). Other neighborhoods systems in the image can be obtained by multi-scale segmentations, often obtained by region growing, region merging [205], or the watershed method in mathematical morphology [141, 357]. Other neighborhood systems turn out useful, like those given by making a Delaunay triangulation of detected key points.

Probability Probabilistic inference is inherent to the stochastic nature of most image or texture models (Markov random fields, hidden Markov models for patterns). Noise and texture models are often Gaussian random fields or Markov Random fields [140].

Using Grenander's principle, they can be both estimated and simulated (sampled). Given a data and a Markov chain model, possibly depending on hidden variables, the principle is to estimate the parameters by expectation maximization, or mean field techniques. The maximization can be accelerated by Markov chain Monte Carlo techniques, Gibbs sampling, simulated annealing, particle filters, or belief propagation [260].

Variational models Because of the ill-posed assumption of the computer vision reconstruction problems, variational formulations are frequent and use a Tikhonov regularization. They minimize an energy combining a fidelity term replacing the ill posed equation and a smoothing term controlling the uncertainty by imposing some smoothness principle. This is for example the case for the following energies: the bundle adjustment in multiple view stereo vision [344], the block matching and the Potts model for disparity estimation in binocular vision [57], geodesic active contours or snakes in boundary detection [75], the deformable templates, for patterns, shape or surface matching, the thin plate splines or radial basis functions for surface interpolation and regularization, the Mumford-Shah functional in image segmentation [259], the optical flow regularizing terms to resolve the aperture problem in video analysis [177], the total variation of the image in the deblurring and denoising problems [305], the earth mover distance for several matching problems, the geodesic distance for deformable shapes. Most of these variational models have actually a Bayesian interpretation where the minimized energy corresponds to a maximum likelihood expectation maximization or a maximum a posteriori estimation.

Non parametric statistics For those who consider computer vision an ill-posed problem where humans must intervene to fabricate or dictate "ground truths", the non parametric data analysis tools are adequate. The method always starts by building up for each image or set of images a "feature space", made of vectors containing feature invariant characteristics or moments of shapes, textures, patterns, images, etc. To some of the vectors is associated a ground truth giving the class they belongs to. The question is to segment the feature space into meaningful classes consistent with the ground truth, and therefore to be able to classify correctly other existing or new incoming data. This is always done by a sort of dimension reduction, trying to find the relevant classification parameters and the relevant clusters that they separate in a large learning data set. Machine Learning sees this process as an open loop. The first learning tools as as old as the Perceptron [300], which has now become multi-layer, and as simple as principal component analysis, which is the simplest dimension reducer, followed by independent component analysis [85]. Clustering tools take raw data sets and intend to segment them into meaningful clusters. The most classic clustering (or classification) tools are vector quantization, linear classifiers like Fisher's discriminant analysis, the K-nearest neighbor search, the K-means clustering, the mean shift (Nadaraya-Watson estimator), the EM algorithm, and support vector machines. Probably the main difficulty remains how to do it multiscale and to find hierarchical cluster structures.

PDE's Partial differential equations appear as a natural tool in computer vision for several reasons: first, local filters, actually any adaptive smoothing, like the bilateral or the median filter can be modeled by an anisotropic diffusion [287]. Second, the zoom out operation is modeled by a Gaussian convolution [203] and therefore by the heat equation. The Laplace equation is used for the periodic + smooth image decomposition [250], and the Poisson equation has become a standard tool for performing copy-paste (editing) operations on images [286]. Then, the gradient descent of many

energy methods (like the optical flow or geodesic active contours) ends up being a PDE related to mean curvature motion or to the Laplace Beltrami operator. Also shape derivatives in energy functional give image evolution PDEs. Curve intrinsic smoothing of image level lines (the curvature scale space) leads to the curvature motion, which by the level set method [315] becomes a non linear PDE on the image. Hamilton-Jacobi equations like the eikonal equation also play a role to model shape evolution as a front propagation, particularly for the mathematical morphology operations (erosion, dilation) [314]. Most of the mentioned evolution PDE's are well posed in the viscosity solution sense. Fluid dynamics and the Navier Stokes equation have also been invoked for image inpainting and shape matching.

Statistics Since most image analysis tasks involve some sort of decision based on the observed samples, classical statistics has been widely used. Hence the use of multivariate statistics estimating the parameters of observed density distributions like Gaussian mixtures, the Bayes rule for pattern recognition, the computation of confidence intervals, and hypothesis testing or error analysis strategies. Co-occurrence matrices have been used to model local pixel dependence in textures, invariant moments of local neighborhoods are used in many image matching methods, the normalized cross correlation is the basic tool for performing block matching in stereo vision. In general, well posed pattern recognition uses robust statistics because there are always outliers. In detection algorithms, false positive and false negative rates and the ROC curves are natural measurements of a detection method efficiency.

Computational tools Last but not least, computer vision and image processing need fast, if possible parallel algorithms and fast and efficient optimization methods. Acceleration tools such as dynamic programming (for stereo vision, used on the epipolar lines), the Levenberg-Marquardt method, multi-grid methods (for PDE's), graph cuts (for segmentation and stereovision), fast marching methods, and of course parallel and distributed computing are in the horizon and must be kept in mind for every new algorithm.

6.3 A Scientific Program for IPOL

In this section we shall try to deduce many problems and elementary questions in Computer vision and image processing from a general formulation of image acquisition. This deduction will confirm that a big chunk of the techniques, methods and concepts considered in the preceding section stem naturally from a consideration of the image formation process.

6.3.1 Definition of Image Processing and Basic Computer Vision

The matrix of all image processing and computer vision tasks is the following formula, which summarizes image formation for most biological and technological image capture devices.

$$u = g((S_1 \cdot D \cdot G) * A\mathbf{u}_0) + n. \tag{6.1}$$

In this formula, where $\cdot$ denotes the composition of operators, $u = u(i, j)$ is the digital image (a matrix of discrete values, usually called pixels), g is a contrast change (local or

global) adapted to the image, S_1 is the sampling operator that picks the image values on the discrete image grid, D is the optical deformation caused by the optical focalization device, G is the optical kernel, A is the projective transform associated with the six parameters of the camera position, u_0 is the original photon emission of a physical surface covered by the camera. For a flat physical surface, u_0 can be conceived as an infinite resolution image, in which case the above formula is global instead of being local. The $+n$ term is the CCD's white noise. The additive model is actually a slight simplification. The observed intensity in each captor is a Poisson variable whose intensity is $G * \mathbf{u}_0$. For decent exposure times, the difference between the intensity and the observed value is close to a Gaussian variable.

Formula (6.1) describes in the simplest possible way image formation. According to the founders of image processing and image analysis in the sixties and seventies of the past century, the ultimate goal of image processing and image analysis is to get back from the digital image u to $\mathbf{u}_0$, the original landscape, and its 3D physical shape. But to do this, there are six operators to inverse! All steps of this inversion interact strongly. This inversion cannot possibly be complete with just one image. Fortunately, the theory indicates that with a few images of the same scene the inversion becomes feasible, and a whole 3D map of the surrounding world reconstructible.

Finally, A contains the scene geometry. It gives the relative position of the camera A observing the patch $\mathbf{u}_0$. Thus, A is defined by six position parameters (translation, rotation). To this must be added the camera parameters which define the distortion model D, a smooth deformation of the image plane.

One can define the main goal of basic image processing as to recover the real image patch $\mathbf{u}_0$ *from the digital image* u*.*

A definition of basic computer vision is to recover everything about the image formation operators, which permits in principle an arbitrarily accurate visual reconstruction of the 3D surrounding world.

For both image processing and computer vision, we must distinguish the case where only one image is available and the case where multiple snapshots of the same scene are available. The main goal in all cases is to process optimally each image knowing the other ones.

To provide a visible image from a single raw image an automatic image chain must estimate automatically the noise and blur. The optical distortion can also be corrected when straight segments are available, or a distortion model.

From the image processing viewpoint, there is little difference between single image and video processing, the main difference being that all operators and $\mathbf{u}_0$ may depend on a time variable t. The simultaneous processing of several images of the same scene and video processing are almost equivalent problems. In the video case, the use of the time arrow leads to a slightly different implementation, though. On the other hand having several images of a same scene opens the way to a 3D reconstruction.

6.3.2 Single Image and Video Processing

In this section we deduce all processing operations implied by the image formation formula if one wants to recover the native image (or video) $\mathbf{u}_0$. Thus, each paragraph below deals with one of the image formation operators and its inversion.

Dealing With S_1: Interpolation and Sampling Theory

There is a single universal theory for signal or image sampling, based on the physical fact that only band-limited signals can be acquired by any physical device because of the uncertainty principle. We shall in the sequel take the assumption that the image is sampled on a regular grid. Then the Shannon-Whittaker formula [316] gives the exact ways to pass from samples to the continuous image and vice-versa. Unfortunately this theory is not exactly realizable because it requires infinitely many samples. Thus, the discrete version is based on the false but necessary assumption that the image is periodic. Once this bias is accepted, however, the Shannon-Whittaker theory can be exactly transposed to a rigorous finite discrete framework treating images as trigonometric polynomials sampled on rectangular grids. Then exact DFT and DCT formulas permit to interpolate and to re-sample the image. This first approximation actually raises the questions that any scientific community on imaging should have resolved. The basic image processing tasks therefore deal with the manipulations of image samples to interpolate, and re-sample, therefore performing all classic geometric deformations: translation, rotation, zooms in and out, homographic deformation. Also must be raised the choice of a convolution kernel to maintain an aliasing free image. Last but not least, compression strategies should be discussed thoroughly on line with evaluation of the resulting image quality. This leads to the following list of algorithms and questions that *any* image processing software must consider. For most of the algorithms, references are given to seminal papers and when available to a corresponding IPOL article.

- Given a band-limited image, how far is its DCT interpolation on a rectangular grid (thus ignoring the samples outside the rectangle) from its exact Shannon-Whittaker interpolate? The error caused by the ignorance of outside samples must be experimentally and theoretically quantified.
- Almost exact spline-based [346] algorithms for re-interpolating the digital image on other grids (translation, rotation, zoom-in, zoom-out, homographies) [145, in IPOL].
- Yaroslavsky's clever resampling algorithms for rotation, and their extension to any fast implementation of affine maps [375]
- Aliasing error: to quantify in theory and in practice the aliasing error caused by image under-sampling, as a function of the point spread function (PSF) standard deviation
- Show in theory and in practice the best choice for the Gaussian kernel standard deviation required before sampling with minimal aliasing [255]. Compare with other optimal blurs (the prolate functions)
- Compare all linear zoom-in algorithms, in a hierarchy from zero order spline to Shannon-Whittaker [145, in IPOL]
- Discuss smart zoom-ins, which violate Shannon's conditions, but give sharp images, at least in appearance aliasing-free [143, 144, 147, in IPOL]
- Desaliasing or super-resolution: compare the strategies to reinsert missing samples in an aliased image [354]
- Image compression: a comprehensive review of Lempel-Ziv coding, Huffman coding, JPEG 1991, JPEG 2000, Lossless compression algorithms like LOCO, geometric compression algorithms, graphic compression algorithms
- A thorough explanation and analysis of image classic formats.

Of course compressed sensing promises new forms of cameras (not yet existing, though) for which clearly the above plans might request someday a full reexaximination. Nevertheless, compressed sensing simulators, typically by minimization of an L^1 norm, are of immediate interest and some are already submitted to IPOL [211].

Dealing With $+n$: Denoising

This is probably the second operation to consider in image processing after sampling, because it is the main perturbation of the sampling operation: the acquired sample is a sample of a Poisson noise whose intensity is the "real" image, with some thermal or electronic noise added from the CCD itself. Estimating the noise from the raw image itself can be done on special patterns if the camera is at hand. Otherwise the noise is estimated from robust statistics in an image or a set of images. The noise being signal dependent, the estimation must be signal dependent, and permit to retrieve the noise model for the raw image. If the acquired image is not raw, and has undergone transforms (typically compression and contrast changes), the resulting noise is no more white. It is signal dependent and also scale dependent. For example compression usually performs a sort of denoising at the finest scale. Thus noise estimation must give noise curves that are signal and scale dependent. The Anscombe transform and its generalizations can be used to recreate an image with white noise from an image with structured noise. A denoising benchmark needs a method to create noiseless images. The best method seems to be to take high resolution images and to apply a Shannon zoom by a large factor, of the order of 8 or 16, to ensure a very low noise level. In other terms, the image can be convolved by a Gaussian filter of standard deviation 6.4, to ensure that almost no aliasing is present by a 1/8 subsampling. Such data and methods should be published at IPOL. There are many kinds of colored signal dependent noise, but an Anscombe transform and a diagonal filter on the Fourier coefficients permit usually to whiten the noise. Nevertheless, impulse noise and thermal line multiplicative noise require a completely different treatment. Because most images have undergone a compression, the observed noise is generally not white. Its estimate at the first scale can become very imprecise. Thus denoising JPEG images is a different and more complicated topic than denoising raw images. These considerations lead us to the following list of algorithms and problems, each being a potential IPOL article.

- Estimate signal and frequency dependent noise from any given image [232]
- Anscombe transform: apply the Anscombe transform to recreate an image with white noise from an image with structured noise [323]
- Publish noiseless images and the method to create them. Based on them, a denoising benchmark is easy. It can be restricted to white noise only and compute sound error distances, like the RMSE or the PSNR.
- Wiener ideal denoising filter
- Wavelet thresholding [82]
- Wavelet coefficient correlation based denoising [289]
- DCT denoising [378, in IPOL]
- Total variation denoising [305]
- Patch based method: NL-means [63, in IPOL]
- Patch based methods with fixed basis: BM3D [90]

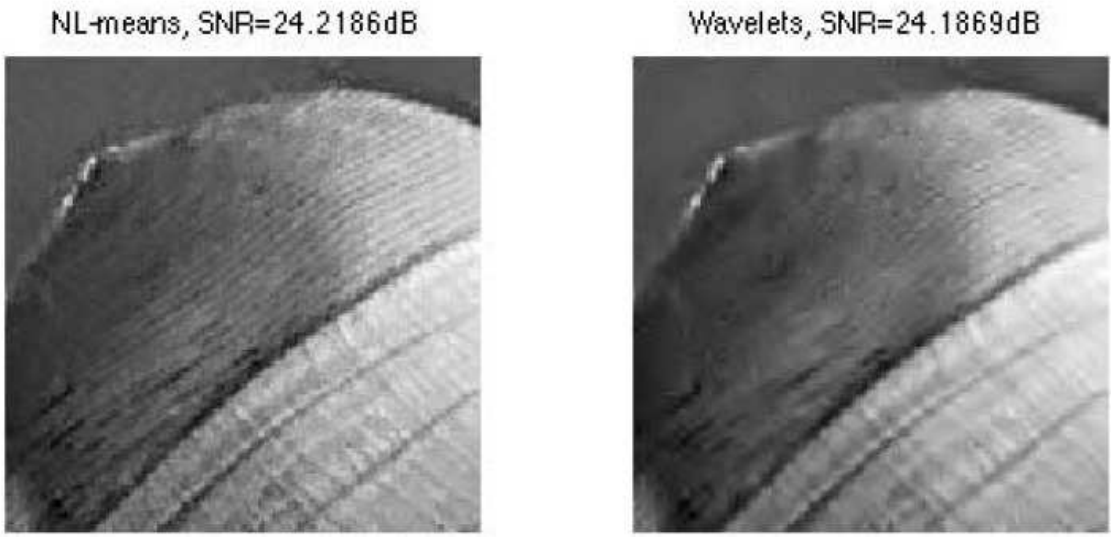


Figure 6.1: Comparing NL-means and wavelet thresholding. Exp: Gabriel Peyré

Figure 6.2: Denoising an old photograph by nonlocal means.

- Patch based with dictionary learning: KSVD [237], LPE [376]
- Patch based Bayesian: Non local Bayes
- Make a denoising benchmark on: small noise $\sigma = 2$ to 5, medium $\sigma = 10$ to 40, high $\sigma = 50$ to 100
- Algorithm removing impulse noise: conditional median filters and others, filters removing high curvatures caused by impulses (typically mean curvature motion)
- Infrared or thermal camera line multiplicative noise: total variation denoising (Moisan method)
- Infrared or thermal camera line multiplicative noise: the midway technique [336]
- "JPEG Noise clinic": multiscale denoising combined with multiscale signal dependent noise estimation. This noise clinic can be applied with any denoising algorithm working initially on white noise
- Extensions of all these algorithms to video denoising.

Dealing With S_1 and G: Demosaicking and Super-resolution

Demosaicking A *raw image* is an image recorded by a digital camera (or an image scanner), before any processing. Most reflex cameras and more and more compact cameras provide this raw data. The camera contains a CCD or a CMOS matrix that simply records a number proportional to the number of photons hitting each CCD. These pixels have color filters and are specialized in the red, green and blue colors. Thus, the raw image is not visible ; each pixel contains only one of the three colors. The operation to infer the other three colors at each pixel is a tricky and challenging interpolation method called *demosaicking*. Previously to demosaicking, the image has to be denoised because noise is not interpolable. For off the shelf optical cameras, demosaicking is the first or second operation performed on the raw image: in principle denoising must be applied first. Demosaicking is a super-resolution, or antialiasing operation by which missing samples are recreated from existing ones. The difference with super-resolution is that in super-resolution only one channel is available while in demosaicking the channels exchange information. Thus super-resolution algorithms must be considered first, and they actually are used to initialize the green channel in the Bayer configuration, where one out of two pixels is green. A demosaicking benchmark is relatively easy. Indeed, there are two well defined criteria: the RMSE and a zipper effect measurement. However, the choice of the benchmark database is still problematic, certain databases favoring certain kinds of algorithms. Thus the results may depend strongly on the database. It is very easy to create a ground-truth, since test images can be original images that are first mosaicked by retaining a single color at each channel. Last but not least, compressed sensing, a recent theory initiated by Emmanuel Candès, Terence Tao and DAvid Donoho, proposes to take advantage of the image sparsity in some Hilbert bases to capture directly the image by a tiny number of linear tests.

- Blind point-spread function estimation from several digital images
- Pattern based point-spread function estimation
- Super-resolution algorithms [284]
- Hamilton Adams demosaicking [64, in IPOL]
- Zhang Wu demosaicking [148, in IPOL]
- Self Similarity driven demosaicking [64, in IPOL]

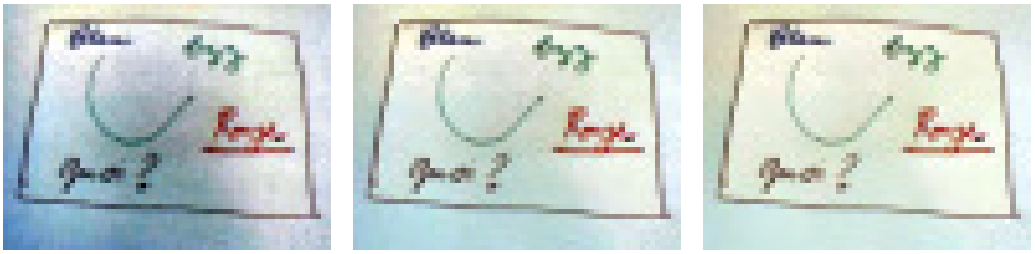


Figure 6.3: *Left:* Original image, *center:* Retinex partial differential equation with $t = 10$, *right:* Retinex with $t = 15$. Colors are enhanced and the background clutter eliminated.

- Malvar, He, Cutler, Lin demosaicking [146, in IPOL]
- Getreuer demosaicking with contour stencils [149, in IPOL]
- Gunturk demosaicking [142, in IPOL] }
- Learning based demosaicking (KSVD) [376]
- To create and enrich a demosaicking database of many challenging patches and to conceive a demosaicking benchmark
- Compressed sensing: reconstruction algorithms from sparse samples [72].

Dealing With g: Color and Contrast

Color balance and color contrast adjustment are linked. In both cases some histogram manipulation is performed, typically monotonous on each channel, and more or less coordinated. Its goal is to simulate the "Retinex" capability of human perception to enhance contrast in shadows, and to perceive the relations of colors independently of ambient light and shadows. The goal of white balance is to restore colors as independent of possible of the physical lighting conditions. Here again the work forks depending on whether we have at hand the camera or not. If we have the camera, a color matrix estimating how wavelengths have been mixed in the color pixels can be devised. Otherwise, white balance simply weights the colors so that image extrema are white. A fast implementation of the Land Retinex based on a new PDE, $\Delta u = div(\delta(Du_0))$, has been devised recently[3]. This surprising Poisson equation manages to reconstruct from a given image u_0 an image that has exactly the same singularities, but eliminates all small gradients ($\delta(s) = s$ if $|s| \geq \delta$, $= 0$ otherwise.)

- 3D color cube visualization [231, in IPOL]
- Histogram equalization (by channels or only the grey level) [288]
- Simplest color balance [226, in IPOL]
- Color spaces, their meaning, and the conversions between them
- Retinex algorithm based on Poisson equation [228, in IPOL]
- Multiscale Retinex [188]
- Screened Poisson equation as smooth background subtraction [46]
- Blind gamma correction [120]

[3]Morel, J.M. and Petro, A.B. and Sbert, C., Fast implementation of color constancy algorithms, Proceedings of SPIE, 7241 (724106), 2009.

Figure 6.4: The Retinex PDE adjusting contrast and color.

- Contrast control by Poisson editing: concave function of gradient on lower level set [286]
- Local color correction (HP method) [155, in IPOL]
- Invariant to g: level set tree transforms [252]
- Invariant to g: level line tree transforms (FLST) [230]
- Shape extraction based on meaningful level lines [97]
- Contrast invariant FLST based restoration: elimination of small shapes [356]
- Granulometry [355]
- Level line processing: curvature equations [254, in IPOL]

Dealing With the Convolution Kernel G: Deblurring

This subject is extremely challenging. The cases where an image is blurry, well sampled, and where the blur kernel is known are rare. In general the blind estimation of the blur from a single image is rarely possible. Thus deblurring must be considered as a serious problem only when the camera calibration is well known and its blur completely modeled. Nevertheless classic deblurring algorithms can be compared for simulated data, in which case Gaussian blur or motion blur are the most obvious candidates for a benchmark of deblurring algorithm. Needless to say, the deblurring performance depends on the amount of noise. If there were no noise, a Gaussian blur could be completely eliminated by a simple division in the Fourier domain. But if there is noise, the division blows up the noise high frequencies. Thus, deblurring Gaussian blur is equivalent to the denoising of a colored noise.

The "digital revolution" leads to a thorough revision of the very concept of camera blur. See the recent works of Ramesh Raskar at MIT media labs. Frédéric Guichard's 2005

invention permits to extend the camera depth of field by numerical means, taking the best advantage of an optical defect, the chromatic aberration[4]. The mathematical design changes the camera architecture.

- Blur simulation (gaussian, motion) and Wiener deblurring filter (under a noise assumption)
- Total variation deblurring [304]
- Invertible blurs: the flutter shutter theory [294]
- Invertible blurs: motion invariant photography [219]
- Blind deblurring methods [78]
- Psf estimation from photographs of patterns [94, in IPOL]
- Psf estimation from one or several photographs of the environment
- link to super-resolution [264]
- methods that increase the depth of field by combining photographs taken with different depths of field [262]

Dealing With the Camera Distortion D: Internal Camera Calibration

The camera model (for each given ISO, aperture and focal length) can be estimated from a small set of arbitrary images, and even often from a single photograph. One of its innovations is to use systematically the images themselves as camera mires. A pioneering work in that direction (but that was left unfinished) is the camera self-calibration project [100]. Projective geometry tells us that distortion is eliminated when a straight line remains straight, and that a flat image undergoes a homography. These properties can be used for two different distortion correction methods. The following list gives some hints of the major algorithms that should be discussed in IPOL.

- Distortion estimation and correction by plumb line methods [160]
- Blind distortion correction [21, in IPOL]
- Distortion correction by matching a flat pattern [52]
- Devernay-Faugeras blind distortion correction [100]
- Lavest global calibration method [220], extension of the Zhang technique [381]
- Bundle adjustment [344]
- Camera self-calibration [122]
- Correction of chromatic aberration (differential distortion between color channels) [53]
- Parametric distortion models: theory and comparison
- Nonparametric distortion models: polynomial, thin plates, rational, etc. [159].

[4] `http://www.dxo.com/var/dxo/storage/fckeditor/File/embedded/2009_EI_EDOF.pdf`

6.3.3 Multi-images processing

As soon as we have more than one image of a given scene, particularly if the images are taken by the same camera and almost simultaneously and if the position of the camera varied little, new possibilities open up that completely change the panorama for most image processing tasks. Most of what will be said on two images, but the generalization to an arbitrary number of images will be implicit. We will first review what can be done from the image processing view point, and then pass to 3D reconstruction issues. The now-classic Hartley-Zissermann [168] and Faugeras [123] books explain how to identify A, therefore fixing the camera position and permitting a 3D reconstruction by triangulation from two or more photographs of the same scene.

Image matching

The impact of online zero-parameters algorithms can be immense. Probably the most influential recent computer vision algorithm is the zero-parameter SIFT method [235] published en 2004, already quoted by more that 5000 papers, and used in all domains of imaging and robotics. This method can be applied with closed eyes to any image pair. Its mathematical analysis [255] shows that it is perfectly rigorous. This is a clever method: Recognizing objects at different distances implies identifying and compensating different blur kernels.

Although it is perfectly scale invariant, SIFT is, however, only partially invariant to perspective. In general image descriptors under geometric and contrast invariance have been recently playing a central role in computer vision, for image comparison and indexing, in particular since the SIFT descriptor was proposed by David Lowe. Since then, many attempts to extend Lowe's work to larger invariance groups have been made, specially to the affine group (Hessian-Affine, Harris-Affine, LLD or MSER). None of them have proved performing enough, and in practice, people still prefer to use SIFT. However the ASIFT method, by an adapted sampling of the affine space, shows that the cost of extending SIFT to the affine framework is not prohibitive at all.

Fig. 6.5 shows what perspective invariance means in practice: the same object seen under various angles can undergo strong geometric distortions by factors (the so-called transition tilts) that can be as large as 40. In the example of the figure, the same object has a 36 transition tilt, and is still recognized very reliably (116 features match correctly).

On the other extremity of deformation spectrum, the images to be registered can be very similar. This happens with successive frames in a video or in small baseline stereovision. Then one can envisage *optical flow* techniques, that find a dense registration field.

- SIFT (scale invariant recognition) [235]
- ASIFT (makes SIFT affine invariant) [377, in IPOL]
- MSER (not exactly affine invariant but shape based image recognition) [235]
- RANSAC and variants eliminating wrong matches by scene coherence (ORSA) [251]
- SURF (fast approximate SIFT) [33]
- Discussion of all steps in the SIFT method: heat equation and scale sampling, key point accuracy [66], right thresholds deduced from noise estimator for the key point detector, orientation histogram peaks, Orientation histogram mode detection [95], histogram distances [293], the problem of multiple shapes, classification of different

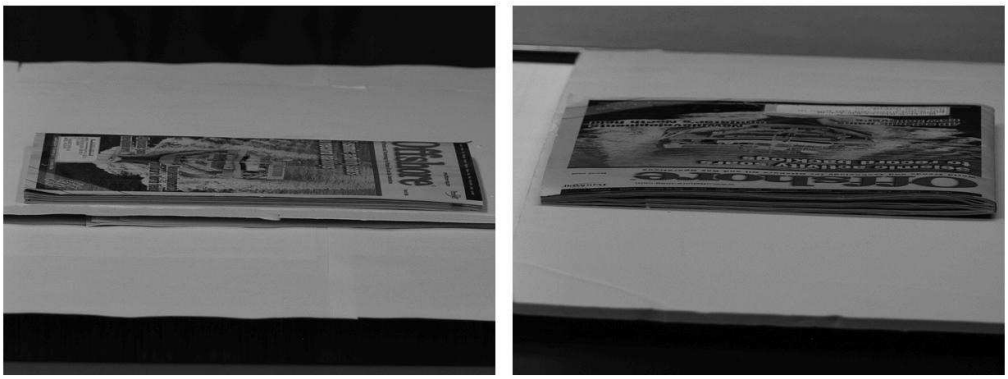


Figure 6.5: In this pair, ASIFT finds 116 correct matches and 4 wrong matches.

Figure 6.6: Projective invariant image comparison: ASIFT method

coherent groups with respect to a geometric transform (homographies), similar shape grouping (link to gestalt theory) [317].
- Optical flow, Horn and Schunck [177], TV-L1 Optical flow [379], ...

Stereovision from Multiple Views or Video

Surprisingly, 3D reconstruction from multiple view and "structure from motion" have been often considered as separate topics, while they are in fact extremely similar and almost indistinguishable techniques. The following list gives the very basic algorithmic lines that should be represented at IPOL. Starting from two or more images, they attempt to go back to a 3D information, starting often with a stereo rectification and the computation of a disparity map. This first step, namely a local image matching, is fundamental. It is divided between dense methods, which have regularizing terms because of the ill-posed nature of the problem, and non-dense methods working with local block matching. The main problems for computing the disparity map are: the *fattening effect* occurring with block-matching, the still overwhelming complexity of dense variational methods, and the false matches caused by noise, image self-similarities and occlusions. Thus there are hundreds of titles proposing stereo algorithms. What follows is just a sample.

- epipolar stereorectification: several theories [167], [253, in IPOL]
- bundle adjustment [344]
- multiscale block matching
- subpixel block matching [307]
- projective invariant block matching (after stereorectification)
- *a contrario* block matching [306]
- Coherence of block matching: left-right coherence, noise threshold criterion, min filter, bilateral matching, adaptive windows to elevation model, Coherence of block matching: 3D coherence analysis [313]
- Fattening free block matching [48]
- Interpolation of disparity maps
- Detection of occlusions
- variational matching methods (graph cuts [207], belief propagation, dynamic programming,...) [309]
- The epipolar geometry of a motion [50]
- Camera navigation from a movie
- Structure from motion [204].

3D Data Point Sets Processing and Rendering

For a sake of concision, we write "point cloud" or "cloud" for a data point set cloud. Point cloud processing is an integrant part of computer vision because such clouds can be acquired by imaging methods, such as binocular or multiimages stereo, or "structure from motion". Furthermore such clouds can be obtained by active lighting stereo setups. Once they are obtained the clouds must be oriented (being the skin of objects with interior and exterior), they must be triangulated, and several clouds obtained for camera views must be merged into one. The surfaces must be segmented, registered, and recognized. This

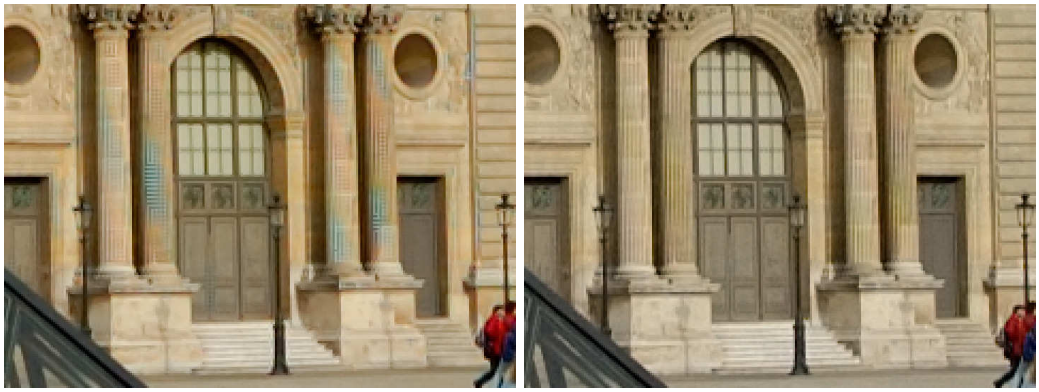


Figure 6.7: Wrong and good demosaicking: Nikon NX and Buades, Coll, Morel Algorithm (in DxO OPv5).

leads to a relatively standard list of questions that are only recently being resolved with numerical efficient and mathematically sound algorithms.

- Point cloud fast neighbor search
- Database of multiscan raw point clouds with high precision [104, in IPOL]
- Raw cloud scale space meshing and orientation [103]
- Raw cloud triangulation (pivoting ball algorithm) [35]
- Computation of local moments and local orientations (and principal curvatures and principal directions which are linked to them)
- Cloud multiscale segmentation (in ridges and hollows)
- Scale space merging of multiple scans, super-resolution and fusion of matched point clouds [101]
- MSER applied to the mean curvature or another scalar function [102]
- Construction of local patches and SIFT for cloud matching
- Tritangent cloud matching [303]
- Non rigid registration algorithms
- Fusion of photographs and triangulated point clouds: texture projection
- Fast 3D visualization algorithms
- Stereo by active lighting and registration

6.4 Image Analysis and Understanding

Image understanding is this part of computer vision delivering end results, namely semantic detections of any kind on an image. However a part of it still can deal with low level features, which can in turn possibly be used for a final semantic understanding. Thus, we shall first mention the many feature extraction algorithms, and then move on to mention briefly semantic algorithms, which are by far more challenging.

6.4.1 Single Image Analysis: Geometric Features and the Gestalt Program

Having reliable feature detectors on images is crucial. Since their main use is in the very first steps of an image analysis chain, they must by all means be automatic. Nevertheless, surprisingly, to the best of our knowledge there are very few fully automatic, parameterless feature detectors. We can mention the SIFT transform [235], the MSER method [241], and the Line Segment Detector (LSD) [157]. Since SIFT and MSER are always used in the context of image matching, we have considered them in the corresponding section. Thus we shall list here single image detectors. Again, no parameterless detector should start without knowledge of the noise level, which can cause many false detection (while if the noise level is very low, almost every detection is right). Thus all of the mentioned detectors should be preceded by an accurate noise estimator. In essence, most detectors proposed in the literature correspond to features mentioned as basic geometric features (or gestalts) in the Gestalt program. Many of them involve a 1D histogram analysis providing a meaningful mode detector. Most of them actually analyze angles, parallelism, alignments, convexity, curve smoothness, singularities linked to segments or curves (angles, junctions). The final goal of these partial detector should be a complete image indexing in regions, boundaries, and various groups linked by their common color, orientation, texture, etc.

- On line gestalt games predicting the perceptual thresholds in alignment and cluster detection
- On line games evaluating learning capabilities of structured perceptual organization (for example the game proposed by Fleuret, Geman, et al. [127])
- Edge detectors (Canny and variants) [73]
- Hough transforms for lines, circles, etc. [28]
- Edge detectors with *a contrario* models [97]
- Line segment detectors with *a contrario* model [157] [158, in IPOL]
- Segment alignment detector
- Point detector
- cluster detector (against a uniform background assumption) [98]
- Point set alignment detector
- Corners, T-junctions, X-junctions detectors
- Curve detectors (the good continuation gestalt)
- Amodal and modal contour completion algorithms (Kanizsa) [192]
- Shape group detectors
- Segmentation methods: variational methods [205]
- Segmentation *a contrario* methods [97]
- Histogram mode detector (against a flat background assumption)
- Histogram mode detector (against a decreasing background histogram assumption) [95]
- Group of parallel segments detector (on histograms of segments detector) and more generally vanishing point detector [20]
- Constant width detector (on histograms of distances parallel segments in front of each other)

- Constant length detector (on histogram of segment lengths, background model decreasing)
- Curvy edge detector (by good continuation on meaningful edge pieces)
- Alignment of dots (for extremities of segments that are not aligned)
- Constant angle detector (for chained segments, constant curvature), in particular regular polygon detector (squares, polygons)
- Vanishing point detector [20]
- Harris point detector [166]
- Group of similar shapes detector (can be based on SIFT or similar) [235]
- Image variational segmentation: grey level, color, texture
- Image grouping algorithms: contrarily to segmentation, a point can have various indexes and they can be nonlocal.

The zero-parameter requirement is easy to understand: Most of our visual perceptions are *sure*. Of course nothing is ever absolutely sure but, still, certainty can be evaluated with probabilities. The line segment detector due to Rafael Grompone [157] is a good example of a many-times-solved problem that has been revisited, to finally get a fast zero-parameter detection working on *any* digital image. Fig. 6.8 illustrates one result of this automatic algorithm.

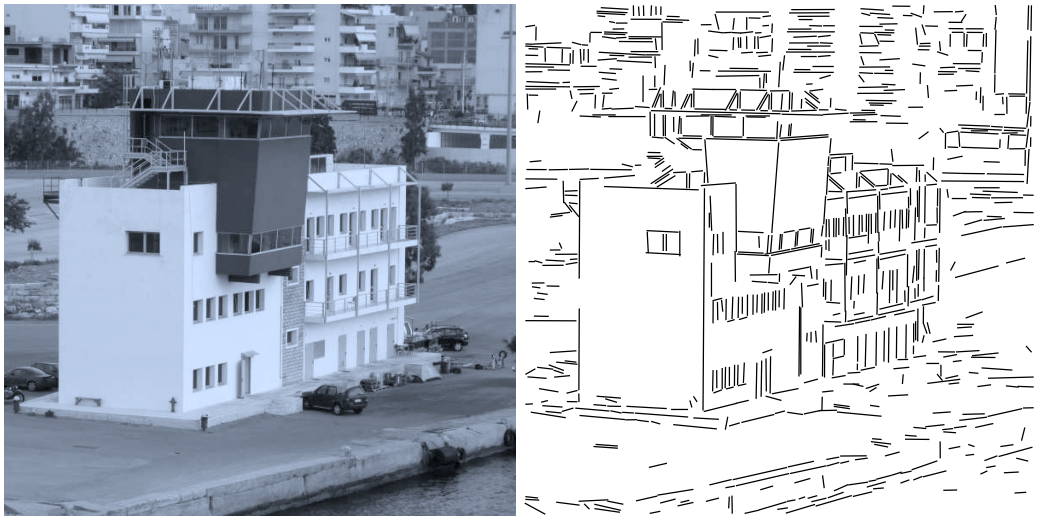


Figure 6.8: Result of the zero parameter line segment detector [157].

6.4.2 Object Recognition (Learning Methods)

It is not clear if this discipline is mature for online display. Indeed, it requires the build up of a very large learning database, of a subset used for learning and of a subset used for testing. The learning requires a *ground truth*, usually made manually by humans employed in developing countries. The scientific value of such ground truths is discussable, but, above all, it remains difficult to demonstrate such learning algorithms on line, since they depend so much on big databases and subjective characters. Nevertheless some machine learning algorithms have had a strong impact and machine learning by itself is

an absolutely legitimate problem, in spite of its Achilles heel of a manual human ground truth. Probably its most striking success has been with face detection algorithms, which are by now operational in most digital cameras. Thus the following kinds of algorithms can definitely be published on line and tested on line by uploading images:

- Face detection algorithms [358]
- Generic object recognition learning algorithms (Viola-Jones)
- Character recognition algorithms [107]
- Cursive writing recognition algorithms
- Recognition of other objects by learning? (vehicles, animals, etc.) [124]
- Human detection
- Gait analysis [212]

6.4.3 Graphics

There are several close links between Computer Graphics and CVIP, although the goals are obviously different. Computer Graphics aims at recreating realistic or imaginary scenes by 3D modeling and rendering techniques. Nothing opposes a publication in IPOL of rendering algorithms. Taking a set of geometric objects (whose skin is usually triangulated) along with their colors, reflectance properties and motion properties, given also a set of light sources, rendering algorithms perform accelerated variants of ray tracing methods.

Second, ray tracing creates the same kind of perturbations present in real images, in particular noise and aliasing. However the noise and aliasing are of a special kind and there is additional information in the G-buffer which should permit to boost image restoration techniques on synthetic images.

Hybrid computer graphics mixes 3D information and images taken in a real scene with implants of synthetic objects. Thus a handy representation of 3D point clouds acquired from real scenes by computer vision techniques is necessary. The insertion of images or movies in each other is often called "editing" and involves a series of copy-paste operations, be it to insert or remove an object in an image or a movie. The removal and replacement by another texture is called "inpainting".

Finally, the modeling of surface aspect relies often on procedural texture synthesis algorithms, able to "paint" quickly the geometric objects. In that context, texture analysis and synthesis (from examples) is a clear meeting point between computer graphics and image analysis.

- Rendering algorithms (given geometry and light sources) a key algorithm founded by [190]
- Image restoration of rendered images (for the noise and aliasing)
- Insertion of new parts in images or movies : this operation is called "editing". The key algorithm is Poisson editing [286]
- Image inpainting algorithms [45]
- Random phase texture synthesis from examples [138, in IPOL]
- Efros Leung [109] texture synthesis from examples
- Wei Levoy [364] texture synthesis from examples

- Steerable Filters (used for texture synthesis)
- Heeger Bergen [169] texture synthesis from examples
- Cartoon+texture decomposition [65, in IPOL]
- Inpainting algorithms [45, 376],...

6.5 Conclusion: Journal Methodology

We have revised in section 6.2 all key notions (and actually 2000 most used key words) in Computer Vision and Image Processing (CVIP). The 2000 key words cover probably most research topics, but give little deductive indication on how the discipline has emerged and should evolve. In section 6.3 we have proposed a organization of the topics based on the main image formation model. This model permits to categorize most operations in image processing by their inherent invariance requirements, and by the inverse problems they foment. When it comes, however, to image analysis, the main source of inspiration for understanding the work done or to be done has a phenomenological inspiration, in particular Gestalt theory for the geometric features, and psychophysics for texture analysis. Finally, when image formation, geometric modeling and psychophysics are mute, one has to recur to Machine Learning, where the features and objects are no more modeled, but simply pointed out by humans. This is a strong methodological limitation, which also hinders probably online publication, unless some fixed learning and test databases are given and the focus is on comparing learning algorithms. All in all, the IPOL program, namely the kind of algorithms that should have a prior focus, come from image processing and, level computer vision (feature detectors) and 3D computer vision. Indeed, these topics allow for the upload of one or a few images (or a short video sequence), and perform focused operations on the data: one restoration, one detection. Nevertheless, each restoration interacts with other restorations, each detection interacts with other detections and with image processing issues. Therefore, even in this restricted context, it is unavoidable to put up a strict requirement of making parameterless algorithms. No building block on any image analysis or processing chain should depend on any parameter. The obvious consequence if they did would be a complexity blow up for chains composed of more than two operations. Thus, the IPOL requirement that algorithms should be parameterless, as much as possible, is also the obvious technical requirement to build up an imaging science able to build up elaborate processing chains. It turns out that almost no well known algorithm matches these requirements, with only a few exceptions. Thus rethinking well known algorithms in this perspective is a legitimate research topic. Experience has shown that the "republication" in IPOL of a well known algorithm raises non trivial questions which make this republication legitimate. So much so, that in general this gives a second chance to the topic that was only apparently solved, and also often lead to fully reconsider the general opinion on the state of the art.

For example the study of denoising algorithms that will lead to the (re)publication of seven algorithms shows that the hierarchy of algorithms given in the past ten years was illusory. As shown in by "{O}ktem and Yaroslavsky [383], translation invariant DCT-denoising has a performance equivalent, or sometimes even better than later more sophisticated propositions such as translation invariant wavelet thresholding. Furthermore, this study uncovers the fact that all algorithms fail with very low noise ($\sigma < 2$) and with large noise ($\sigma > 40$), creating in the latter case unacceptable artifacts. The online testing reveals many flaws or limitation that were not apparent in the original publications. On other

very successful algorithms, like SIFT, many question open up with a closer analysis, such as the problem of repetitive shape matching, or the question of whether SIFT really is scale invariant as claimed with its sparse scale sampling.

Chapter 7

Examples

Contents

Abstract

This chapter contains two articles adapted from their original online version published in IPOL [226, 228], with supplements about the design of the code and online demo.

The algorithm proposed in the second article is extremely basic, and probably known for a long time. It was nevertheless proposed for publication at IPOL and accepted. Indeed, the image quality improvement obtained by many recently proposed sophisticated color correction methods seems to rely essentially on a (not explicit) final color balance. Thus, it seemed important to make the research community aware of this alternative. Color balance can be used as a sanity check against uselessly complicated color perception and color correction theories.

7.1 Retinex Poisson Equation: a Model for Color Perception

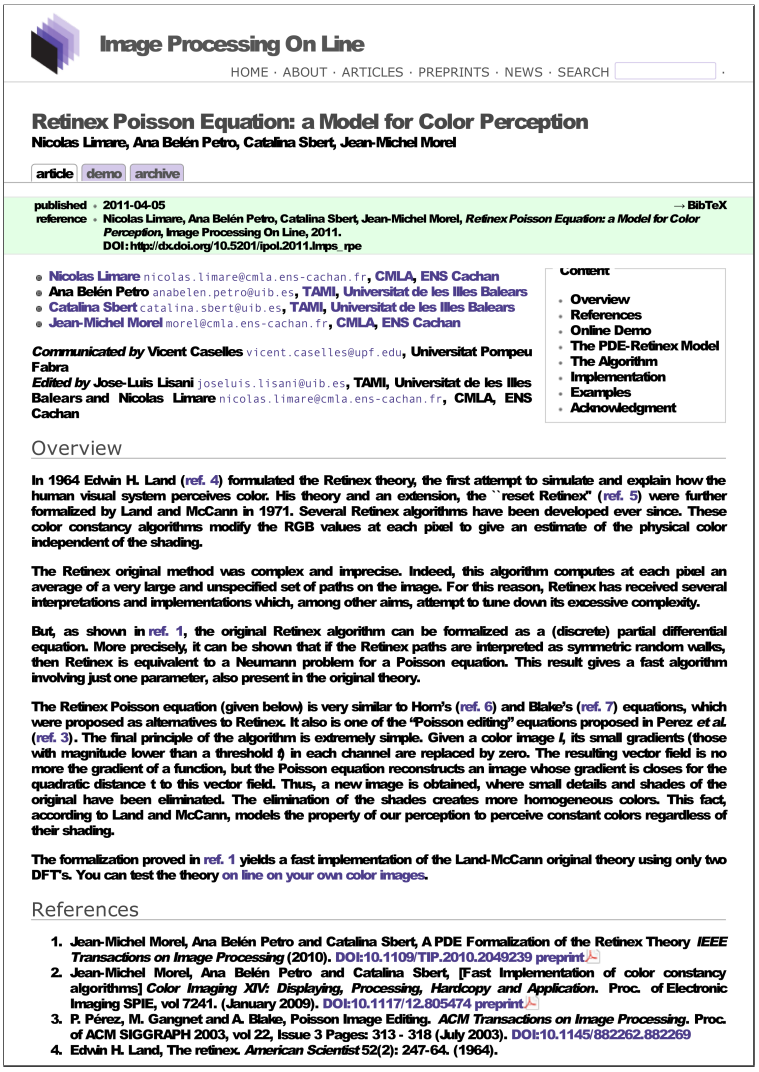


Image Processing On Line

HOME · ABOUT · ARTICLES · PREPRINTS · NEWS · SEARCH

Retinex Poisson Equation: a Model for Color Perception

Nicolas Limare, Ana Belén Petro, Catalina Sbert, Jean-Michel Morel

article demo archive

published 2011-04-05 → BibTeX
reference Nicolas Limare, Ana Belén Petro, Catalina Sbert, Jean-Michel Morel, *Retinex Poisson Equation: a Model for Color Perception*, Image Processing On Line, 2011.
DOI: http://dx.doi.org/10.5201/ipol.2011.lmps_rpe

- Nicolas Limare nicolas.limare@cmla.ens-cachan.fr, CMLA, ENS Cachan
- Ana Belén Petro anabelen.petro@uib.es, TAMI, Universitat de les Illes Balears
- Catalina Sbert catalina.sbert@uib.es, TAMI, Universitat de les Illes Balears
- Jean-Michel Morel morel@cmla.ens-cachan.fr, CMLA, ENS Cachan

Communicated by Vicent Caselles vicent.caselles@upf.edu, Universitat Pompeu Fabra
Edited by Jose-Luis Lisani joseluis.lisani@uib.es, TAMI, Universitat de les Illes Balears and Nicolas Limare nicolas.limare@cmla.ens-cachan.fr, CMLA, ENS Cachan

Content
- Overview
- References
- Online Demo
- The PDE-Retinex Model
- The Algorithm
- Implementation
- Examples
- Acknowledgment

Overview

In 1964 Edwin H. Land (ref. 4) formulated the Retinex theory, the first attempt to simulate and explain how the human visual system perceives color. His theory and an extension, the ``reset Retinex'' (ref. 5) were further formalized by Land and McCann in 1971. Several Retinex algorithms have been developed ever since. These color constancy algorithms modify the RGB values at each pixel to give an estimate of the physical color independent of the shading.

The Retinex original method was complex and imprecise. Indeed, this algorithm computes at each pixel an average of a very large and unspecified set of paths on the image. For this reason, Retinex has received several interpretations and implementations which, among other aims, attempt to tune down its excessive complexity.

But, as shown in ref. 1, the original Retinex algorithm can be formalized as a (discrete) partial differential equation. More precisely, it can be shown that if the Retinex paths are interpreted as symmetric random walks, then Retinex is equivalent to a Neumann problem for a Poisson equation. This result gives a fast algorithm involving just one parameter, also present in the original theory.

The Retinex Poisson equation (given below) is very similar to Horn's (ref. 6) and Blake's (ref. 7) equations, which were proposed as alternatives to Retinex. It also is one of the "Poisson editing" equations proposed in Perez *et al.* (ref. 3). The final principle of the algorithm is extremely simple. Given a color image *I*, its small gradients (those with magnitude lower than a threshold *t*) in each channel are replaced by zero. The resulting vector field is no more the gradient of a function, but the Poisson equation reconstructs an image whose gradient is closes for the quadratic distance t to this vector field. Thus, a new image is obtained, where small details and shades of the original have been eliminated. The elimination of the shades creates more homogeneous colors. This fact, according to Land and McCann, models the property of our perception to perceive constant colors regardless of their shading.

The formalization proved in ref. 1 yields a fast implementation of the Land-McCann original theory using only two DFT's. You can test the theory on line on your own color images.

References

1. Jean-Michel Morel, Ana Belén Petro and Catalina Sbert, A PDE Formalization of the Retinex Theory *IEEE Transactions on Image Processing* (2010). DOI:10.1109/TIP.2010.2049239 preprint
2. Jean-Michel Morel, Ana Belén Petro and Catalina Sbert, [Fast Implementation of color constancy algorithms] *Color Imaging XIV: Displaying, Processing, Hardcopy and Application*. Proc. of Electronic Imaging SPIE, vol 7241. (January 2009). DOI:10.1117/12.805474 preprint
3. P. Pérez, M. Gangnet and A. Blake, Poisson Image Editing. *ACM Transactions on Image Processing*. Proc. of ACM SIGGRAPH 2003, vol 22, Issue 3 Pages: 313 - 318 (July 2003). DOI:10.1145/882262.882269
4. Edwin H. Land, The retinex. *American Scientist* 52(2): 247-64. (1964).

Figure 7.1: Online published version of "*Retinex Poisson Equation: a Model for Color Perception*" (first page).

7.1.1 Overview

In 1964 Edwin H. Land (ref. 4) formulated the Retinex theory, the first attempt to simulate and explain how the human visual system perceives color. His theory and an extension, the "reset Retinex" (ref. 5) were further formalized by Land and McCann in 1971. Several Retinex algorithms have been developed ever since. These color constancy algorithms modify the RGB values at each pixel to give an estimate of the physical color independent of the shading.

The Retinex original method was complex and imprecise. Indeed, this algorithm computes at each pixel an average of a very large and unspecified set of paths on the image. For this reason, Retinex has received several interpretations and implementations which, among other aims, attempt to tune down its excessive complexity.

But, as shown in ref. 1, the original Retinex algorithm can be formalized as a (discrete) partial differential equation. More precisely, it can be shown that if the Retinex paths are interpreted as symmetric random walks, then Retinex is equivalent to a Neumann problem for a Poisson equation. This result gives a fast algorithm involving just one parameter, also present in the original theory.

The Retinex Poisson equation (given below) is very similar to Horn's (ref. 6) and Blake's (ref. 7) equations, which were proposed as alternatives to Retinex. It also is one of the "Poisson editing" equations proposed in Perez *et al.* (ref. 3). The final principle of the algorithm is extremely simple. Given a color image I, its small gradients (those with magnitude lower than a threshold t) in each channel are replaced by zero. The resulting vector field is no more the gradient of a function, but the Poisson equation reconstructs an image whose gradient is close for the quadratic distance t to this vector field. Thus, a new image is obtained, where small details and shades of the original have been eliminated. The elimination of the shades creates more homogeneous colors. This fact, according to Land and McCann, models the property of our perception to perceive constant colors regardless of their shading.

The formalization proved in ref. 1 yields a fast implementation of the Land-McCann original theory using only two DFT's. You can test the theory on line on your own color images[1].

7.1.2 References

1. Jean-Michel Morel, Ana Belén Petro and Catalina Sbert. A PDE Formalization of the Retinex Theory. *IEEE Transactions on Image Processing*, 2010.
   DOI: `http://dx.doi.org/10.1109/TIP.2010.2049239`.
2. Jean-Michel Morel, Ana Belén Petro and Catalina Sbert. Fast Implementation of color constancy algorithms. *Color Imaging XIV: Displaying, Processing, Hardcopy and Application*, Proceedings of Electronic Imaging SPIE, volume 7241, 2009.
   DOI: `http://dx.doi.org/10.1117/12.805474`.
3. P. Pérez, M. Gangnet and A. Blake. Poisson Image Editing. *ACM Transactions on Image Processing*, Proceedings of ACM SIGGRAPH 2003, volume 22, issue 3 pages 313 - 318, 2003.
   DOI: `http://doi.acm.org/10.1145/882262.882269`

[1] `http://www.ipol.im/pub/demo/lmps_retinex_poisson_equation/`

4. Edwin H. Land. The retinex. *American Scientist* 52(2): 247–64, 1964.
5. Edwin H. Land and John J. McCann. Lightness and Retinex Theory. *Journal of the Optical Society of America* 61, 1–11, 1971.
6. Berthold K. Horn, Determining lightness from an image. *Computer Graphics and Image Processing* 3, 277–299, 1974.
DOI: http://dx.doi.org/10.1016/0146-664X(74)90022-7.
7. Andrew Blake. Boundary conditions of lightness computation in Mondrian world. *Computer Vision Graphics and Image Processing* 32, 314–327, 1985.
DOI: http://dx.doi.org/10.1016/0734-189X(85)90054-4.
8. Marcelo Bertalmio, Vicent Caselles, Edoardo Provenzi and Alessandro Rizzi. Perceptual Color Correction Through Variational Techniques. *IEEE Transactions on Image Processing* volume 16(4), 1058–1072, 2007.
DOI: http://dx.doi.org/10.1109/TIP.2007.891777.

7.1.3 Online Demo

An online demo[2] allows you to try Retinex with your own images. The demo has only one parameter, the contrast threshold t present in the original theory.

The uploaded images will be converted to color PNG format and may be resized for an efficient Fourier transform. The images dimensions are kept under 1024 and adjusted to the nearest multiple of 2, 3, 5 and 7 to avoid large primes, and the image is resampled using a cubic spline interpolation. The original non resized images are kept and available in the demo archive. This pre-processing is not in the implementation, it is only added to the demo to ensure fast results.

The aim of the Retinex algorithm is to simulate and explain how the human visual system perceives color, *it is not to improve the image quality.* In the last decade, the "Retinex" trademark has been extended to many color contrast algorithms which actually deviate from the initial Retinex scope. These algorithms successfully enhance the local image contrast and also perform a color balance. See for example (ref. 8) for a fast color enhancement algorithms. You can use the "Simplest Color Balance"[3] algorithm previously to Retinex, for improving the contrast of your image.

The result image, after the Retinex algorithm, is normalized using the mean and the variance of the original image. Thus, the output image and the original image have the same mean and variance and therefore the same global contrast.

7.1.4 The PDE-Retinex Model

In ref. 1 it is proven that the output of the Retinex algorithm proposed by Land and McCann is the solution of the discrete partial differential equation with Neumann boundary conditions

$$-\Delta_d u(i,j) = F(i,j)$$

where

[2] http://www.ipol.im/pub/demo/lmps_retinex_poisson_equation/
[3] http://dx.doi.org/10.5201/ipol.2011.llmps-scb

$$\Delta_d u(i,j) = u(i+1,j) + u(i-1,j) + u(i,j+1) + u(i,j-1) - 4u(i,j)$$

is the discrete Laplacian,

$$\begin{aligned} F(i,j) &= f(I(i,j) - I(i+1,j)) + f(I(i,j) - I(i-1,j)) \\ &\quad + f(I(i,j) - I(i,j+1)) + f(I(i,j) - I(i,j-1)) \end{aligned}$$

and $f(x)$ is a threshold function, whose value is zero if $|x|| < t$ and the identity in other case and I is the image to process. This function f eliminates the small variations of the intensity image I.

The parameter t (the threshold) is by default $t = 4$ but you can choose the value depending of the variations you want to eliminate. When I is a gray level image, the algorithm applies to I. When I is a color image, the algorithm is applied to each scalar channel separately.

7.1.5 The Algorithm

The output of the algorithm is an image which is the result of the Retinex PDE applied separately to the three channels of the color image, completed by a normalization using the mean and the variance of the original image.

The discrete partial differential equation is easily solved by Fourier transform. To enforce the Neumann boundary condition, the image is first mirrored across its right and bottom sides to obtain an image four times larger, which is symmetric with respect to its vertical and horizontal medial axes.

The discrete Fourier transform of a two-dimensional function $u(n,m)$ defined on a $N \times M$ grid is defined for (k,l) in $\{0,\ldots,M-1\} \times \{0,\ldots,N-1\}$ by

$$\widehat{u}(k,l) = \frac{1}{NM} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} u(n,m) e^{-i\frac{2\pi kn}{N}} e^{-i\frac{2\pi lm}{M}}$$

and the discrete inverse Fourier transform for (m,n) in $\{0,\ldots,M-1\} \times \{0,\ldots,N-1\}$ by

$$u(n,m) = \sum_{k=0}^{N-1} \sum_{l=0}^{M-1} \widehat{u}(k,l) e^{i\frac{2\pi kn}{N}} e^{i\frac{2\pi lm}{M}}.$$

The discrete Fourier transform has the following property

$$u(n-n_0, m-m_0) = \sum_{k=0}^{N-1} \sum_{l=0}^{M-1} \widehat{g}(k,l) e^{i\frac{2\pi kn}{N}} e^{i\frac{2\pi lm}{M}},$$

where

$$\widehat{g}(k,l) = \widehat{u}(k,l) e^{-i\frac{2\pi kn_0}{N}} e^{-i\frac{2\pi lm_0}{M}}$$

Applying the discrete Fourier transform to the discrete Poisson equation and using this last property yields

$$\widehat{u}(k,l)\left(4-2\cos\frac{2\pi k}{N}-2\cos\frac{2\pi l}{M}\right)=\widehat{F}(k,l)$$

which entails

$$\widehat{u}(k,l)=\frac{\widehat{F}(k,l)}{4-2\cos\frac{2\pi k}{N}-2\cos\frac{2\pi l}{M}},\quad \text{for } (k,l)\neq(0,0)$$

Using the inverse Fourier transform we obtain the value of u at each point of the grid, defined up to a constant since the constant Fourier coefficient is arbitrary. The values of u are finally normalized and receive the mean and the variance of the original image. After this normalization some values may fall outside the interval $[0, 255]$. This values are saturated to 0 or 255.

All of the above computations are performed on the extended symmetric image F defined on the $2N \times 2M$ grid. F being symmetric, its Fourier coefficients are real. This property is transferred by the equation to the Fourier coefficients of u, and u is therefore also symmetric and verifies the Neumann boundary condition. All of these operations are performed for each channel of the color image, u being in turn the red, green and blue channel.

The algorithm (applied to each channel) therefore is

1. Compute $F(i, j)$;
2. Compute the Fourier transform of F by DFT (symmetrization is handled by the fftw library);
3. Deduce the Fourier transform of u using the formula above;
4. Compute the final solution u by the inverse DFT and apply the normalization.

7.1.6 Implementation

The `retinex_pde` implementation and documentation are available on the article web page[4].

It should compile on any system since it is only ANSI C. This implementation is used in the online demo[5].

This code requires `libpng`[6] for PNG file input/output and `libfftw3`[7] to process the Fourier transforms[8].

Compilation and usage instructions are provided in the `README.txt` file.

[4] `http://dx.doi,org/10.5201/ipol.2011.lmps_rpe`

[5] `http://www.ipol.im/pub/demo/lmps_retinex_poisson_equation/`

[6] `http://www.libpng.org/pub/png/libpng.html`

[7] `http://www.fftw.org/`

[8] Linux: you can install `libpng` and `libfftw3` with your package manager; Mac OS X: you can get `libpng` and `libfftw3` from the Fink project (`http://www.finkproject.org/`); Windows: precompiled DLLs are available online for `libfftw3` (`http://www.fftw.org/install/windows.html`) and `libpng` (`http://gnuwin32.sourceforge.net/packages/libpng.htm`); note that `libpng` requires `zlib` (`http://gnuwin32.sourceforge.net/packages/zlib.htm`).

Implementation notes: The `fftw3 library` supports several Fourier transform types, in particular discrete Fourier transforms of input real data with even/odd symmetry (i.e. cosine/ sine transform). With this kind of mirror symmetries across the boundary there is no need for complex input/output. Moreover, one gains a factor of two in computational time and space. Because of the discrete sampling, this library permits to choose the type of symmetry. The mirror symmetry can be alternatively made with respect to the boundary sample points, or with respect to the points obtained by shifting a half pixel toward the exterior the boundary samples. In our implementation we use this second symmetry, which duplicates exactly the image size. After this mirror symmetry the cosine transform implements our equation.

Note from the editor: The source code and its history are available online in a version control browser[9]. The "IPOL" tag is attached to the version published in IPOL. Future improvements will be available there.

7.1.7 Examples

Here are some examples. Note that Retinex is not a model conceived for image enhancement or image improvement; it is only an algorithm to mimic our color perception. Thus, Retinex will enhance an effect that our perception does anyway.

The role of the t threshold is to eliminate the small intensity variations due to shading. Thus t cannot be too large (less than 10 typically) to avoid removing significant details. But is must be large enough to eliminate light shading effects. This is not always possible, since shadows can be very contrasted.

Adelson's Checker

A first classic example (figure 7.2) shows the effect of Retinex on the Adelson's checker shadow illusion. In the left image a green cylinder standing on a black and white checkerboard casts a diagonal shadow across the board.

The image has been so constructed that the white squares in the shadow, one of which is labeled "B," have actually the very same gray value as the black squares outside the shadow, one of which is labeled "A."

If Retinex is faithful to human perception, it should make B much brighter than A, and it does.

Circles on a Gradient

Simultaneous contrast is a name for the fact that the appearance of a color depends on the colors surrounding it. The original image (figure 7.3) shows a background with a smooth, but intense, variation and two circles with the same gray value (170). One of them is placed in the darker part of the image, and the other one in the brighter part.

The usual perception is that the circle in the darker part looks conspicuously brighter than the other. If we use a threshold $t = 3$ larger than the background variation, the result is an image with nearly constant background (from 105 to 140). The left circle gets a 0 gray value and the right circle a 255 gray value, which predicts well our perception.

[9] `http://dev.ipol.im/git/?p=nil/retinex_pde.git`

original image
(A and B have a gray value of 120)

Retinex result with $t = 3$
(A has a gray value of 120 and B of 160)

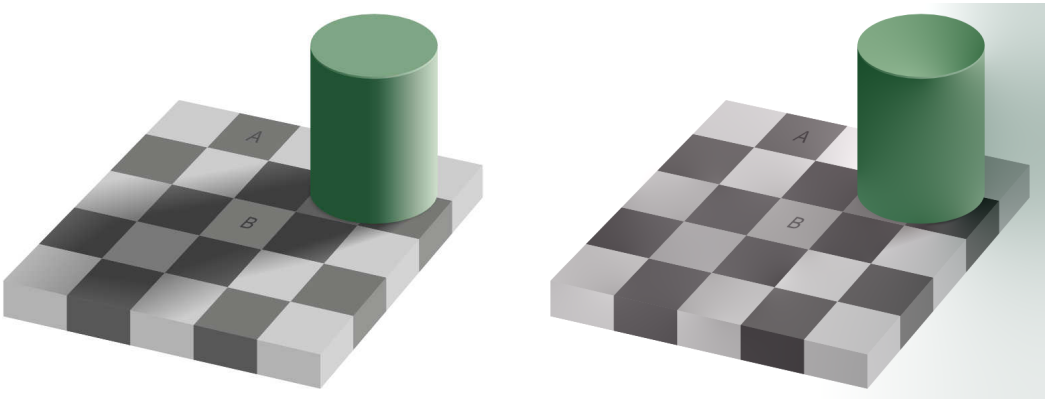


Figure 7.2: Adelson's checker.

original image, white balanced

Retinex result

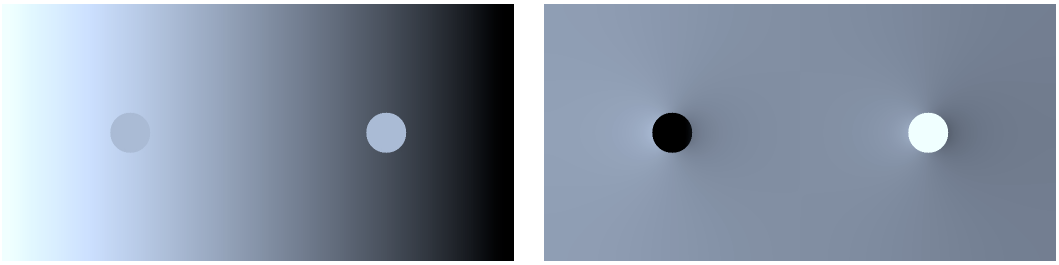


Figure 7.3: Circles on a gradient.

Noisy Image

To understand the effect of the threshold t the figure 7.4 shows a noisy original and the result of Retinex with increasing threshold values $t = 1$, $t = 3$ and $t = 5$.

The background clutter and the shades are progressively filtered out when t increases, but the main edges are kept. At $t = 5$, however, edges start loosing contrast and low contrasted details start disappearing.

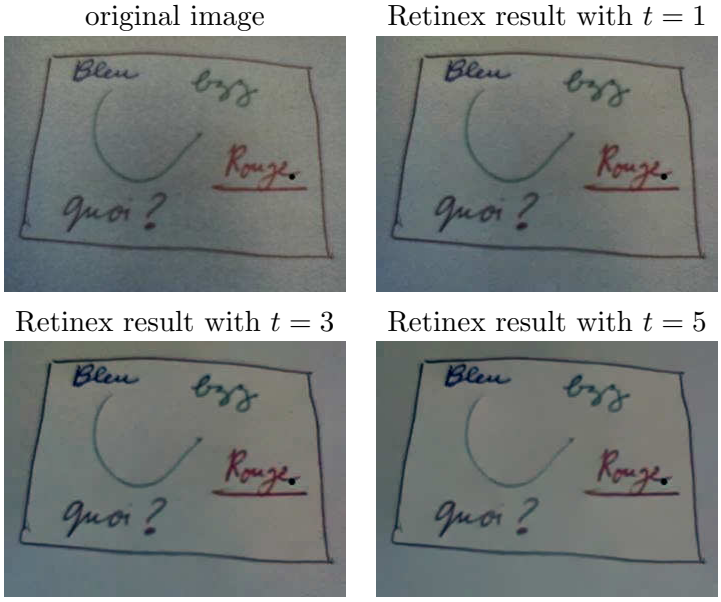


Figure 7.4: Noisy image.

As we have mentioned before, the Retinex method is not a model for image enhancement or image improvement. We could apply other methods for image enhancement, previously to Retinex, to obtain better results. For example, to the previous image, we can apply the "Simplest Color Balance"[ˆscp] algorithm and we can observe the better results (figure 7.5).

Shadows Removal

The figures 7.6 and 7.7 demonstrate how Retinex can be used for removing shadows. This is not always effective, but here are two good examples. The shadow removal works only if the boundary of the shadow is blurry, and therefore has a small gradient.

Lena

Application to Lena (figure 7.8): the smooth shading variations on the shoulder or the face and in the background fade out when the threshold t increases.

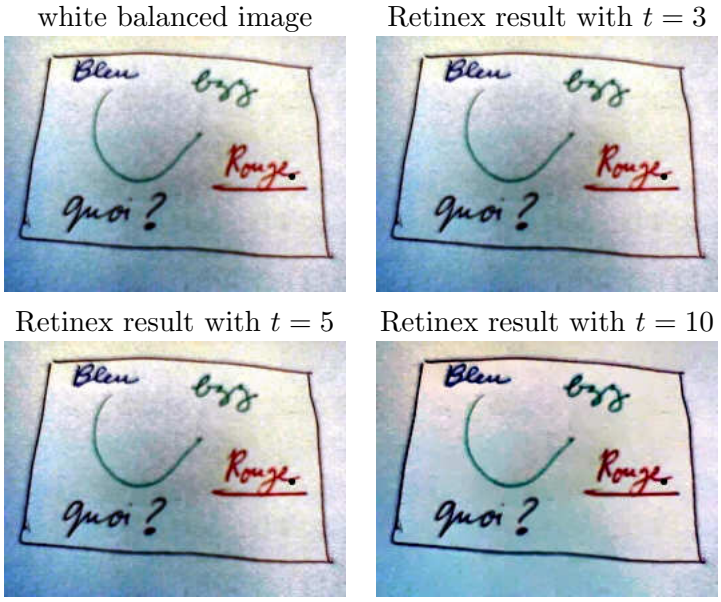


Figure 7.5: Noisy image preprocessed with "*Simplest color Balance*" before applying Retinex.

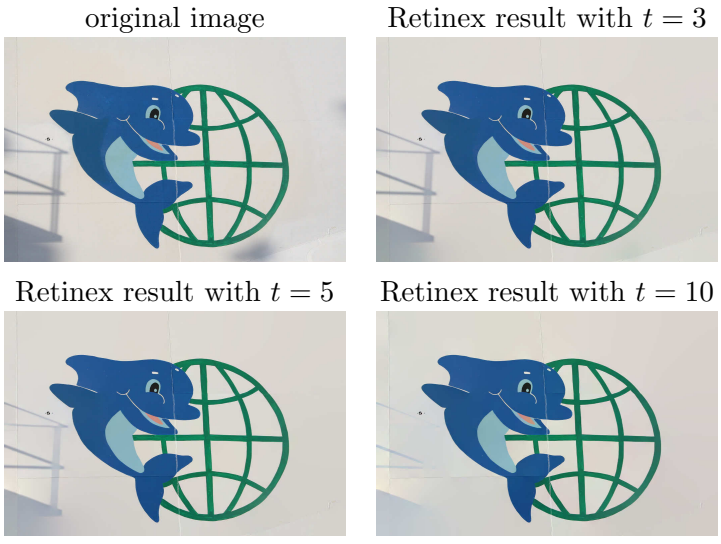


Figure 7.6: Shadows removal

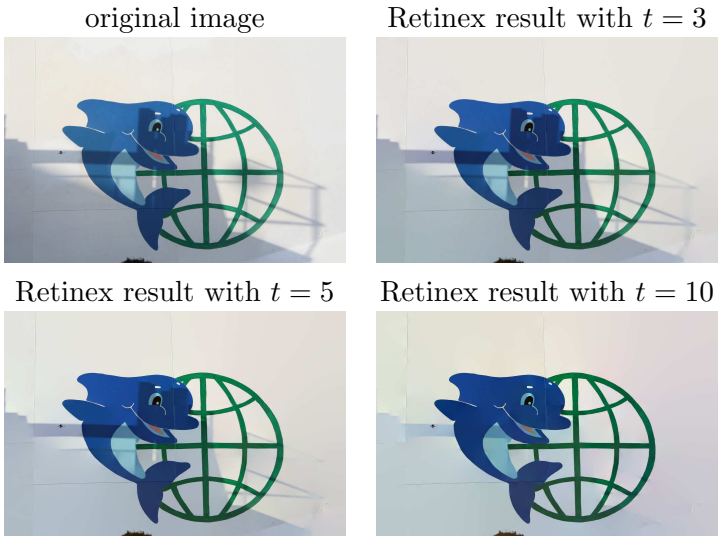


Figure 7.7: Shadows removal.

Figure 7.8: Lena.

Limitations of the method

The Retinex method has been used by several authors as a model for image enhancement or image improvement. But this apparant improvement when it occurs is simply due to the image normalization or the color restoration applied after having applied the Retinex algorithm. Thus the very same contrast improvement can be obtained by a simple color balance algorithm. To avoid mixing up any color balance effect with the Retinex effect, the algorithm always applies a normalization keeping for the Retinex result the mean and the variance of the original image. Retinex should *never* improve the contrast of the image; its perpetual effect is "color constancy": it simply flattens the color in low gradient regions. That's all.

Thus, dark images or bad quality images should not improve with the mere application of the Retinex method. An example of these effects can be observed in the figure 7.9. The application of Retinex only "flattens" the image. On the other hand, the simplest color balance algorithm yields a significant improvement of the image.

Figure 7.9: comparison with "*Simplest Color Balance*".

The figure 7.10 is another example with a dark image. There is no significant difference between the original image and the Retinex result. The result of the simplest color balance algorithm instead is a serious improvement of the image quality.

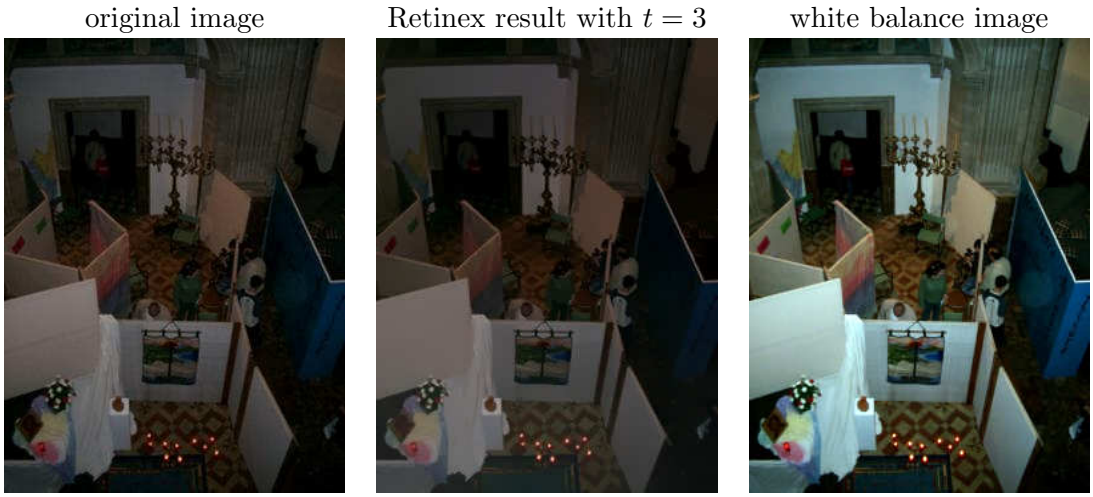


Figure 7.10: Effect on a dark image.

Since it only alters small non zero gradients, the Retinex algorithm does not produce any

change in images having only completely flat areas. The figure 7.11 shows this phenomenon on a synthetic image. Observe that the original image is identical to the Retinex result.

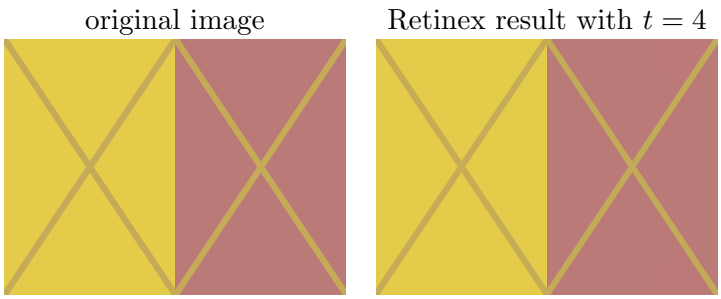


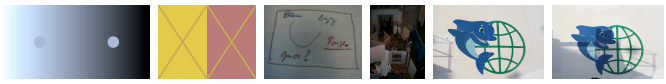
Figure 7.11: Effect on a synthetic image with flat areas.

7.1.8 Acknowledgment

The authors thank the referees, Gabriele Facciolo and Vicent Caselles, for their very valuable corrections and comments.

7.1.9 Credits


Edward H. Adelson[10]


the authors, CC-BY


courtesy Philip Greenspun[11]


standard test image

7.1.10 Software and Demo Design

These additional sections about the implementation design and the online demonstration were not in the online version.

Implementation

External Libraries The "*Retinex Poisson Equation*" algorithm was first implemented while we were exploring the idea of a web-based image processing interface, which even-

[10] `http://web.mit.edu/persci/people/adelson/checkershadow_illusion.html`
[11] `http://philip.greenspun.com/`

tually became a part of IPOL. One critical condition of the feasibility of these web experiments is the execution time: how long does the algorithm take, and how large the data can be while still being processed in less than 30 seconds? These questions led to a performance profiling of the code and, unsurprisingly, the computation time was dominated by the discrete Fourier transforms.

Early versions of the code were based on the MegaWave framework [136] and its `fft2d()`[12] implementation of the discrete Fourier transform on images. A short research for a high-performance alternative, free, stale, usable with a C code, hinted at the FFTW[13] library as the leading portable, free and fast implementation. Its performance was compared[14] with the MegaWave code and the measurements, visible in figure 7.12, confirmed that the FFTW library was significantly faster.

This is naturally explained by the research and efforts devoted to build a library like FFTW: different algorithms are available and selected depending on the array properties; some numerical values are precomputed for usual input sizes; the code is optimized to the lowest level, with provisions for the characteristics of various families and models of processors, data locality, vector instructions and cache efficiency; and a code generation tool is used to compose the high-level library from many special-purpose low-level code fragments, each highly optimized for one array size, DFT algorithm or hardware capacity [135]. In contrast, the MegaWave code is a straightforward implementation of the classic Cooley-Tuckey FFT algorithm.

The lesson is clear: when the performance of an algorithm depends on some classic numerical components, fast programs require specialized software tools instead of simple, straightforward implementations. Another advantages of specialized libraries, not illustrated in this Retinex example, is that they usually are more tested, better maintained and have better numerical stability and accuracy than custom code.

To achieve a better performance, the Retinex implementation was rewritten out of the MegaWave environment, with the FFTW library. For a complete implementation, we needed the image file input/output layer, previously handle internally by MegaWave. This was done by using PNG images and the `libpng` library, and this layer evolved later into the `io_png` light interface to `libpng`, now proposed to the IPOL authors as an easy way to use PNG image files.

Code Design The implementation of this algorithm was an opportunity to show how a code could be designed to facilitate review and reuse. There is no innovation there, but these standard design rules are too often ignored in research code. The `retinex_pde` program is modest, with only 600 lines of code split into four files described hereafter. The function calls and file dependencies are represented in figure 7.13.

- `retinex_pde_lib.c` contains the actual implementation of the algorithm, isolated in a single source code file reusable in other software projects. This file is the one the reviewers should focus on, because it contains the software implementation of the algorithm described in the IPOL article. Only one entity is exported from

[12] `http://megawave.cmla.ens-cachan.fr/stuff/guid3/node165.html`

[13] `http://fftw.org/`

[14] For every size from 128 to 1024, a two-dimension array was filled with random single-precision floating-point values and transformed by discrete Fourier transform. This operation was repeated 10 times, without parallel processing options, on an Intel Core Duo L2300 processor (2M Cache, 1.50 GHz).

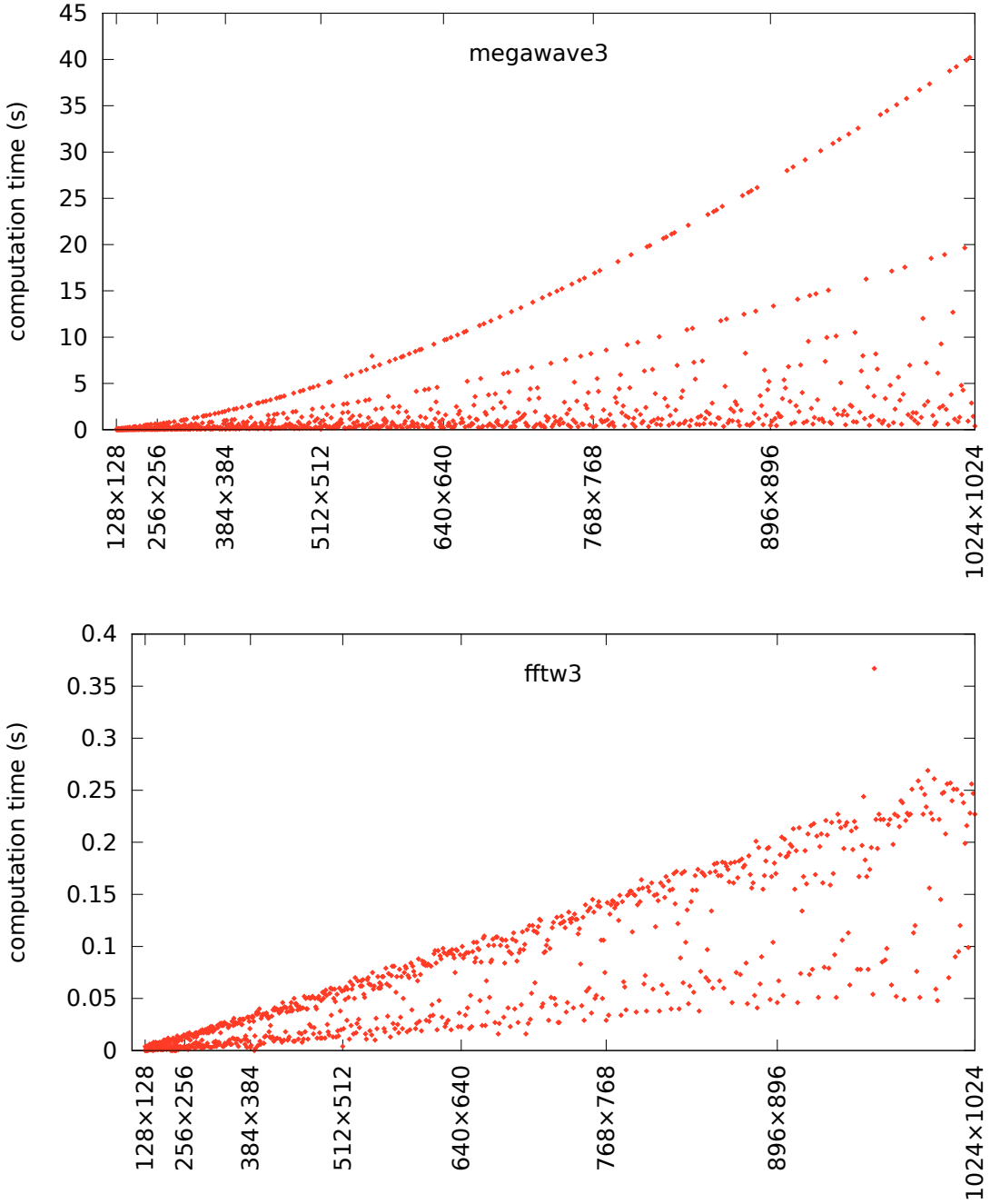


Figure 7.12: Compared performances of two implementations of the discrete Fourier transform. The vertical scales are different.

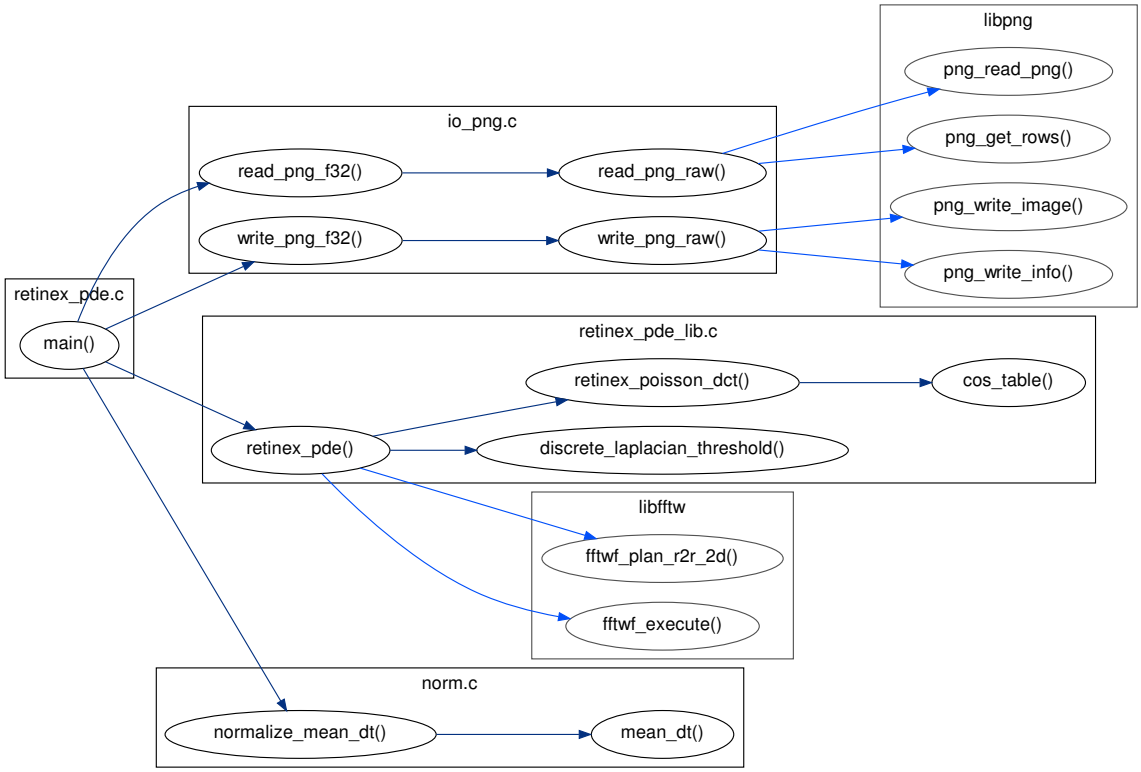


Figure 7.13: Function calls and file dependencies for the "*Retinex Poisson Equation*" implementation. Minor functions have been omitted for clarity.

the compilation of this file: the `retinex_pde()` function, used to apply the Retinex transform to an image. Retinex makes sense for images but is described in the article in terms of Fourier transforms, Laplacian and Poisson equation, which make sense for any 2D array. So the `retinex_pde()` function is generic and processes numeric arrays, without any image model or custom structure. The function either succeeds, and returns a pointer to the transformed array, or fails and returns an invalid pointer that can be detected from the calling context. This function uses `libfftw`, but the Fourier transform part of the code could be modified to use another library without impacting the call syntax of the function.

- `norm.c` implements the post-processing normalization step. Here again, this file is self-sufficient. One function computes the mean and covariance of an array and another function normalizes an array; these functions are expressed in term of one-dimension vectors as they don't need the concepts of images or even two dimension arrays.

- `retinex_pde.c` contains the `main()` function and is responsible for all the tasks only relevant to the execution environment: processing the command-line parameters, reading the input file and writing the output; calling the Retinex function and the post-processing normalization function. This is the only source code file with assumptions about how the program will be used, and as such this code is not reusable and not expected to be reused.

- `io_png.c` has the simplified interface to `libpng`, with routines to read image files into arrays or writes them with a single function call. It lets the programmer choose which kind of image is to be processed (grayscale or color, integer or floating-point) and internally converts the PNG image to the expected format.

Optimization Performance profiling of the program with Valgrind[15] revealed that most of the computation time was devoted to the Fourier transforms. The speed of the program was improved by using the "advanced" FFTW interface to compute directly the cosine transform from real arrays instead of obtaining this result from complex values after a symmetry. This saves some CPU instructions, memory space, data transfer and cache misses, and leads to a faster execution.

The other optimizations, applied to the Laplacian computation and Poisson transform in the Fourier space, were deliberately kept simple to preserve the portability and readability of the source code. These two functions consist in applying a local operation to every component of a 2D array. We tried to avoid redundant costly operations like the computation of the sine and cosine factors, to keep reusable values in memory, to use simpler operations (multiplications by $1/x$ instead of divisions by x), to limit cache miss by processing the arrays in their memory order, and reduce the number of temporary variables.

Two other optimizations are possible but were not carried out in the published version of the program because the performances were already sufficient. One would be to ease the use of SIMD instructions by the automatic vectorization features of the compilers. The other one would be to leverage the CPU pipelining efficiency by avoiding branching instructions in the loops; this may be achieved in the Laplacian computation by using

[15] Valgrind (`http://valgrind.org/`) is a collection of dynamic code analysis tools, including a memory error detector.

bit-twiddling instructions instead of `if` branches and by handling the array border out of the main loop.

Quality Control The program is a standard C89 source code. It only needs a C compiler and the two aforementioned external libraries and should be compilable and usable in any computing environment. The standard compliance was tested by using strict compilation options[16] and the source code quality was further checked by static code analysis with Splint[17]. These tests do not guarantee the quality of the code, but they can be used to detect and fix known dangerous patterns and abuses of the language.

After the publication of this code in IPOL[18], these tests were further improved with the usage of other static code analysis tools (Clang and Cppcheck[19]) and by compiling the program with various C and C++ compilers[20] and fixing the compiler warnings. Finally, functional tests were added to verify that the exact same results are obtained with all these compilers and all the successive versions of the code.

This quality control is routinely performed via tests integrated in the build procedure, and automatically performed every time a new version is recorded:

- the code is built by the default C compiler, with cross-compiler compatible options and the possibility to specify the compiler at build time;
- the pseudo-target `lint` checks all the source code with static checking tools and compiles it with strict compiler options;
- the pseudo-target `beautify` maintains the indentation by processing all the source code with a code reformatting tool;
- the pseudo-target `test` builds the program with various compilers, runs functional tests and executes the program in a dynamic memory checking environment.

Demonstration

Workflow The demonstration of this algorithm proposed in IPOL [301] follows a basic workflow, illustrated in the chart 7.14 and screen captures 7.15.

- The start page proposes a few input images for the algorithm. With these images, demo users can reproduce the results published in the article. They can also upload their own images to try the algorithm.
- On the parameter page, one can choose the threshold to be used, and launch the algorithm.
- The result page is displayed after a few seconds, with the output of the algorithm. Then one chooses to restart the demo with another threshold parameter, or an new input image.

[16] The basic set of compilation options was `gcc -ansi -pedantic -Wall -Wextra -Werror`.

[17] Splint (`http://www.splint.org/`) is a static code analysis tool.

[18] `http://dev.ipol.im/git/?p=nil/retinex_pde.git`

[19] The Clang compiler (`http://clang.llvm.org/`) has static analysis functions. Cppcheck (`http://cppcheck.sourceforge.net/`) is another code analysis tool for C and C++ programs.

[20] We use the following compilers: GNU `gcc` and `mingw`, LLVM `clang`, Intel `icc`, Sun `suncc`, `tcc`, and `nwcc`.

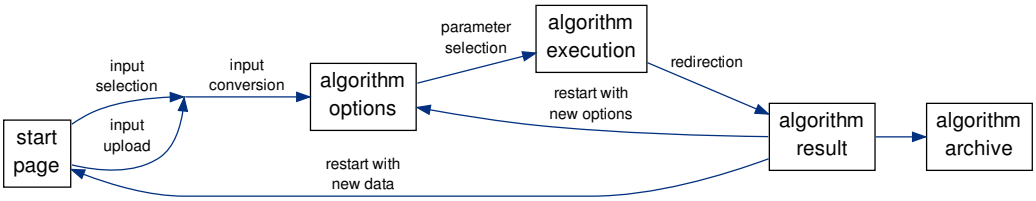


Figure 7.14: Typical IPOL demo flow chart, applicable to the "*Retinex Poisson Equation*" case.

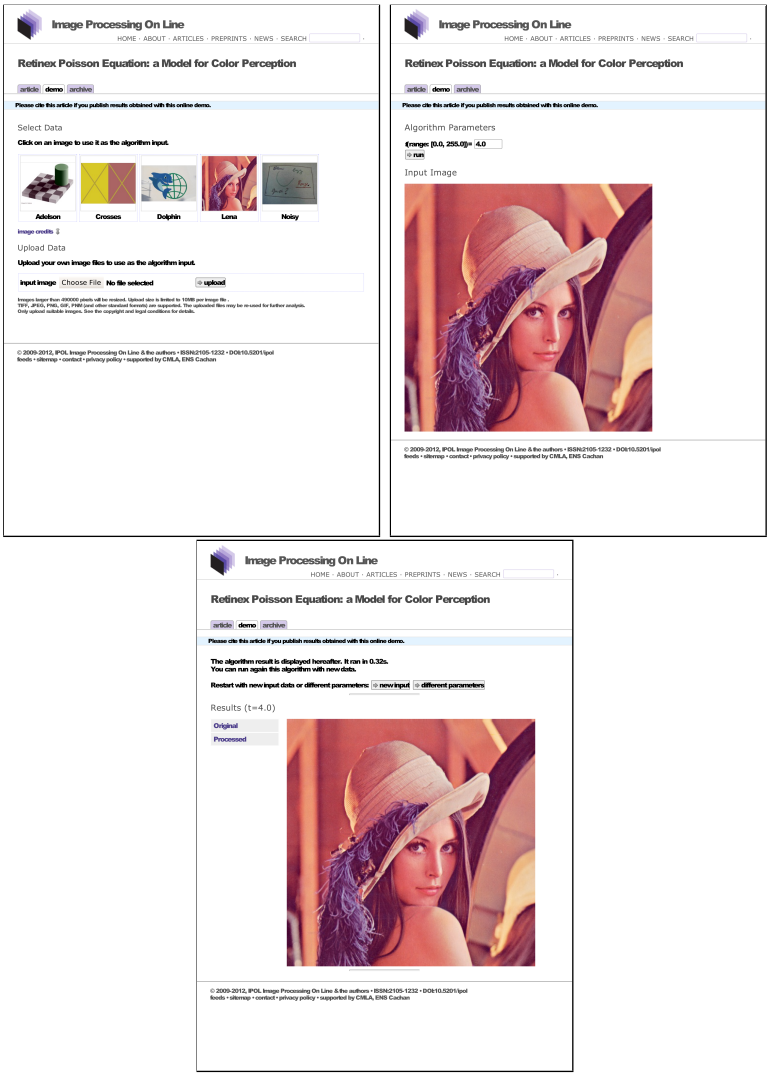


Figure 7.15: Input selection, parameters and result of the IPOL demo for "*Retinex Poisson Equation: a Model for Color Perception*".

Input Filtering To keep the demo fast and avoid too large images difficult to display properly in a web interface, the input is limited to a maximum width or height of 1024 pixels, and larger images are zoomed out before processing.

But the performance of the algorithm depends on the speed of the Fourier transforms, which are faster when the size of the image is divisible by low prime numbers. For example, in the conditions of the speed performance comparison already mentioned, the Fourier transform of a 509×509 array takes 38 ms, while a 512×512 array only takes 4 ms. The demo avoids large prime numbers by resizing the input images to the closest multiple of 2, 3, 5 and 7. This preprocessing changes the image size by 3%, and is acceptable for this algorithm because it preserves the important input properties, lightness and contrast. It would not be possible for an algorithm focusing on the pixel level, such as denoising methods.

One must note that, as explained in the article, this preprocessing step is built in the online demo code layer, not in the compiled implementation of the algorithm. This is important because the implementation of the algorithm must be reusable in a processing chain, and as such only the operations described as the algorithm must be performed. The demo could probably be improved by making this resizing optional for users who prefer to obtain the exact result of the algorithm even if it is slower.

Usage From its first public release on April 27th, 2009, to December 31st, 2011, this demo has been used more than 2000 times with original input images; 1500 of these experiments are publicly available in the archive[21]. Based on recent statistics, we can estimate that the algorithm has been tested more than 5500 times by IPOL visitors and collaborators with this demonstration interface during the same period.

7.2 Simplest Color Balance

7.2.1 References

1. Wikipedia contributors. Color Balance. *Wikipedia, The Free Encyclopedia* (accessed January 14, 2010).
   URL: `http://en.wikipedia.org/w/index.php?title=Color_balance&oldid=336334367`.

2. Marc Ebner. *Color Constancy*, p. 104, 2007. John Wiley & Sons.
   DOI: `http://dx.doi.org/10.1002/9780470510490`.

7.2.2 Overview

Color balance algorithms attempt to correct underexposed images, or images taken in artificial lights or special natural lights, such as sunset.

There are many sophisticated algorithms in the literature performing color balance or other color contrast adjustments. The performance of these many color correction algorithms can be evaluated by comparing their result to the simplest possible color balance algorithm proposed here. The assumption underlying this algorithm is that the highest values of R,

[21] `http://www.ipol.im/pub/demo/lmps_retinex_poisson_equation/`

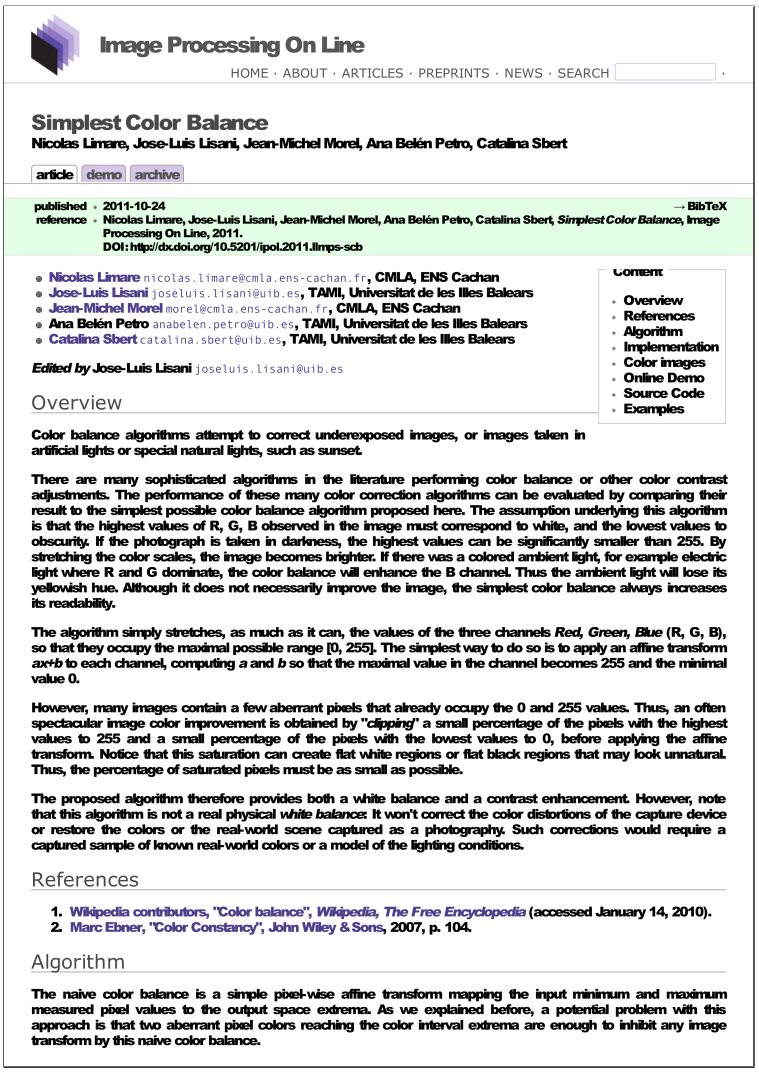


Image Processing On Line

HOME · ABOUT · ARTICLES · PREPRINTS · NEWS · SEARCH

Simplest Color Balance

Nicolas Limare, Jose-Luis Lisani, Jean-Michel Morel, Ana Belén Petro, Catalina Sbert

article demo archive

published • 2011-10-24 → BibTeX
reference • Nicolas Limare, Jose-Luis Lisani, Jean-Michel Morel, Ana Belén Petro, Catalina Sbert, *Simplest Color Balance*, Image Processing On Line, 2011.
DOI : http://dx.doi.org/10.5201/ipol.2011.llmps-scb

- Nicolas Limare nicolas.limare@cmla.ens-cachan.fr, CMLA, ENS Cachan
- Jose-Luis Lisani joseluis.lisani@uib.es, TAMI, Universitat de les Illes Balears
- Jean-Michel Morel morel@cmla.ens-cachan.fr, CMLA, ENS Cachan
- Ana Belén Petro anabelen.petro@uib.es, TAMI, Universitat de les Illes Balears
- Catalina Sbert catalina.sbert@uib.es, TAMI, Universitat de les Illes Balears

Edited by Jose-Luis Lisani joseluis.lisani@uib.es

Content
- Overview
- References
- Algorithm
- Implementation
- Color images
- Online Demo
- Source Code
- Examples

Overview

Color balance algorithms attempt to correct underexposed images, or images taken in artificial lights or special natural lights, such as sunset.

There are many sophisticated algorithms in the literature performing color balance or other color contrast adjustments. The performance of these many color correction algorithms can be evaluated by comparing their result to the simplest possible color balance algorithm proposed here. The assumption underlying this algorithm is that the highest values of R, G, B observed in the image must correspond to white, and the lowest values to obscurity. If the photograph is taken in darkness, the highest values can be significantly smaller than 255. By stretching the color scales, the image becomes brighter. If there was a colored ambient light, for example electric light where R and G dominate, the color balance will enhance the B channel. Thus the ambient light will lose its yellowish hue. Although it does not necessarily improve the image, the simplest color balance always increases its readability.

The algorithm simply stretches, as much as it can, the values of the three channels *Red, Green, Blue* (R, G, B), so that they occupy the maximal possible range [0, 255]. The simplest way to do so is to apply an affine transform $ax+b$ to each channel, computing a and b so that the maximal value in the channel becomes 255 and the minimal value 0.

However, many images contain a few aberrant pixels that already occupy the 0 and 255 values. Thus, an often spectacular image color improvement is obtained by "*clipping*" a small percentage of the pixels with the highest values to 255 and a small percentage of the pixels with the lowest values to 0, before applying the affine transform. Notice that this saturation can create flat white regions or flat black regions that may look unnatural. Thus, the percentage of saturated pixels must be as small as possible.

The proposed algorithm therefore provides both a white balance and a contrast enhancement. However, note that this algorithm is not a real physical *white balance*: It won't correct the color distortions of the capture device or restore the colors or the real-world scene captured as a photography. Such corrections would require a captured sample of known real-world colors or a model of the lighting conditions.

References

1. Wikipedia contributors, "Color balance", *Wikipedia, The Free Encyclopedia* (accessed January 14, 2010).
2. Marc Ebner, "Color Constancy", John Wiley & Sons, 2007, p. 104.

Algorithm

The naive color balance is a simple pixel-wise affine transform mapping the input minimum and maximum measured pixel values to the output space extrema. As we explained before, a potential problem with this approach is that two aberrant pixel colors reaching the color interval extrema are enough to inhibit any image transform by this naive color balance.

Figure 7.16: Online published version of "*Simplest Color Balance*" (first page).

G, B observed in the image must correspond to white, and the lowest values to obscurity. If the photograph is taken in darkness, the highest values can be significantly smaller than 255. By stretching the color scales, the image becomes brighter. If there was a colored ambient light, for example electric light where R and G dominate, the color balance will enhance the B channel. Thus the ambient light will lose its yellowish hue. Although it does not necessarily improve the image, the simplest color balance always increases its readability.

The algorithm simply stretches, as much as it can, the values of the three channels *Red, Green, Blue* (R, G, B), so that they occupy the maximal possible range [0, 255]. The simplest way to do so is to apply an affine transform $ax + b$ to each channel, computing a and b so that the maximal value in the channel becomes 255 and the minimal value 0.

However, many images contain a few aberrant pixels that already occupy the 0 and 255 values. Thus, an often spectacular image color improvement is obtained by "*clipping*" a small percentage of the pixels with the highest values to 255 and a small percentage of the pixels with the lowest values to 0, before applying the affine transform. Notice that this saturation can create flat white regions or flat black regions that may look unnatural. Thus, the percentage of saturated pixels must be as small as possible.

The proposed algorithm therefore provides both a white balance and a contrast enhancement. However, note that this algorithm is not a real physical *white balance*: It won't correct the color distortions of the capture device or restore the colors or the real-world scene captured as a photography. Such corrections would require a captured sample of known real-world colors or a model of the lighting conditions.

7.2.3 References

1. Wikipedia contributors.
   Color Balance.
   Wikipedia, The Free Encyclopedia (accessed January 14, 2010)
   URL: `http://en.wikipedia.org/w/index.php?title=Color_balance&oldid=336334367`

2. Marc Ebner.
   Color Constancy, p. 104, 2007.
   John Wiley & Sons
   DOI: `http://dx.doi.org/10.1002/9780470510490`

7.2.4 Algorithm

The naive color balance is a simple pixel-wise affine transform mapping the input minimum and maximum measured pixel values to the output space extrema. As we explained before, a potential problem with this approach is that two aberrant pixel colors reaching the color interval extrema are enough to inhibit any image transform by this naive color balance.

A more robust approach consists in mapping two values V_{min} and V_{max} to the output space extrema, V_{min} and V_{max} being defined so that a small user-defined proportion of the pixels get values out of the $[V_{min}, V_{max}]$ interval.

7.2.5 Implementation

Our input image is an array of N numeric values in the $[min, max]$ interval. The output is a corrected array of the N updated numeric values. Multiple channel images are processed independently on each channel with the same method.

We will perform a color balance on this data where we have saturated a percentage $s_{min}\%$ of the pixels on the left side of the histogram, and a percentage $s_{max}\%$ of pixels on the right side; for example, $s_{min} = 0$ and $s_{max} = 3$ means that this balance will saturate no pixels at the beginning and will saturate at most $N \times 3/100$ at the end of the histogram. We can't ensure that *exactly* $N \times (s_{min} + s_{max})/100$ pixels are saturated because the pixel value distribution is discrete.

Sorting Method

V_{min} and V_{max}, the saturation extrema, can be seen as quantiles of the pixel values distribution, *e.g.* first and 99th centiles for a 2% saturation.

Thus, an easy way to compute V_{min} and V_{max} is to sort the pixel values, and pick the quantiles from the sorted array. This algorithm would be described as follow:

1. **sort the pixel values**
   The original values must be kept for further transformation by the bounded affine function, so the N pixels must first be copied before sorting.

2. **pick the quantiles from the sorted pixels**
   With a saturation level $s = s_{min} + s_{max}$ in $[0, 100[$, we want to saturate $N \times s/100$ pixels, so V_{min} and V_{max} are taken from the sorted array at positions $N \times s_{min}/100$ and $N \times (1 - s_{max}/100) - 1$.

3. **saturate the pixels**
   According to the previous definitions of V_{min} and V_{max}, the number of pixels with values lower than V_{min} or higher than V_{max} is at most $N \times s/100$. The pixels (in the original unsorted array) are updated to V_{min} (resp. V_{max}) if their value is lower than V_{min} (resp. higher than V_{max}).

4. **affine transform**
   The image is scaled to $[min, max]$ with a transformation of the pixel values by the function f such that $f(x) = (x - V_{min}) \times (max - min)/(V_{max} - V_{min}) + min$. For 8-bit representations of an image, $min = 0$ and $max = 255$.

Histogram Method

Sorting the N pixel values requires $O(Nlog(N))$ operations and a temporary copy of these N pixels. A more efficient implementation is achieved by a histogram-based variant, faster ($O(N)$ complexity) and requiring less memory ($O(max - min)$ *vs.* $O(N)$).

1. **build a cumulative histogram of the pixel values**
   The cumulative histogram bucket labeled i contains the number of pixels with value lower or equal to i.

2. **pick the quantiles from the histogram**
   V_{min} is the lowest histogram label with a value higher than $N \times s_{min}/100$, and the number of pixels with values lower than V_{min} is at most $N \times s_{min}/100$. If $s_{min} = 0$ then V_{min} is the lowest histogram label, *i.e.* the minimum pixel value of the input image. V_{max} is the label immediately following the highest histogram label with a value lower than or equal to $N \times (1 - s_{max}/100)$, and the number of pixels with values higher than V_{max} is at most $N \times s_{max}/100$. If $s_{max} = 0$ then V_{max} is the highest histogram label, *i.e.* the maximum pixel value of the input image.

3. **saturate the pixels**

4. **affine transform**
   Same as for the sorting method.

Pseudo-code

The following steps are presented for images with pixel values in the 8-bit integer space ($min = 0$, $max = 255$) with one color channel only. See the following remarks for higher-precision image. The basic implementation is shown in figure 7.17, refinements are available in the proposed source code.

```
// build the cumulative histogram
for i from 0 to N-1
    // fill the histogram
    histo[image[i]] := histo[image[i]] + 1
for i from 1 to 255
    // convert to a cumulative histogram
    histo[i] := histo[i] + histo[i - 1]
// search vmin and vmax
vmin := 0
while histo[vmin + 1] <= N * smin / 100
    vmin := vmin + 1
vmax := 255 - 1
while histo[vmax - 1] > N * (1 - smax / 100)
    vmax := vmax - 1
if vmax < 255 - 1
    vmax := vmax + 1
// saturate the pixels
for i from 0 to N - 1
    if image[i] < vmin
        image[i] := vmin
    if image[i] > vmax
        image[i] := vmax
// rescale the pixels
for i from 0 to N-1
    image[i] := (image[i] - vmin) * 255 / (vmax - vmin)
```

Figure 7.17: Pseudo-code for 8-bit integer images. `image[i]` are the pixel values, `N` is the number of pixels, `histo` is an array of 256 unsigned integers, with a data type large enough to store `N`, initially filled with zeros. The arrays indexes start at 0.

Higher Precision

For 16-bit integer pixel values, the histogram array method can be used, and needs 65.536 buckets (256 Kb on a 32-bit system, 512 Kb on a 64-bit system, to be compared with the 128 Kb used for a 256×256 image). But the determination of `vmin` and `vmax` would benefit of a faster search method, like bisection.

For 32-bit integer pixel values, the histogram size (4.294.967.296 buckets) becomes a problem and can't be properly handled in memory. We can switch to a multi-step process:

- build a histogram with buckets containing more than one single pixel value, such that the histogram size is limited (256 buckets for example, each for a pixel value interval);
- search for the buckets *containing* `vmin` and `vmax`;
- restart the histogram construction and search on a subdivision of these buckets.

If an exact precision isn't required, the latest refinements can be skipped.

For floating-point data, the pixel value can no more be used as an array index, and we must use either a sorting method, or a multi-step method with an histogram containing intervals (not values), then a sorting method on the buckets *containing* `vmin` and `vmax`.

Note that the proposed pseudo-code can also be used for images with integer pixel values (as produced by common image capture devices and found in common image formats) stored as floating-point data (often desired for image processing), by converting the pixel value `image[i]` to its integer equivalent while filling the histogram.

Special Cases

If the image is constant (all pixels have the same value v), then, according to the described implementation and pseudo-code, the histogram values are 0 for labels lower than v, and N for labels higher or equal to v, and then for any value of s_{min} and s_{max} , $V_{min} = v$, $V_{max} = v$.

This ($V_{min} = V_{max}$) can also happen for non-constant image, the general case being images with less than $N \times s_{min}/100$ pixels with values below or with more than $N \times s_{max}/100$ above a median value v. This case can be handled by setting all the pixels to the value v.

7.2.6 Color images

RGB Color Balance

For RGB color images we can apply the algorithm independently on each channel. We call this algorithm *RGB color balance*. The color of the pixels is modified in the process because each RGB channel is transformed by an affine function with different parameters and the saturation does not occur on the three RGB channels together. This can be desirable do correct the color of a light source or filter, but in some applications we may want to maintain the colors of the input image.

In that case, many solutions are possible, depending on how we define the "color" to be maintained (hue, chroma, R/G/B ratio) and what we want to correct with this algorithm (lightness, brightness, intensity, luma, ...). A discussion about these color correction variants will be published in a later article, and we present hereafter the simplest version.

IRGB Intensity Balance

The goal of *IRGB intensity balance* is to correct the intensity of a color image without modifying the R/G/B ratio of the pixels. We first compute the gray level intensity ($I = (R + G + B)/3$), then this intensity is balanced and transformed into I' by the affine transformation with saturation. Finally, for each pixel, the three color channels are multiplied by I'/I.

But the RGB color cube is not stable by this transformation. Multiplied by I'/I, some RGB components will be larger than the maximum value. This is corrected in a post-processing step by a projection on the RGB cube while maintaining the R/G/B ratio, *ie* replacing pixels out of the RGB cube by the intersection of the RGB cube surface and the segment connecting the (0,0,0) point and the pixels to be corrected. This projection has three consequences :

- *commutativity* : computing the intensity I of an image after correction by this algorithm doesn't give the same result as computing the intensity of an image and correcting this intensity by the affine balance algorithm with saturation described at the beginning of this article;
- *monotonicity* : some pixels with intensities $I_1 < I_2$ can be be transformed into pixels with intensities $I_1 > I_2'$ if the pixel with initial intensity I_2 has to be corrected by projection;
- *precision* : because the projection step is darkening the projected pixels, less than $s_{max}\%$ of the pixels will have their final intensity saturated to the maximum value.

Moreover, adjusting the saturation on the average I of the three RGB channels means that, unless the three channels are equal (gray image), before the projection less than $s_{max}\%$ of the pixels are saturated to the maximum value on the three RGB channels while more than $s_{max}\%$ of the pixels are saturated to the maximum value on at least one of the RGB channels.

Better solutions to achieve a balance of a color image without these problems require the use of other color spaces and are beyond the scope of this article.

7.2.7 Online Demo

With the online demonstration[22], you can try this algorithm on your own images and set the desired percentage of saturated pixels. This demo presents the algorithm applied independently to the R, G and B channels (*RGB color balance*), and to the intensity channel while maintaining the R/G/B ratio (*IRGB intensity balance*).

For gray-scale images, these two versions are identical to applying the simple algorithm to the gray level.

[22] `http://www.ipol.im/pub/demo/lmps_simplest_color_balance/`

7.2.8 Source Code

An ANSI C implementation is provided and distributed under the GPL[23] license. Source code and documentation are available on the article web page[24].

Basic compilation and usage instructions are included in the `README.txt` file. This code requires the `libpng` library[25][26].

This source code includes two implementations of the color balance: an 8-bit integer implementation based on the histogram algorithm with $O(N)$ complexity and a lookup table for fast affine transform is used for the RGB color balance, and a generic floating-point implementation based on `qsort()`, with $O(Nlog(N))$ algorithmic complexity is used for the IRGB intensity balance.

The histogram code is used for the online demo[27]. The source code history and future releases are available on an external page[28].

7.2.9 Examples

We show from left to right the original image, and its result by RGB color balance with 0%, 1%, 2% and 3% of the pixels saturated, half at the beginning of the histogram and half at the end of the histogram. In these examples, the algorithm has been applied independently on each color channel. It is quite apparent that some saturation is almost always necessary, but that the needed percentage is variable.

In the figure 7.18, the 0% saturation already gives a result, and 1% is optimal. Notice how the orange ambient light has been corrected to more daylight image.

A white thin rim surrounds the image in the figure 7.19. This rim occupies more than 2% of the image. Hence, the 3% threshold is the right one.

Like the preceding ones, the image in figure 7.20 in completely unnatural blue light is often used to illustrate color balance, or color contrast adjustment algorithms. A trivial affine transform corrects it adequately by removing the bluish effect. A still more contrasted result is obtained by saturating only 1%.

The same remarks as for the preceding one apply to the figure 7.21.

Even a good quality image can benefit from a moderate 1% color balance (figure 7.22). A contrast improvement is noticeable when switching between the 0% and 1% versions.

There is no real good solution for the sunset image (figure 7.23). The colors are strongly blue/orange and will stay so. By pushing too far the saturation (3%), the orange pixels diminish and a completely unnatural blue color is created.

The images in figures 7.24, 7.25 and 7.26 have been used in recent papers on color perception theory (Retinex).

[23] `http://www.gnu.org/licenses/gpl.html`

[24] `http://dx.doi.org/10.5201/ipol.2011.llmps-scb`

[25] `http://www.libpng.org/pub/png/libpng.html`

[26] Linux: you can install `libpng` with your package manager; Mac OS X: you can get `libpng` from the Fink project (`http://www.finkproject.org/`); Windows: precompiled DLLs are available online for `libpng` (`http://gnuwin32.sourceforge.net/packages/libpng.htm`).

[27] `http://www.ipol.im/pub/demo/lmps_retinex_poisson_equation/`

[28] `http://dev.ipol.im/git/?p=nil/simplest_color_balance.git`

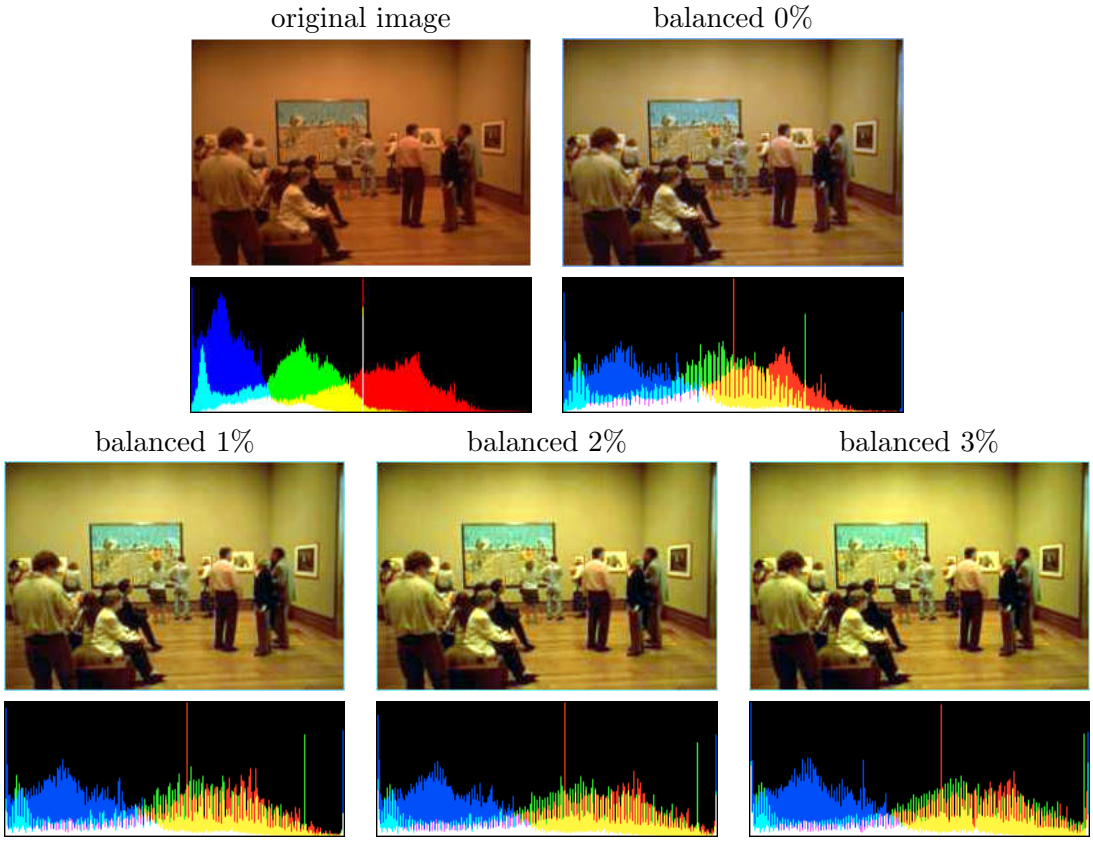


Figure 7.18: Museum in orange ambient light.

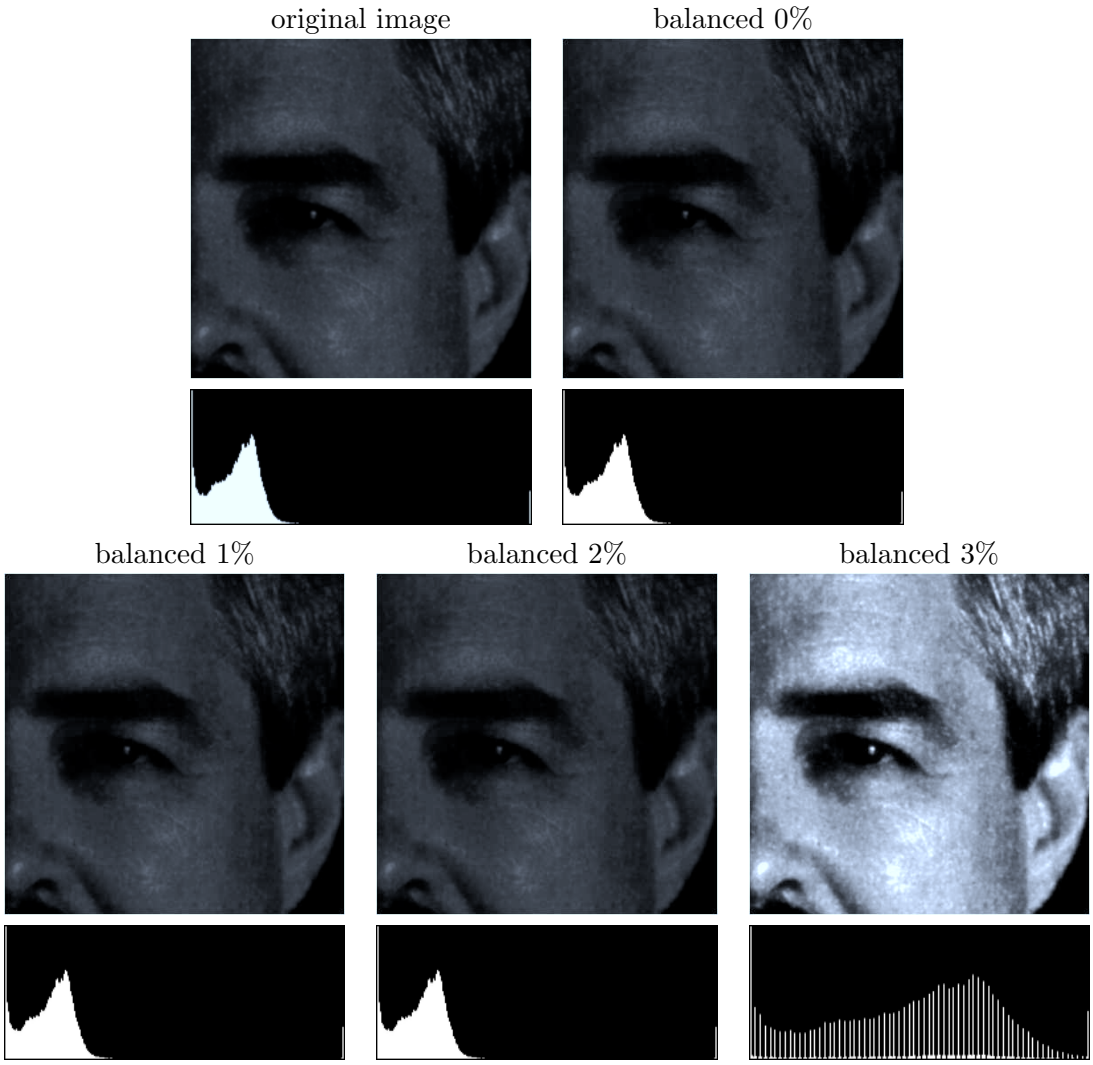


Figure 7.19: Face with a white rim.

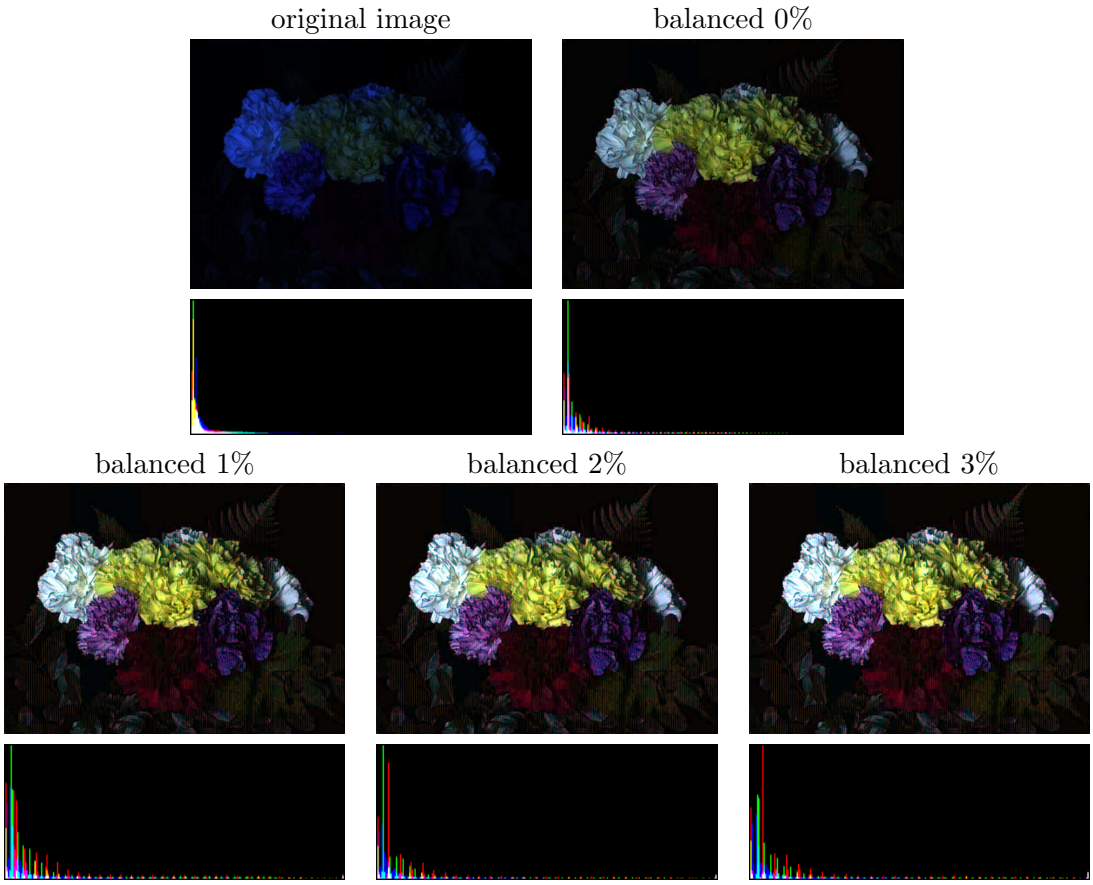


Figure 7.20: Flowers in blue light.

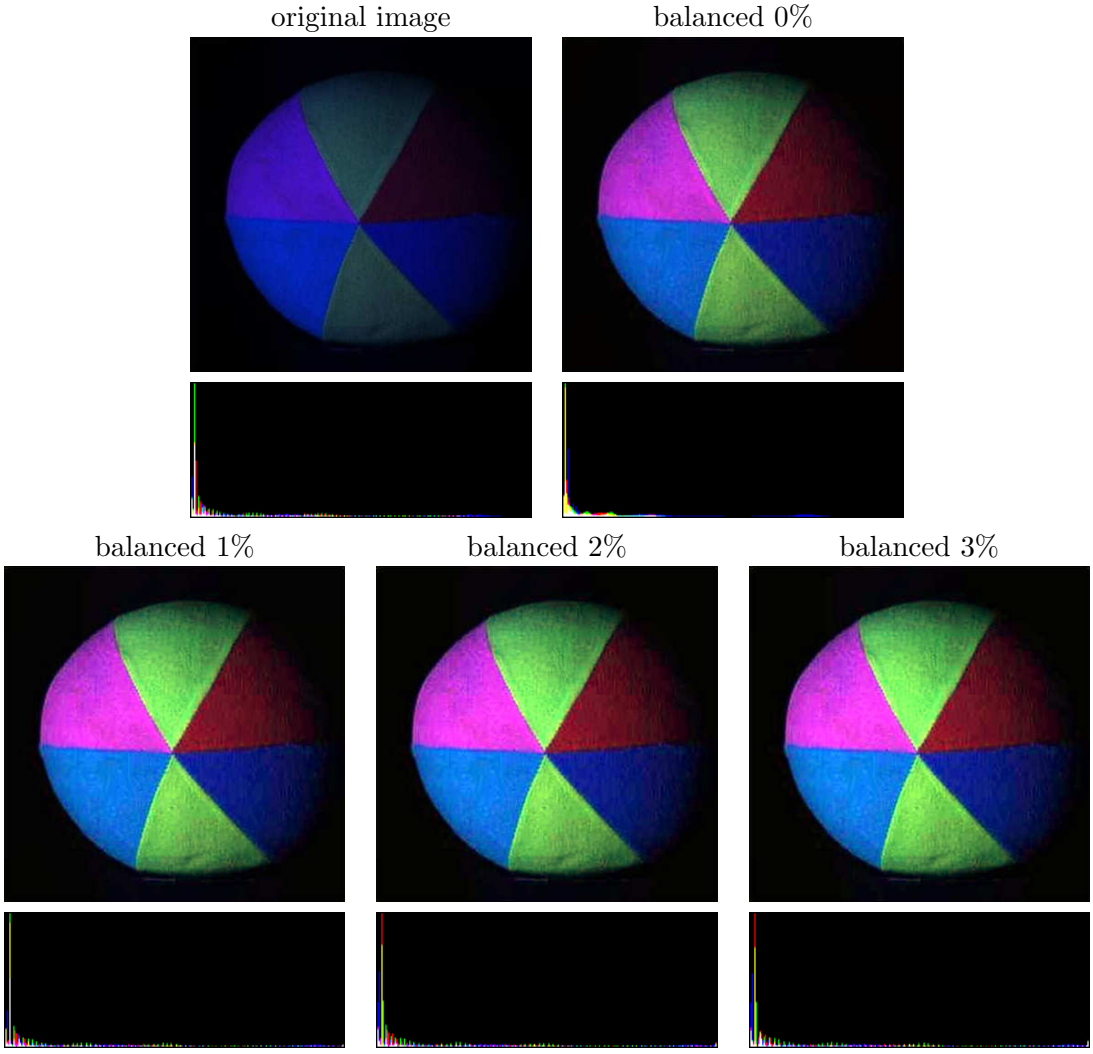


Figure 7.21: Ball in blue light.

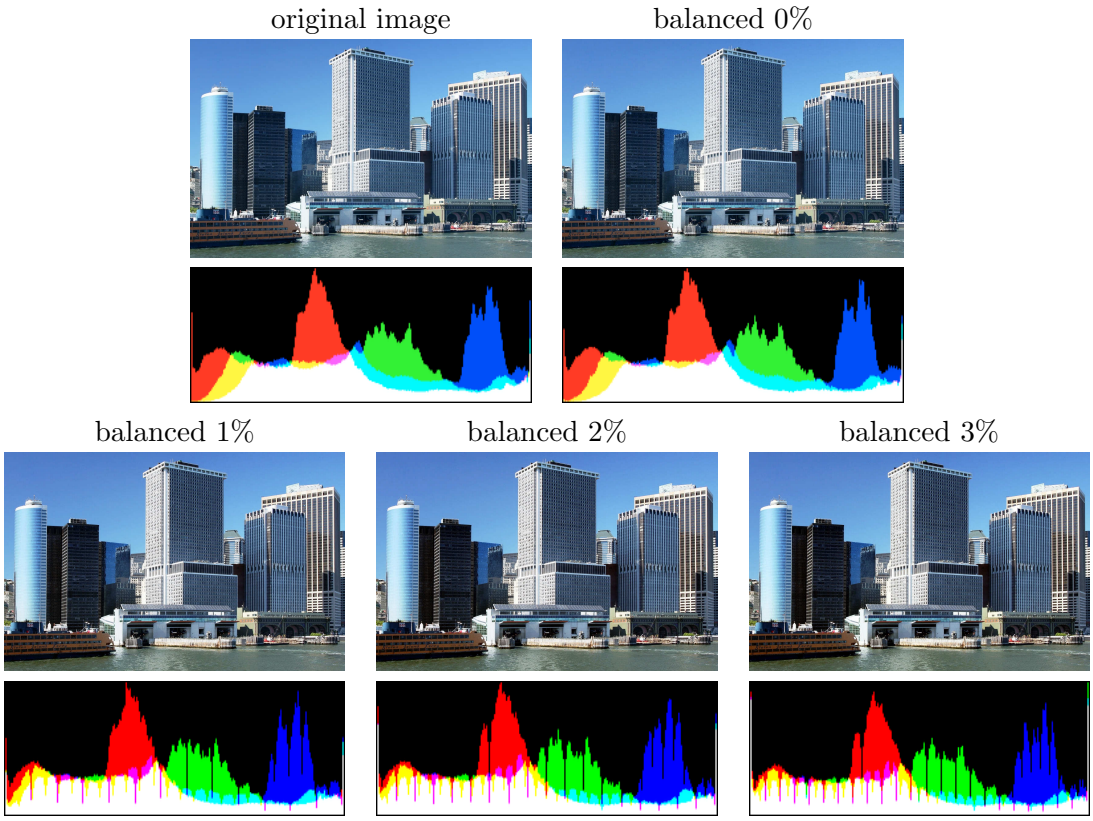


Figure 7.22: Good original image improved by a 1% balance.

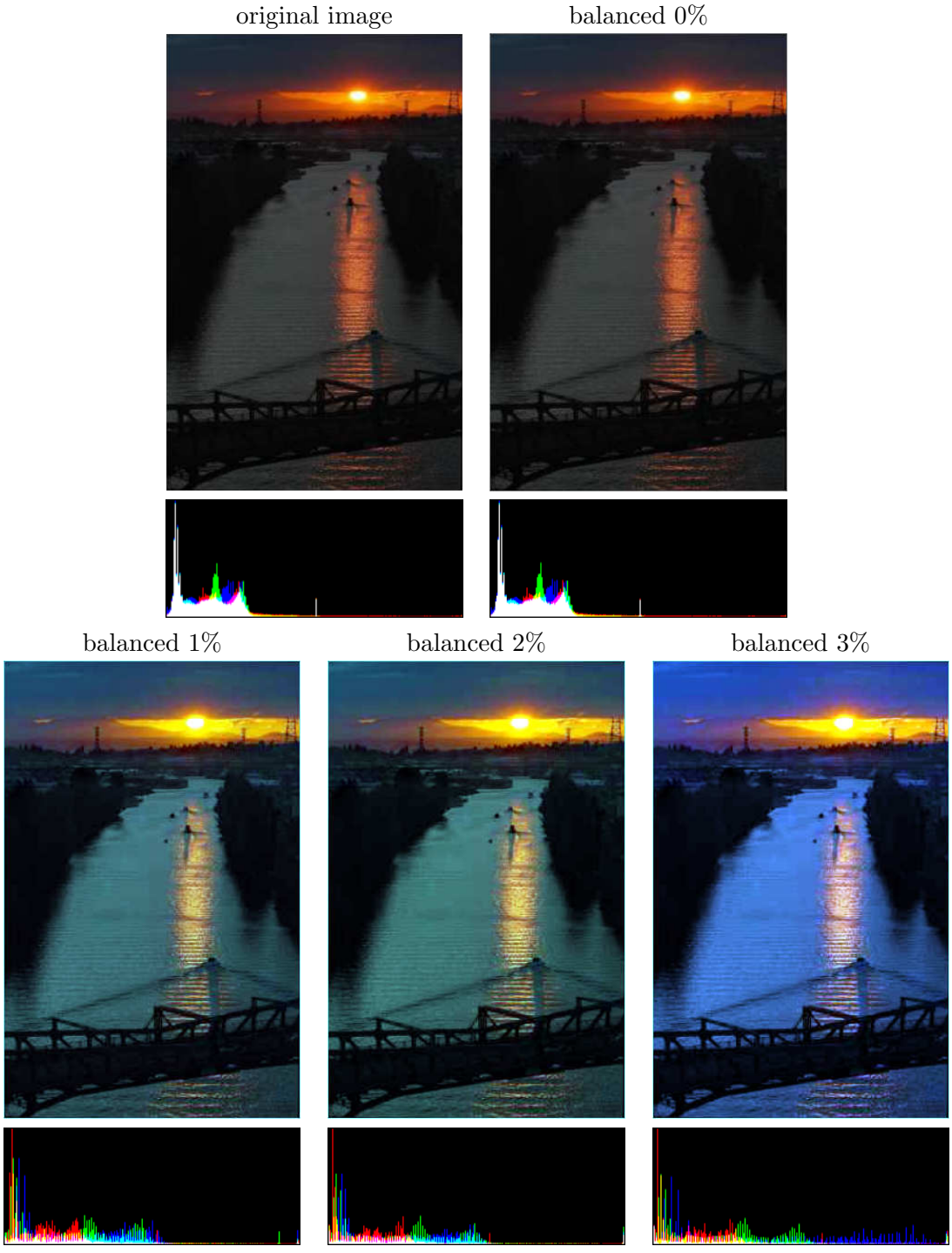


Figure 7.23: Sunset.

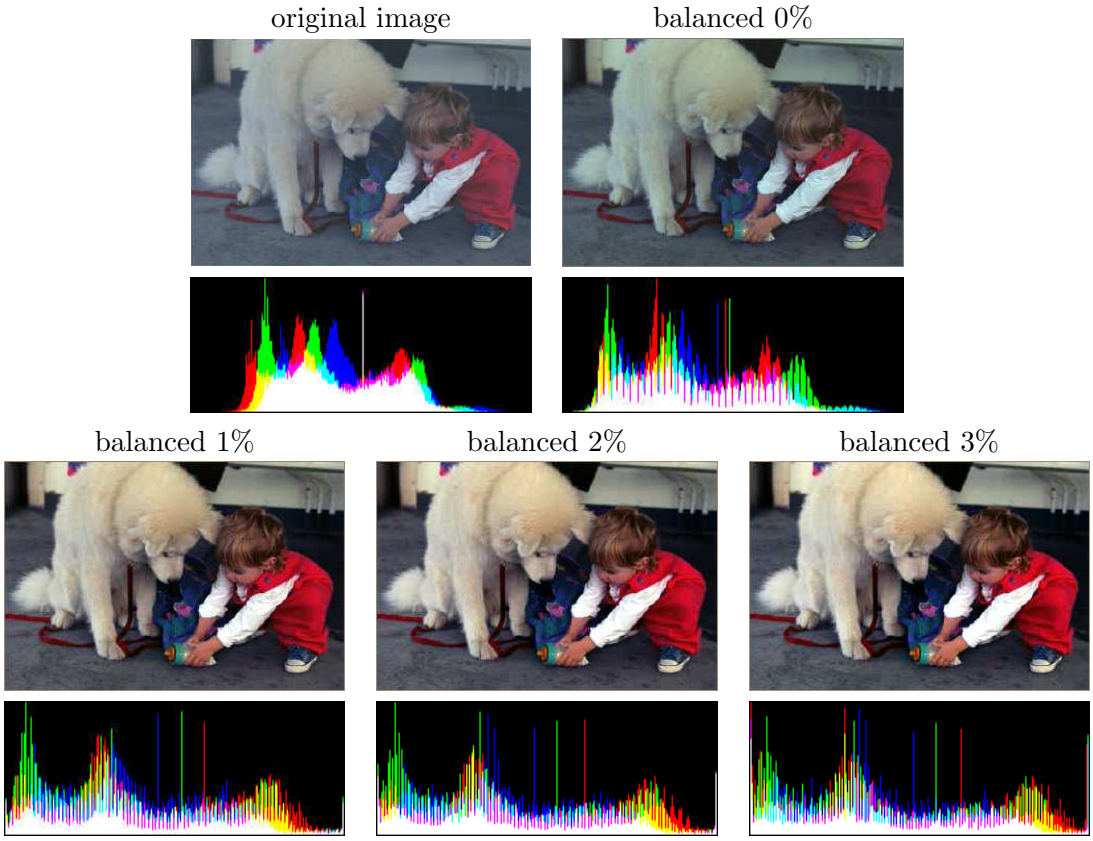


Figure 7.24: Child with a dog.

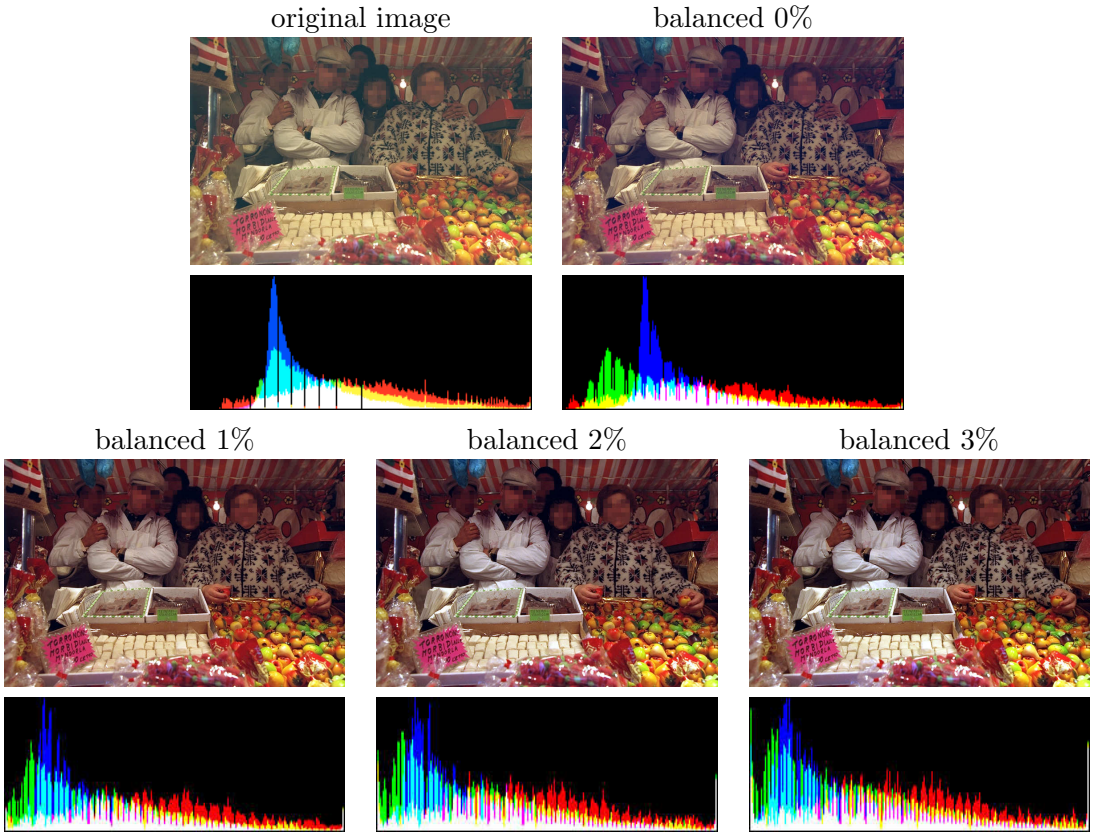


Figure 7.25: Merchants.

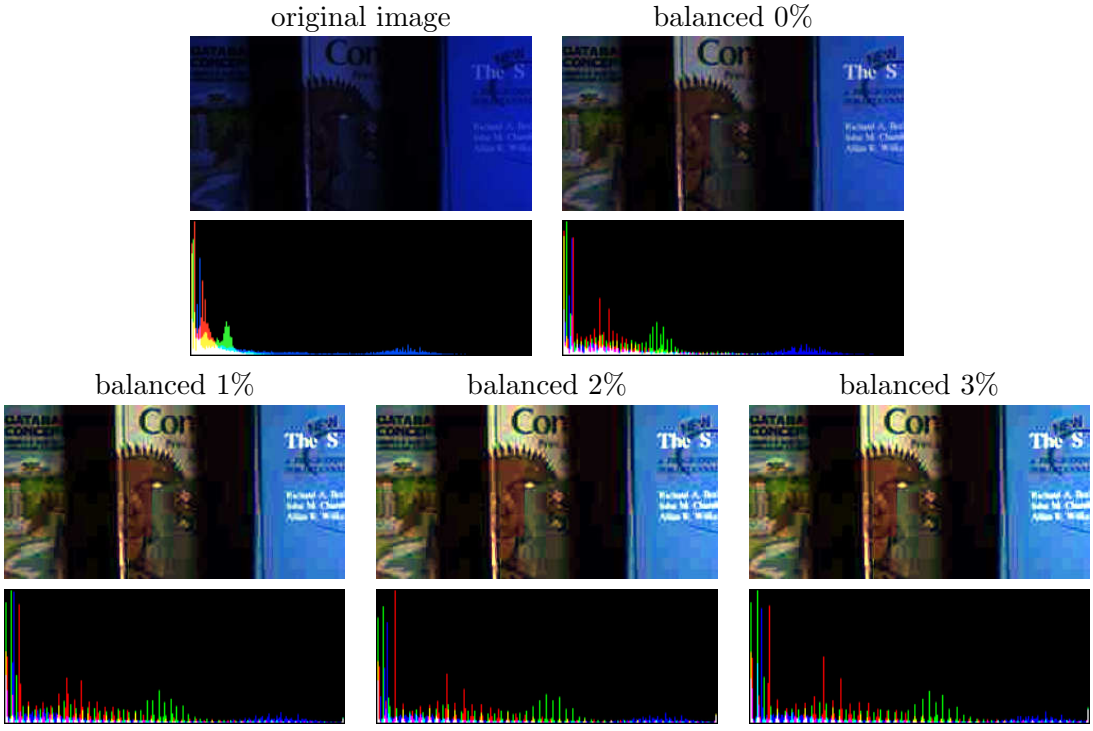


Figure 7.26: Books.

Examples on Gray-scale Images

We present two examples on grayscale images. The algorithm is the same, since the three channels are equal. In the figure 7.27 we observe a small improvement of the image. In the figure 7.28, the improvement is more significant, but we lose the little details in the image.

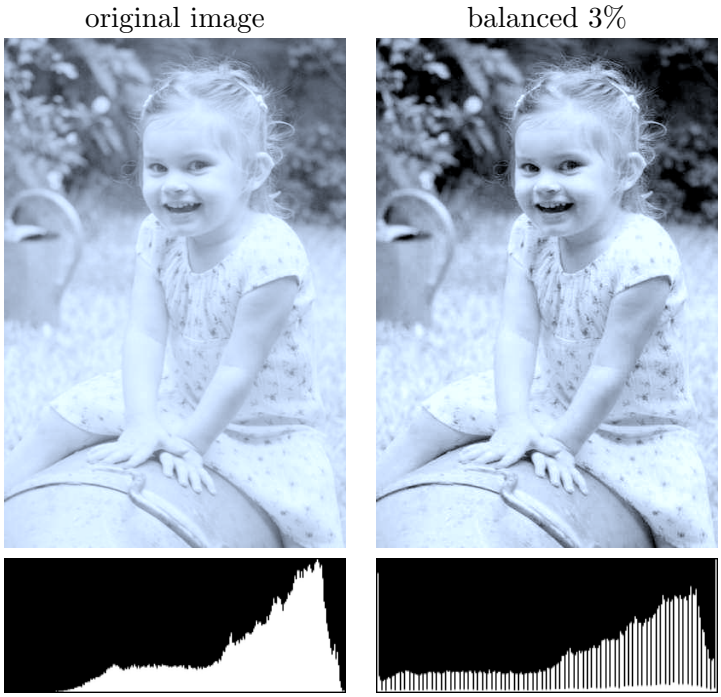


Figure 7.27: Young girl in gray-scale.

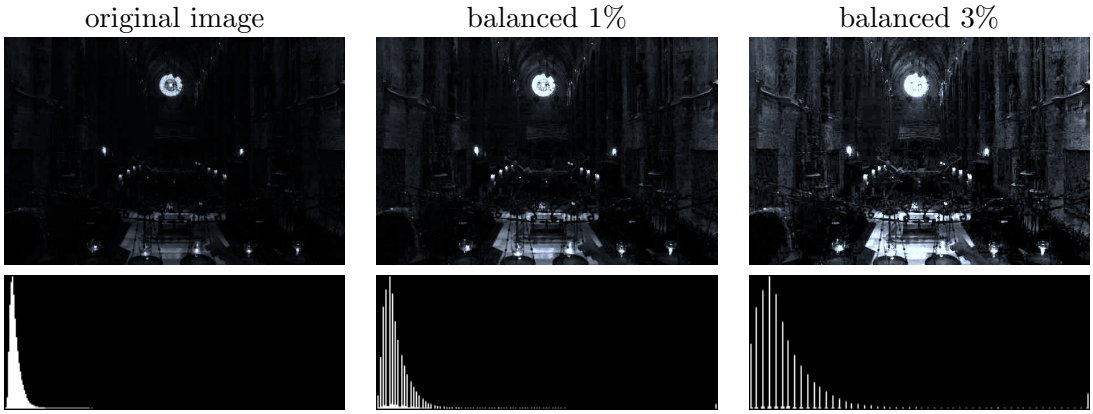


Figure 7.28: Cathedral in gray-scale.

Examples with Little or no Improvement

In the figure 7.29, it is probably best not to apply any color balance: the colors become quickly unearthly.

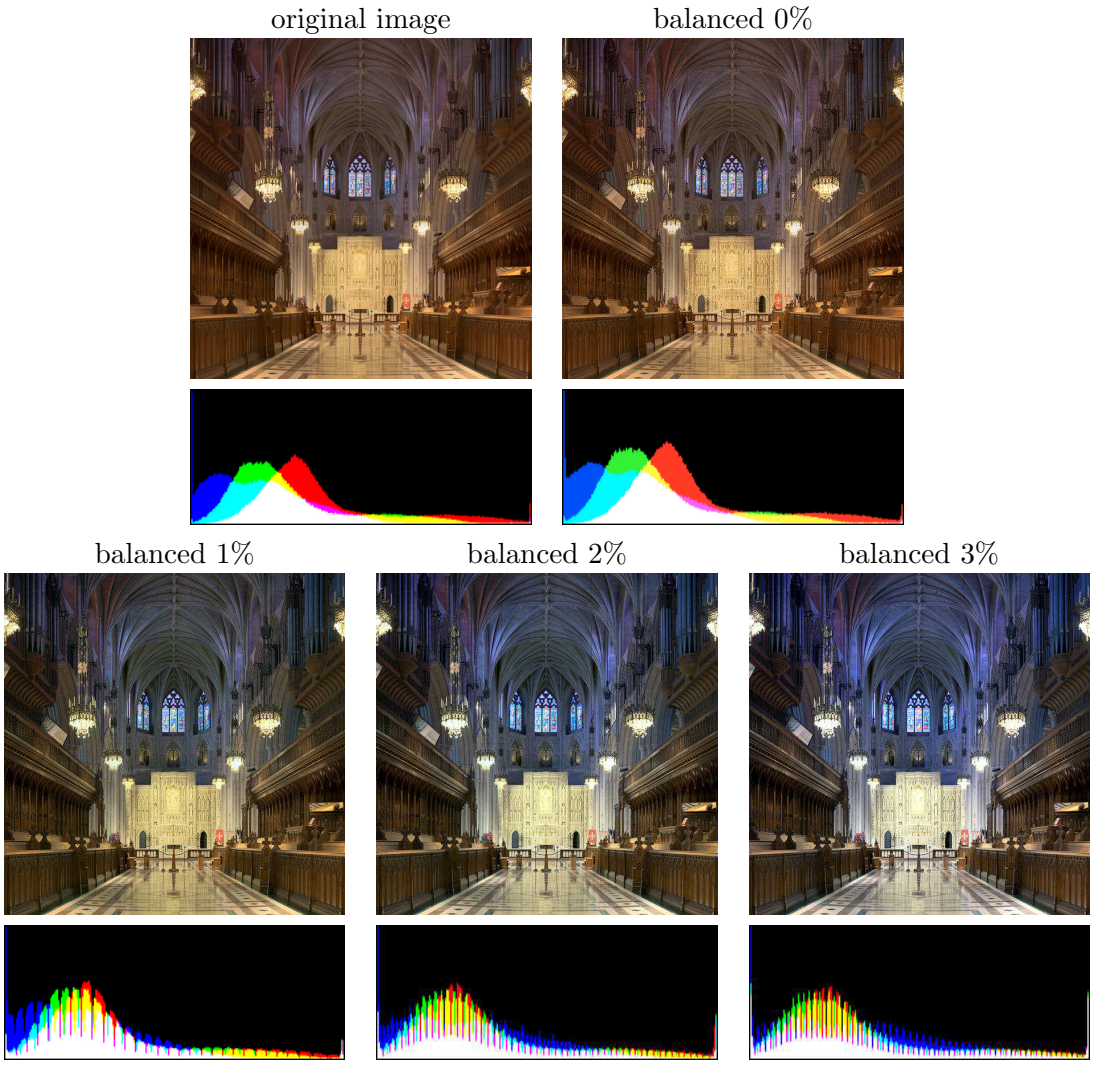


Figure 7.29: Cathedral in high dynamic range color.

The figures 7.30 and 7.31 with back-lighting have no simple solution.

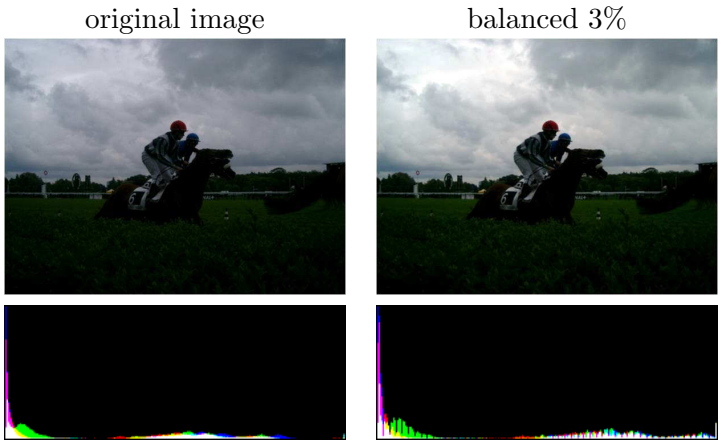


Figure 7.30: Horse in back-lighting.

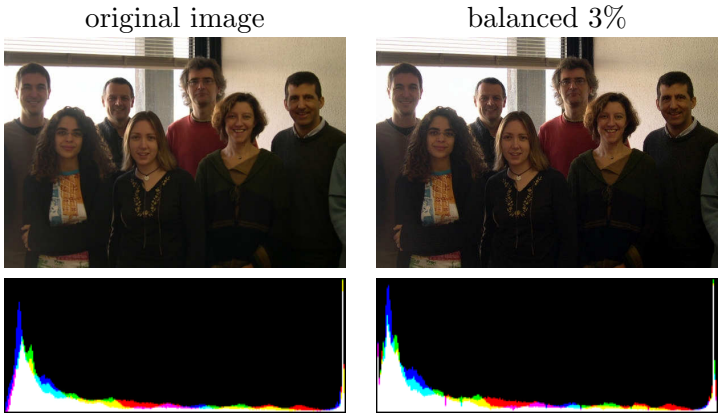
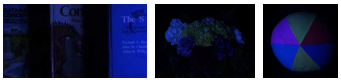


Figure 7.31: Group in back-lighting.

7.2.10 Credits

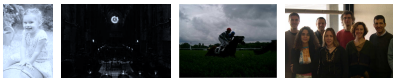
courtesy Philip Greenspun[29]



Kobus Barnard, SFU Computational Vision Laboratory[30]

Noclip, Wikimedia Commons, public domain[31]

[29] http://philip.greenspun.com/
[30] http://www.cs.sfu.ca/~colour/data/
[31] http://commons.wikimedia.org/wiki/File:National_Cathedral_Sanctuary_Panorama.png



Ana Belén Petro

Daniel Schwen, Wikimedia Commons, GFDL/CC-BY-SA[32]

Drew Streib, Flickr, CC-BY-NC[33]

7.2.11 Software and Demo Design

These additional sections about the implementation design and the online demonstration were not in the online version.

Implementation

The "*Simplest Color Balance*" is probably the simplest algorithm published in IPOL, only composed of a quantile computation and an affine transform. However, even such a simple algorithm can be implemented with care for efficiency and reusability.

Code Design In fact two algorithms are implemented: the "*RGB color balance*" and the "*IRGB intensity balance*". These two variants are similar, but while the former can be applied to images expressed as unsigned 8-bit integers, the latter cannot because of the RGB/I conversion step. The "*IRGB*" variant could have been implemented with integer arithmetic, with $3I \in [0..3V_{max}]$, but be preferred to use floating-point values in a straight forward code demonstrating an alternative to efficient but integer-only methods.

The two variants are thus implemented with different functions but follow the same design. The function calls and file dependencies are represented in figure 7.32.

- `balance.c` contains the `main()` function and is responsible for all the tasks only relevant to the execution environment: processing the command-line parameters, reading the input file, calling the color balance functions and writing the output file. The image data format (integer or floating-point) and the algorithm variant to be applied are determined by a command-line parameter. This is the only source code file with assumptions about how the program will be used, and as such this code is not reusable and not expected to be reused.
- `colorbalance_lib.c` contains the high-level implementation of the color balance algorithms, isolated in a single source code file reusable in other software projects via two functions, `colorbalance_rgb_u8()` and `colorbalance_irgb_f32()`. These functions process color vectors, without any need for the notions of images or two-dimension arrays.

[32] `http://commons.wikimedia.org/wiki/File:Staten_Island_Ferry_terminal_crop.png`
[33] `http://www.flickr.com/photos/dtype/145118964/`

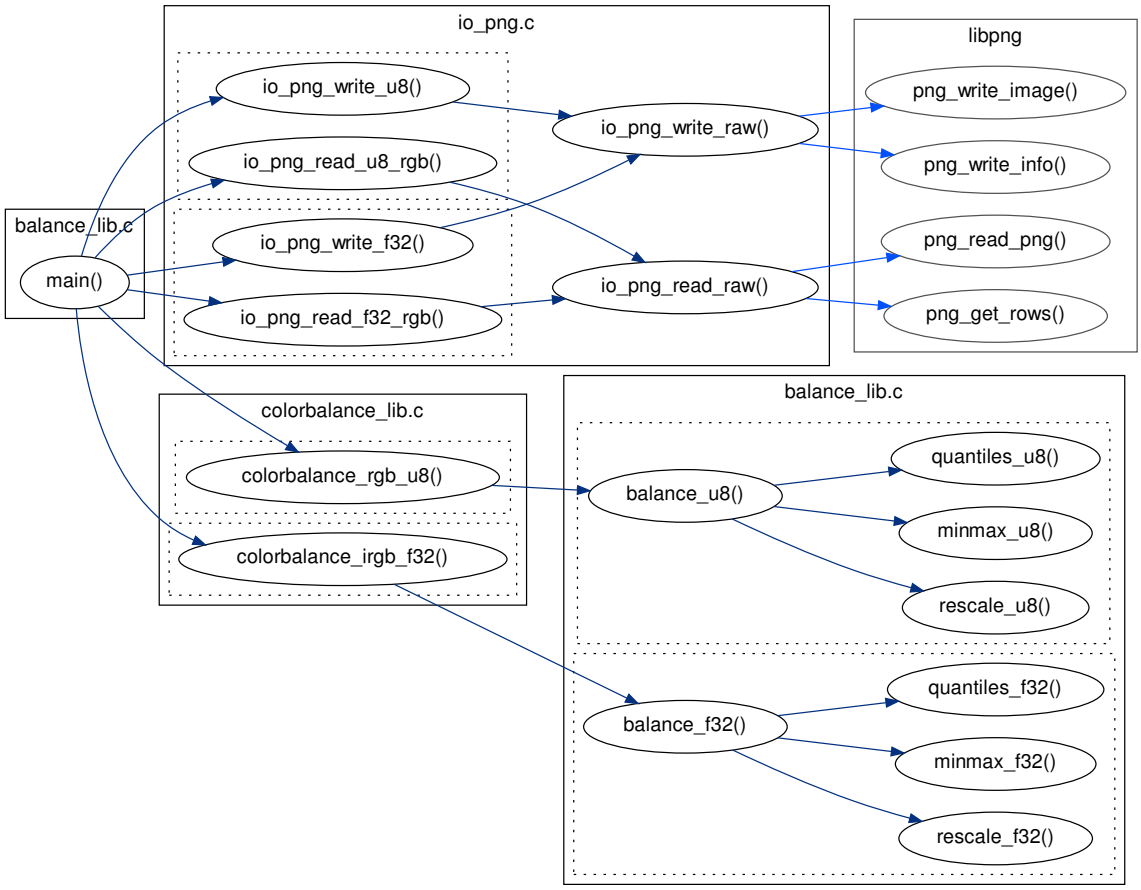


Figure 7.32: Function calls and file dependencies for the "*Simplest Color Balance*" implementation. Minor functions have been omitted for clarity.

- `balance_lib.c` contains the low-level routines used by the color balance functions: quantile computation and affine transforms for integer and floating-point. These generic functions do not need any color information, they only process generic vector data.

- `io_png.c` has the simplified interface to `libpng`, with routines to read image files into arrays or writes them with a single function call. It lets the programmer choose which kind of image is to be processed (grayscale or color, integer or floating-point) and internally converts the PNG image to the expected format.

Optimization First, the quantile computation can be avoided if the color balance is to be done without saturation. In that case, an early test replaces the quantiles with the minimum and maximum of the array.

Then, for 8-bit integer data, the quantiles are computed by a histogram method. This method has many advantages: is is very simple, computes in linear time and uses a fixed memory space for every image size. Floating-point data, however, cannot be processed into an histogram. A `quickselect` algorithm [128] would probably be the best solution with fast computation in linear time, but was no chosen for the implementation published in IPOL in order to minimize the amount of code to be reviewed, and because the performances were good enough with a simple sorting algorithm using `qsort()`.

The quantiles are then used in an affine transform applied to the image values. For integer images, a lookup table is used. For every of the 256 possible input values, the result of the color balance transform is precalculated and stored. Then these values are retrieved instead of re-computing them for every pixel. This is efficient because few input values can be encountered, and the table is small enough to be efficiently stored in the processor cache. This method is not adapted to floating-point values, which are handled instead by a simple affine transformation loop. The performances of this loop could probably be improved: the branching instructions could be avoided in the loop by using bit-twiddling code, and the use of vector SIMD instructions by the compiler could be facilitated.

Quality Control The program is a standard C89 source code. It only needs a C compiler and the `libpng` external library and should be compilable and usable in any computing environment. The standard compliance was tested by using strict compilation options[34] and the source code quality was further checked by static code analysis with Splint and Clang[35]. These tests do not guarantee the quality of the code, but they can be used to detect and fix known dangerous patterns and abuses of the language.

After the publication of this code in IPOL[36], these tests were further improved with the usage of other static code analysis tools (Cppcheck[37]) and by compiling the program with various C and C++ compilers[38] and fixing the compiler warnings. Finally, functional tests were added to verify that the exact same results are obtained with all these compilers and all the successive versions of the code.

[34] The basic set of compilation options was `gcc -ansi -pedantic -Wall -Wextra -Werror`.

[35] Splint (`http://www.splint.org/`) is a static code analysis tool. The Clang compiler (`http://clang.llvm.org/`) has static analysis functions.

[36] `http://dev.ipol.im/git/?p=nil/retinex_pde.git`

[37] Cppcheck (`http://cppcheck.sourceforge.net/`) is a code analysis tool for C and C++ programs.

[38] We use the following compilers: GNU `gcc` and `mingw`, LLVM `clang`, Intel `icc`, Sun `suncc`, `tcc`, and `nwcc`.

This quality control is routinely performed via tests integrated in the build procedure, and automatically performed every time a new version is recorded:

- the code is built by the default C compiler, with cross-compiler compatible options and the possibility to specify the compiler at build time;
- the pseudo-target `lint` checks all the source code with static checking tools and compiles it with strict compiler options;
- the pseudo-target `beautify` maintains the indentation by processing all the source code with a code reformatting tool;
- the pseudo-target `test` builds the program with various compilers, runs functional tests and executes the program in a dynamic memory checking environment.

Demonstration

Workflow The demonstration of this algorithm proposed in IPOL [302] follows a basic workflow, illustrated in the chart 7.33 and screen captures 7.34.

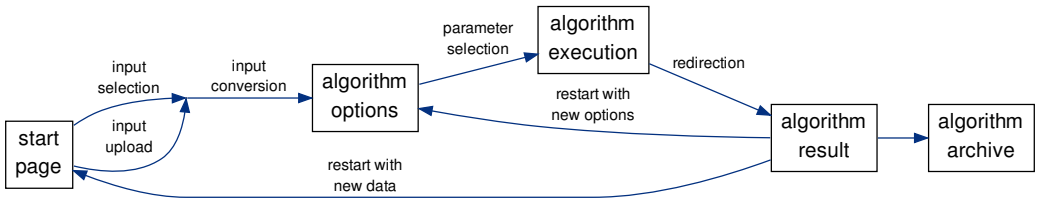


Figure 7.33: Typical IPOL demo flow chart, applicable to the "*Simplest Color Balance*" case.

- The start page proposes a few input images for the algorithm. With these images, demo users can reproduce the results published in the article. They can also upload their own images to try the algorithm.
- On the parameter page, one can choose the saturation levels, and launch the algorithm.
- The result page is displayed after a few seconds, with the output of the algorithm. Then one chooses to restart the demo with another threshold parameter, or an new input image.

Usage From its first public release on August 28th, 2009, to December 31st, 2011, this demo has been used more than 1300 times with original input images; 1000 of these experiments are publicly available in the archive[39]. Based on recent statistics, we can estimate that the algorithm has been tested more than 4500 times by IPOL visitors and collaborators with this demonstration interface during the same period.

[39] `http://www.ipol.im/pub/demo/lmps_retinex_poisson_equation/`

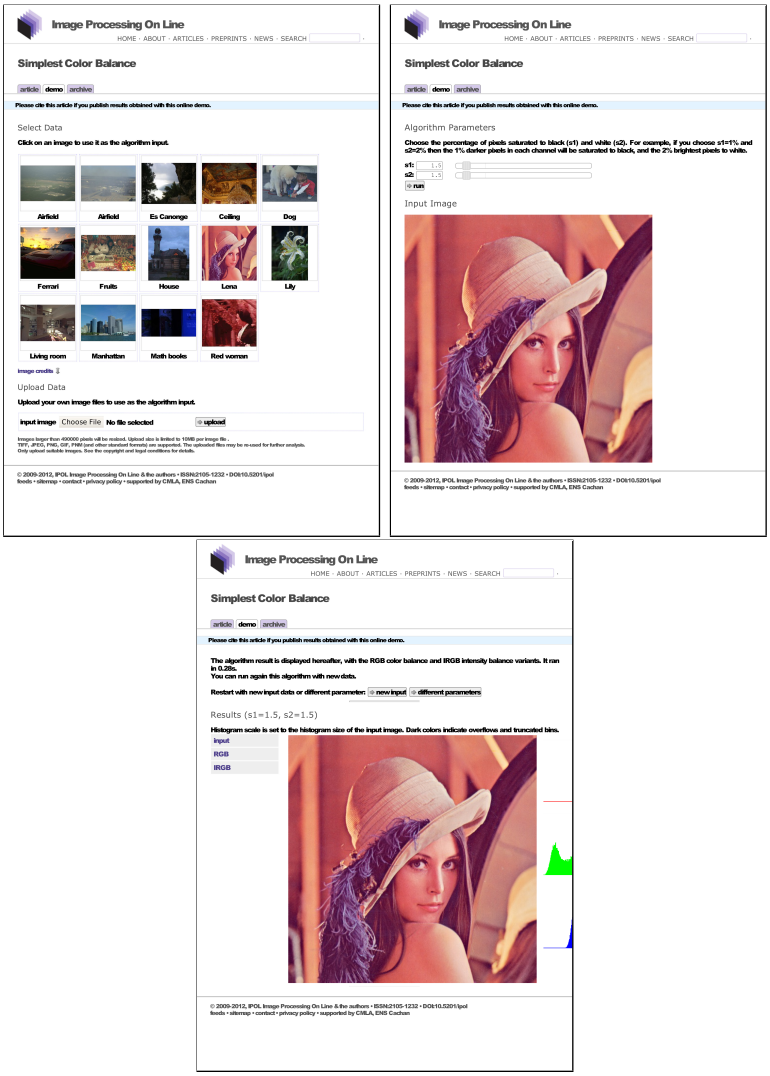


Figure 7.34: Input selection, parameters and results of the IPOL demo for "*Simplest Color Balance*".

Chapter 8

Usage and Feeback

Contents

Abstract

Some authors answered a survey about their experience in publishing an article in IPOL. These answers show their enthousiasm for this new form of publication and the feasibility and interest of an online journal with a software component, but some specific tools and interfaces are needed to assist the authors.

On the other side, usage statistics collected from the visits of the journal provide an estimation of how the articles are accessed and used, and this popularity index can be used as a complement to the traditional citation index.

8.1 Authors Survey

53 authors of IPOL articles, published or in process, were invited to answer a survey in November 2011. 24 answers were collected and analyzed with the LimeSurvey[1] web survey software. When the authors were involved in more than one article, they were asked to consider the article most representative of their IPOL experience when answering. We present a synthesis of these answers hereafter.

8.1.1 Profile of the Authors

Half of the authors who answered the survey were tenured researchers, and the other half were post-doc researchers or PhD students (table 8.1). They were involved in three kinds of articles (table 8.2). 40% of the articles are implementations and analysis of a classic algorithm previously presented by other authors in another paper. 30% of the articles are the implementations and analysis of algorithms previously presented by the same authors in another paper, and the remaining 30% are the implementation and analysis of an original algorithm.

author	
master student	2
PhD student	7
post-doc	2
junior researcher	2
senior researcher	10

Table 8.1: Categories of authors.

article	
original algorithm	7
re-publication	7
classic algorithm	10

Table 8.2: Categories of articles.

The majority of the authors use the Linux operating system for their research tasks. Mac OS X could be merged with Linux in a POSIX category which is clearly the main

[1] `http://www.limesurvey.org/`

computing environment for the authors, but Windows counts for 25% and cannot be ignored (table 8.3).

operating system	
Windows	6
Mac OS X	6
Linux	12

Table 8.3: Usage share of operating systems.

The C and C++ languages are the main programming environments for the authors (table 8.4). Others (20%) use MATLAB and Python. As seen in later survey items, MATLAB and Python users could adapt to the requirement of a C/C++ source implementation of the algorithm. This ranking matches the programming languages observed in the industry[2] and Open Source[3] communities where C/C++ is the most common programming platform, with the exceptoion of Java, notably absent from scientific computing projects.

programming language	
C	8
C++	10
Python	1
MATLAB	4
other	0

Table 8.4: Usage share of programming languages.

No author uses the Internet Explorer browser (table 8.5). This is interesting, because this browser has some issues with standard compliance and some web technologies have been deliberately avoided in IPOL to be accessible from Internet Explorer users. If this tendency is confirmed in a larger survey, we could consider the essential visitors of the IPOL web site do not use Internet Explorer or can switch to another browser, and we could use more advanced web techniques supported by Firefox and Chrome[4].

web browser	
Internet Explorer	0
Firefox	8
Chrome	12
Opera	0
other	3

Table 8.5: Usage share of web browsers.

However, there is a bias in these last three results because only IPOL authors were asked to answer. A larger survey targeting image processing researchers is needed to confirm these statistics.

[2]TIOBE Programming Community Index for April 2012 (`http://www.tiobe.com/index.php/content/paperinfo/tpci/`).

[3]Ohloh language comparisons (`http://www.ohloh.net/languages/compare`).

[4]Users of "other" browsers were asked for details. They use the Rekonq and Epiphany browsers, both based on the standard-compliant rendering engine Webkit.

8.1.2 Writing an Article

The redaction an IPOL article seems to require as much work as writing an article for a "classic" journal (table 8.6), and adapting a software to the IPOL requirement seems to double on average the time required for development (table 8.7), but we can observe large deviations in this table, probably connected to the individual proficiency of the authors in software development and the origin and characteristics of every software[5].

$\times$ 0.25	5
$\times$ 0.5	5
$\times$ 1	8
$\times$ 2	2
$\times$ 4	2

Table 8.6: Redaction time for an IPOL article, compared with a "classic" article.

$\times$ 1	2
$\times$ 1.5	9
$\times$ 2	6
$\times$ 4	6

Table 8.7: Development time for an IPOL program, compared with a "classic" article.

None of the requirements for the implementation of algorithms seems to be a serious issue for the authors (table 8.8). These requirements usually are no problem at all, or can be fulfilled with a small additional work. The two authors who felt the C/C++ requirement had an heavy impact on their work were used to develop in MATLAB or in the MegaWave [136] environment. Overall, the portability, library and file format restrictions seem to have the most noticeable impact, but it appears to be manageable for most of the authors.

	A	B	C	D	E
usable in command-line	0	0	1	9	14
source code	0	0	5	5	14
detailed documentation	1	0	5	10	8
implementation in C/C++	1	1	3	2	17
portable implementation	0	2	6	11	5
restrictions on file formats	1	2	1	10	10
restrictions on libraries	1	3	3	10	6

Table 8.8: Impact of the IPOL software guidelines on software development; A: lot of extra work, B: some extra work, C: not an obstacle, D: minor impact, E: no impact

All the authors chose to publish their code under the GPL family of software licenses (GPL/LGPL/AGPL). Three authors declared that they have registered a patent for the algorithm they implemented and published.

[5]It is interesting to observe that this overhead is compatible with the "rule of thumb" $\times 3$ overhead estimated by Frederick P. Brooks for passing from a debugged program to a "programming product", usable, testable and modifiable by anybody [61].

8.1.3 Priorities

All the authors declared they would cite an IPOL article in a research paper and suggest colleagues to read IPOL materials, and more than 90% of them would write another article in IPOL and suggest colleagues to do the same.

Respondents were also asked to order a list of nine possible improvements, and their preferences were aggregated into a list of priorities[6] (table 8.9).

1	a better interface to edit the articles
2	a better indexation in academic journals databases
3	some source code tools and libraries
4	a better system for the web demos
5	a better archive for the web demos
-	a better interface to handle the reviews of the articles
-	get some feedback from the readers for every article
-	improving the design of the IPOL web site
-	support for other kinds of data (sound, video)

Table 8.9: Priorities for IPOL developments (the last 4 items were ranked too low for a meaningful order).

The first priority is the improvement of the interface used to edit the articles. Other questions showed that the authors were dissatisfied with the wiki-like edition environment and the Markdown[7] syntax currently used for the articles, and they would prefer to send their articles instead, using LaTeX (tables 8.10 and 8.11).

are you satisfied with...	Yes				No
the edition interface	7	1	4	6	6
the edition language	3	4	4	9	4
the article web format	11	10	2	1	0
the information about the review process	6	7	5	4	0
the speed of the review process	7	4	6	1	1

Table 8.10: Satisfaction with the edition environment and review process.

would you prefer...	Yes	Uncertain	No
a submission system	12	2	9
to use LaTeX	19	3	2
to publish as PDF	5	9	8
a web review interfaces	9	11	3

Table 8.11: Propositions of alternative edition options and review tools.

However, there was a positive feedback on the principle of publishing the articles as HTML pages, and not much enthusiasm with the proposition to switch to a PDF format. The

[6] The winning choice was selected with a Condorcet method, then removed from the ballots, and the process was repeated to elect the next choices.

[7] `http://daringfireball.net/projects/markdown/`

good option seems to receive the articles written in LaTeX and produce an HTML version from it.

We expected some criticism of the current review system, but its improvement is ranked low in the priority list. The majority of the authors are satisfied or very satisfied with the current documentation and speed of the review process. They would prefer a web-based review interface, but the demand is not as clear as for a new edition interface, with many uncertain respondents (tables 8.10 and 8.11).

8.1.4 Satisfaction

All the authors declared they would cite an IPOL article in a research paper and suggest colleagues to read IPOL materials, and more than 90% of them would write another article in IPOL and suggest colleagues to do the same.

No author had a bad impression of IPOL and almost all rated their experience positive or very positive. When asked to detail the reasons for this satisfaction in a supplement survey, they cited the use of IPOL as a complete archive of all the materials related to an algorithm.

> "*For the first time I have been able to show to my colleagues what I do, and exactly how I do it.*"
> "*Being forced to wrap-up implementation in a usable and documented way is useful as a future self-reference (not forgetting what you made yourself), for teaching, for reference to other colleagues, and as a way to obtain feedback.*"
> "*It is a very convenient way to demonstrate my work without having to remember where my program is, its arguments, find appropriate images, etc. Associating a complete description with a fixed version of the algorithm avoids forgetting 'what are the features and limitations of this version of the code?'.*"

The web demonstration tools are useful to get a better understanding of the algorithms, for the authors and for the readers.

> "*It's easy to prove that the author's algorithm works using the online demo. The online demo allows the public to understand better the algorithms.*"
> "*IPOL allows to make a lot of experiments and better understand the models.*"
> "*I have learned a lot from the archive of my article.*"

The publication of the implementation is appreciated as a motivation to produce a better program.

> "*It gave me the opportunity to show my work (not only the mathematics, but also the code) to the image processing groups around the world*"
> "*Exposing the code to everybody makes me work harder to have source code I could be satisfied with and not stop at the 'it works' point.*"
> "*It drove me to change the way I will code for future publications.*"
> "*It is an opportunity to write a complete and debugged code and this work is valued.*"

And by publishing in IPOL, the authors re-evaluate the importance and usefulness of academic papers.

> "*It made me re-think the way results are shown, particularly from the visualization point of view. In addition, it made me also re-think what are the important contributions of a scientific publication,* i.e. *the value of re-publishing a previously published paper for instance.*"
> "*My work is more useful for the academic community than a normal paper.*"

These reasons for the satisfaction of the author match the main reasons to share code and data, as observed in the machine learning research community [328]: set a standard for the scientific method in computational sciences and improve the visibility of one's work and its usefulness for the authors and for others[8]. We think that the requirement to provide the code and its inclusion code as a primary reviewed, published and referenced material instead of the usual "supplementary materials" section helps mitigate some of the reasons not to share observed in the same study.

8.1.5 Conclusions

We can draw the following conclusions from this short survey. However, due to the small number of answers, they would need to be confirmed by further studies, and a survey on image processing researchers out of the IPOL community.

- the IPOL software requirements are not a serious obstacle for publication among the current authors;
- more work is needed to publish in IPOL than to publish elsewhere, mainly for the software part, but this doesn't affect the author's overall satisfaction;
- the priority must be a reform of the edition/submission process to accept LaTeX input and abandon the wiki-like model, but publishing the articles in PDF is not enough;
- IPOL needs to be actively inserted in academic publication databases.

Despite the improvements needed in the IPOL tools and process, the idea of publishing algorithms online with their implementation and demonstration is very appreciated by researchers.

8.2 Web Statistics

Visits of the IPOL website are tracked with the Piwik[9] web statistics software. This information is completed with our own measures of the web activity, including the web

[8] In the aforementioned study, The top reasons to share code and data were: - Encourage scientific advancement - Encourage sharing in others - Be a good community member - Set a standard for the field - Improve the caliber of research - Get others to work on the problem - Increase in publicity - Opportunity for feedback - Finding collaborators

[9] `http://www.piwik.org/`

demonstrations of the algorithms and their archives, and external information from Google Webmaster[10]. The results for the one year period between December 2010 and November 2011 are presented hereafter.

8.2.1 Visits and Actions

IPOL received 85000 visits during this year, an average of 7000 visits/month or 230 visits/day. We define a visit as a succession of web page accesses without an interruption of more than 30 minutes between 2 successive page views. Half of the visits came from returning visitors, defined as people who visited IPOL with the same computer and browser during the last 3 months. One third of the visits were long visits, with more than one page view (table 8.12).

event	per year
visit	85370
returning visit	42657
long visit	30733
run demo	41472
original data	14978
download	5447

Table 8.12: Visits and actions between December 1st, 2010 and November 30th, 2011.

12% of the visits included the usage of the web demos to test the algorithms, and the visitors who use the demos run them between three and four times per visit on average, for a total 41000 demo executions/year, or 3500 executions/month. One third of these tests are done with original data uploaded by the visitors and the results are archived with 15000 new archives during the last year. 6% of the visitors downloaded a source code or dataset attached to an article, for 5500 downloads/year or 450 downloads/month.

Detailed statistics were compiled in November for all the articles published before (table 8.13) and show the heterogeneity of the IPOL usage. The most popular article received 1563 unique page views, the least exposed article was viewed 64 times. On average, articles are viewed 298 times per month. This can be compared to other journals; SIIMS[11], for example, had 812 downloads per months in 2011 for articles published during the previous year.

The most popular demo was used 1149 times, including 871 times with original data; the least used one processed 13 images, never on original data. Some algorithms, like the interpolation methods, show very consistent results on all sorts of input images, and they are not tested a lot on original data. Other algorithms are very sensible to the characteristics of the input or the demo provides few input examples, so most of the experiments are done on original data.

[10] `http://www.google.com/webmasters/`

[11] SIAM Journal on Imaging Sciences (`http://epubs.siam.org/siims/`).

article	views	demo	(orig.)
my_affine_sift	1563	1149	(871)
bcm_non_local_means_denoising	833	406	(147)
lmps_simplest_color_balance	529	235	(72)
ys_dct_denoising	439	233	(80)
g_linear_methods_for_image_interpolation	434	127	(33)
ags_algebraic_lens_distortion_estimation	421	138	(11)
blmv_nonlinear_cartoon_texture_decomposition	371	198	(116)
lmps_retinex_poisson_equation	319	213	(77)
ggm_random_phase_texture_synthesis	171	102	(38)
m_quasi_euclidean_epipolar_rectification	164	32	(12)
gl_localcolorcorrection	144	31	(2)
blm_color_dimensional_filtering	122	13	(2)
g_malvar_he_cutler_linear_image_demosaicking	111	16	(1)
bcms_self_similarity_driven_demosaicking	107	21	(0)
g_image_interpolation_with_contour_stencils	97	27	(9)
d_point_cloud_data	92	-	(-)
cm_fds_mcm_amss	82	14	(1)
g_gunturk_ap_demosaicking	67	21	(0)
g_interpolation_geometric_contour_stencils	66	41	(21)
g_roussos_diffusion_interpolation	66	12	(0)
g_zhang_wu_lmmse_image_demosaicking	64	13	(0)

Table 8.13: Usage of published articles in November 2011: unique page views, demo executions, and demo executions with original data in parenthesis; "-" denotes the absence of demo.

8.2.2 Origin

The geolocalization of the visitors from their network address with the MaxMind[12] database indicates that they come from 152 different countries. Almost half of the visits are coming from France, United States and China[13] (table 8.14).

country	%
France	15
United States	14
China	12
India	5
Germany	5
Japan	4
Spain	3
Russia	3
United Kingdom	3
Taiwan	2
Italy	2
Korea	2
Brazil	2
Canada	2

Table 8.14: Geolocalization of the visitors.

A list of more than 850 science, research and education institutions could be recognized in the network provenance of the visitors. This list was extracted by looking for keywords in the reverse names of the visiting IP addresses. There probably are some duplicate denominations, and some locations count for hundreds of visits while some others only generated one page view, but this shows the accessibility of IPOL in the research community.

Half of the visitors access IPOL directly (table 8.15). These direct entries happen when the visitor enters the address directly in the browser or uses a bookmark. This usually means that the visitor knows IPOL and often visits the web site.

origin	%
direct entry	46
website	31
search engines	23

Table 8.15: Origin of the visits.

Google identified 446 different domains with a total of 5700 links to `http://www.ipol.im/`, but an important part of these links come from "content farms". These content farms duplicate Wikipedia pages or research papers, including the links to IPOL included in these documents. However, when the incoming links are counted by the number of visits they generate, 537 websites with links followed to IPOL can be observed. Most of the visitors follow a link from Wikipedia. They also come from university, labs and researcher

[12] MaxMind Geolite City geolocalization database (`http://www.maxmind.com/app/geolitecity`).

[13] With a geolocalization at the city level, we can evaluate the impact of the visits from very active contributor groups. Cachan, Palma, Montevideo and Nagoya count, together, for 5% of the visits.

pages, and various technical forums, blogs and indexes related to image processing and computer vision (table 8.16).

website	%
wikipedia.org	42
polytechnique.fr	15
ens-cachan.fr	9
stackoverflow.com	3
cvpapers.com	2
inspirit.ru	1
siam.com	1
visual-experiments.com	1
cvchina.com	1
parisdescartes.fr	1
graphicon.ru	1
csdn.net	1

Table 8.16: Domain share of the visits following a link (websites counting for less than 1% have been ignored).

8.2.3 Settings

We find interesting to compare the technical profile of the visitors with the average Internet users and our knowledge about the IPOL authors. Windows is the operating system of 70% of the visitors (table 8.17), less than the proportion found in global Internet statistics but more than in our group of authors. The Internet Explorer browser is also much less represented than in global statistics with only 19% of the IPOL visitors (table 8.18), but this is to be compared with 0% for the authors.

These statistics make sense when we consider that a large portion of the IPOL visitors come from the research, education, software development and image communities. Linux has more penetration in these groups than in the general computer market, and so do alternative and innovative browsers.

Finally, the browser profile of the visitors can be completed with an evaluation of the support for some web technologies[14] (table 8.19). This information can guide the development decisions of IPOL by avoiding technologies unavailable to a large percentage of the visitors.

OS	%
Windows	71
Linux	16
Mac OS X	11

Table 8.17: Operating systems, all versions aggregated.

[14] JavaScript support is only measured since October 2011. Other technologies are only measured on non-IE browsers.

browser	%
Firefox	42
Chrome	26
Internet Explorer	19
Safari	7
Opera	5

Table 8.18: Browser share, all versions aggregated.

technology	%
JavaScript	97
cookies	95
Flash plugin	90
Java plugin	77
PDF plugin	70

Table 8.19: Browser features available.

Chapter 9

Annex 1: Software Guidelines

Contents

IPOL reviews, uses, publishes and distributes some software provided by the authors. With the requirements and recommendations expressed in these guidelines, we intend to facilitate the production and review of verifiable and usable software for reproducible research.

In Brief: Check List, Check Service and Examples

The list hereafter is a summary of the guidelines, to quickly check an IPOL program. Some are requirements, others are only recommendations. The guidelines are detailed and explained later in this document.

- zip or tar/gzip archive `name_version.{zip,tar.gz,tgz}`, less than 2 MB
- everything into a `name_version/` folder
- file names with `a-z`,`A-Z`,`0-9`,`-`,`_`,`.`
- no hidden file, backup or useless file, no binary
- C89, C99 or C++98 code tested with `gcc -std=xxx -Wall -Wextra -Werror`
- portable code, 32/64-bits, nothing specific to an operating system
- only `libtiff`, `libjpeg`, `libpng`, `zlib`, `fftw`, `cblas` and `clapack` external libraries
- compilation with `make` or `cmake`, only standard options, `make` uses `$(CC)` or `$(CXX)`
- command-line non-interactive interface
- max 1 GB memory, max 30 s computation in the demo environment
- can read/write in PNG, TIFF, PNM, EPS, SVG, VRML or PLY format
- copyright attribution and GPL/BSD license info in every source file
- patent warning if needed
- `README.txt` essential information
- correct, clean code in English
- max 80 characters per line, max 1000 lines per file
- `main()`, algorithmic and auxiliary code in different files
- detailed comments for every function and every implementation step
- example input data and result

A service to check an IPOL program against some of these guidelines is available with examples of programs following the guidelines at `http://tools.ipol.im/pkg/`. This service can be used by IPOL authors to verify their code before submission, and by reviewers as a preliminary validation of the software[1].

[1] The absence of error reported by this service doesn't imply that all the guidelines are correctly followed. Some guidelines need a human review.

About this Document

Status

This document is the official IPOL software guidelines, version 1.00, published on December 20th, 2011. It is immediately applicable and obsoletes previous versions. The reference version is available on line at `http://tools.ipol.im/wiki/ref/software_guidelines/`.

Revisions

When needed, future versions of this document will be published and will replace the current version. The current version will be kept and a summary of the differences will be provided. This revision will be announced on the IPOL website[2] and the IPOL discussion list[3].

Vocabulary

In this document, the term "IPOL program" is used to designate the reference program implementation of an algorithm submitted for publication in an IPOL article. An IPOL article **may** publish more than one program, an IPOL demo **may** use more than one program.

In this document, the words **must**, **must not**, **should**, **should not**, **recommended**, and **may** are used to express required, recommended, and optional items. Their interpretation is described in IETF RFC2119[4] and detailed in the context of these guidelines in Annex A.

Guidelines

1. Packaging and Content

1.1. Compressed Archive

An IPOL program **must** be packaged as a compressed archive file. This file archive can either be a single volume .ZIP compressed archive or a GZIP compressed tar archive[5]. The size of the compressed archive file **should** be less than 2 MB. In the remainder of this document, we will use the terms "zip archive" and "tar/gzip archive" for convenience.

Annex B of this document provides some examples of programs that can be used to produce such compressed archives.

[2] `http://www.ipol.im/`

[3] `http://tools.ipol.im/mailman/listinfo/discuss`

[4] `http://tools.ietf.org/html/rfc2119`

[5] These file formats are defined by the PKZIP APPNOTE specification[6], version 6.3.2, for the .ZIP compressed archive format, the IETF RFC1952[7] for the GZIP compressed format, and the POSIX.1 ustar definition[8] for the tar archive format.

1.2. Archive Name, Program Name and Version

The compressed archive file of an IPOL program **must** be named according to the `name_version.extension` pattern, where:

- `name` and `version` **must** consist only of lower case letters (`a-z`), digits (`0-9`), minus (`-`) and period (`.`) signs;
- `name` **must** be at least two characters long and start with a letter; it **must** indicate the name of the program; this name can be the name of the executable program file, or another name, at the author's will;
- `version` **must** start with a digit; it **must** indicate a version number for the program, in the sense that two different releases of the program **must** have two different version numbers; if no version numbering scheme is established for the program, the `YYYYMMDD` pattern based on the year, month and day of the release date **may** be used;
- `extension` **must** be `zip` for zip archives and `tar.gz` or `tgz` for tar/gzip archives.

1.3. File and Folder Names

All the files and folders extracted from the compressed archive **must** be located inside a base folder named `name_version`, where `name` and `version` are identical to those used for the compressed archive file name. Absolute path **must not** be used for files and folders extracted from the archive.

The name of all files and folders composing the IPOL program **must** consist only of lower or upper case letters (`a-z`, `A-Z`), digits (`0-9`), minus (`-`), underscore (_) and period (`.`) signs. They **should** start with a letter.

The names **should** provide a meaningful hint of the content of these files and folders.

1.4. Hidden and Useless Files

An IPOL program **should not** include hidden files or folders or by-products of the tools used by the authors, such as (but not limited to):

- files inserted by file managers (`.DS_Store`, `.directory`);
- folders inserted by version control managers (`.svn`, `.git`);
- backup versions (`filename~`, `filename.bak`).

The program **should not** be distributed with files not useful to build, use or study the implementation of the algorithm published in IPOL.

2. Implementation

2.1. Source Code

An IPOL program **must** include all the material necessary to build one or more executable program files implementing the algorithm published in IPOL. This material **must** be provided in human-readable source code form. An IPOL program **must not** be distributed with binary precompiled files if these files can be obtained from source code[9].

Annex C provides some information for IPOL authors to help them perform various frequent implementation tasks.

2.2. Programming Language

The source code of an IPOL program **must** follow the published standard syntax of one or more compiled programming languages. IPOL can currently only process C89 (ANSI C), C99, and C++98 (ISO C++)[10]. If the authors want to publish their program with another well-known and standardized compiled language (such as Fortran 90), they should contact the editorial board to investigate the possibilities.

IPOL authors **should** test their C and C++ source code with the `gcc` compiler in strict compilation mode[14] before submitting it to IPOL.

The source code **may** use the OpenMP 3.0[15] API for shared multiprocessing programming (parallel programming) but it **must** also compile and provide the same results (albeit slower) without OpenMP. Usage of OpenMP **must not** be tied to a specific number of processors and **must** only rely on the OpenMP standard, not on any vendor implementation.

2.3. Portability

The source code of an IPOL program **must not** require any extension of the language or its standard library, or any resource specific to a hardware environment, operating system or compiler. These extensions and resources **may** be used to achieve better performances if they are available but their availability **must** be detected during the compilation or execution and an alternative portable implementation **must** be used in their absence. This includes (but is not limited to)

- language dialects specific to a compiler (GNU C, Microsoft C);
- standard library functions specific to an implementation (`drand48()`);
- assembler code or "intrinsic functions" mapped to a processor instruction (`mulps`, `__builtin_ia32_mulps()`);

[9] If the authors want to distribute binary versions, they can do it in IPOL but not via the compressed archive of the IPOL program.

[10] C89 is defined by the ANSI X3.159–1989 Programming Language C standard[11], C99 is defined by the ISO/IEC 9899:1999 Programming languages — C standard[12], and C++98 is defined by the ISO/IEC 14882:1998 Programming languages — C++ standard[13].

[14] C89, C99 and C++98 code can be tested with `gcc -std=xxx -Wall -Wextra -Werror` where xxx is `c89`, `c99` or `c++98`.

[15] `http://www.openmp.org/mp-documents/spec30.pdf`

- operating system calls (Win32 `GetSystemTime()`, POSIX `gettimeofday()`);
- file system locations (`C:\Documents and Settings`, `/tmp`);
- code specific to a memory model environment (32-bits, 64-bits).

Special attention will be given to the Linux 64-bit environment because it is currently the primary environment for IPOL demonstrations. But the program **must** also be usable in other environments and this portability **must not** be limited to the Win32/POSIX alternative.

2.4. Dependencies

An IPOL program **must** not use external software components except for the libraries and APIs listed hereafter. The program **may** expect these software components to be correctly installed and configured during the compilation and execution:

- `libtiff`[16] version 3.x and `libpng`[17] version 1.4.x to read and write files, with their dependencies `libjpeg`[18] version 8.x, `zlib`[19] version 1.2.x;
- `fftw`[20] version 3.x (single and double precision) for Fourier transforms;
- `cblas`[21] and `clapack`[22] for linear algebra.

Other libraries can be examined and may be added to this list on request, if they are portable, widely used, with a stable API.

This restriction only applies to software components used by the IPOL program but not distributed in source form with the program. Annex C has more details about how some external code, including external library code, can be used in the IPOL program.

2.5. Compilation

An IPOL program **must** be compiled by an automated non-interactive build procedure with `make` or `cmake`. This build tool **must not** be configured to use any special compiler.

For example, `make` **must not** call `gcc` or `g++` but **must** use the `$(CC)` and `$(CXX)` variables instead. The default build procedure **must** use standard compiler options only[23] : `-c`, `-D`, `-E`, `-I`, `-L`, `-l`, `-O`, `-o` and `-U`.

[16] `http://www.remotesensing.org/libtiff/`
[17] `http://libpng.org/pub/png/libpng.html`
[18] `http://www.ijg.org/`
[19] `http://zlib.net/`
[20] `http://www.fftw.org/`
[21] `http://www.netlib.org/blas/`
[22] `http://www.netlib.org/lapack/`
[23] Standard compiler options are defined by the POSIX c99 specification[24]. Other options depending on the environment (hardware, operating system, compiler) **may** also be provided, for example for an optimized compilation, but they **must not** be used in the default build procedure.

2.6. Usage and Input/Output

An IPOL program **should** be minimal and only perform the algorithm published in IPOL. It **must** be usable from the command line environment without any user interaction, taking all its parameters from the command line.

An IPOL program **must** be able to read the input data and write the final output data in at least one of these formats[25]: PNG, TIFF or PNM for raster images, EPS or SVG for vector images, VRML or PLY for meshes, and plain text for other data. Other file formats can be added to this list on request, if they are clearly defined and widely used.

Annex C provides some information for IPOL authors to help them use external libraries to read and write images.

2.7. Computing Resources

In the demo environment, an IPOL program **should not** use more than 1 GB of memory and **must not** use more than 8 MB of stack memory space (for recursion, local variables and variable-length arrays). The program **must not** need more than 30 seconds to process typical data. For slow algorithms, this limit **may** be achieved with parallel processing or a limit on input size.

Annex C provides some information for IPOL authors to help them improve the performance of their implementation.

3. Copyright, License and Patents

3.1. Copyright Attribution

Every source code file in an IPOL program **must** mention its authors in a copyright attribution line at the top of the file. This mention **may** be omitted in very simple files such as header code.

Every person whose contribution to this file is not trivial and implies some creative work **must** be credited. Of course, if the authors use or modify a file previously written by other persons, the copyright attribution to the previous authors **must not** be removed. The copyright attribution **must** include the years of production of the work, the full name and an e-mail address for contributor. It **may** also include other relevant information such as the employer, affiliation or web site.

An simple example for a single author can be:

```
Copyright (C) 2011, Jane Doe <jane.doe@example.org>
```

A complex example with many contributors can be:

[25] PNG is defined by the IETF RFC2083[26], TIFF is defined by the Adobe TIFF 6.0 Specification[27], PNM (PBM, PGM and PPM) is defined by the netpbm documentation[28], EPS is defined by the Adobe Encapsulated PostScript 3.0 Specification[29], SVG is defined by the W3C Scalable Vector Graphics 1.0 Specification[30], and VRML is defined by the ISO/IEC Virtual Reality Modeling Language Specification[31]. There is no formal published specification of PLY, but this simple format introduced by the Standford 3D Scanning Repository[32] is documented on Paul Bourke's site[33]. Plain text output **should** be understandable by a human reader and easy to parse with a software.

```
Copyright (C) 1998-2003, Taro Yamada <taro.yamada@example.jp>
Copyright (C) 2005-2011, Juan Perez <juan.perez@example.es>
Copyright (C) 2011,      Marie Untel, ENS Cachan
                         <marie.untel@ens-cachan.fr>
```

3.2. Patent Warning

When the authors are aware or suspect that a source code file implements an algorithm which might be linked to a patent (the main algorithm published on IPOL or another algorithm used for this implementation), a patent warning **must** be inserted after the copyright attribution, in every file potentially linked to this patent. This wording is **recommended**:

```
This file implements an algorithm possibly linked to the patent
<REFERENCE OF THE PATENT>.
This file is made available for the exclusive aim of serving as
scientific tool to verify the soundness and completeness of the
algorithm description. Compilation, execution and redistribution
of this file may violate patents rights in certain countries.
The situation being different for every country and changing
over time, it is your responsibility to determine which patent
rights restrictions apply to you before you compile, use,
modify, or redistribute this file. A patent lawyer is qualified
to make this determination.
If and only if they don't conflict with any patent terms, you
can benefit from the following license terms attached to this
file.
```

3.3. License

Every source code file **must** mention a usage and redistribution license after the copyright attribution (and patent warning for algorithms potentially linked to a patent). Of course, if the authors use or modify a file previously written by other persons, the license chosen by the previous authors **must not** be modified.

- When the authors are not aware of a possible patent issue, the license **must** be a free software license of the GPL[34]/LGPL[35]/AGPL[36] or BSD[37] type.
- When a source code file can be linked to a patented algorithm and the source code authors are not the patent inventors, the file **must** be distributed under the BSD license.
- When a source code file can be linked to a patented algorithm and the authors of the source code are the patent inventors, the file **must** be distributed either under the BSD license or "for research and education only"[38].

[34] `http://www.gnu.org/licenses/gpl.html`
[35] `http://www.gnu.org/licenses/lgpl.html`
[36] `http://www.gnu.org/licenses/agpl.html`
[37] `http://www.opensource.org/licenses/bsd-license.php`
[38] Distribution "for research and education" can help avoid conflicts between patent rights and software license when the patent inventors are the source code authors. It is not needed in other situations because the validity of the license will depend on the local patent regulations, as stated in the last sentence of the patent warning.

These wordings are **recommended**:

```
This program is free software: you can use, modify and/or
redistribute it under the terms of the GNU General Public
License as published by the Free Software Foundation, either
version 3 of the License, or (at your option) any later
version. You should have received a copy of this license along
this program. If not, see <http://www.gnu.org/licenses/>.
```

```
This program is free software: you can use, modify and/or
redistribute it under the terms of the simplified BSD
License. You should have received a copy of this license along
this program. If not, see
<http://www.opensource.org/licenses/bsd-license.html>.
```

```
This program is provided for research and education only: you can
use and/or modify it for these purposes, but you are not allowed
to redistribute this work or derivative works in source or
executable form. A license must be obtained from the patent right
holders for any other use.
```

The exact terms can differ, for example when a different GPL/LGPL/AGPL license version is chosen. The full text of the license **must** be included in a separate file with the source code.

IPOL authors **should** verify that the usage of these licenses for their software publications complies with their employer policy and local situation and jurisdiction.

4. Documentation

4.1. README.txt

Every IPOL program **must** provide a file named `README.txt` in the base folder and written in plain text and in English. This `README.txt` file **must** include the following essential information, in any order:

- name and brief description of the program
- reference to the IPOL article
- authors and contact information
- version number and release date
- location of future releases and updates
- copyright, patent and license information
- tools and libraries needed to compile and use the program
- compilation instructions
- usage instructions and example

- changes in the program since it was first published in IPOL

This `README.txt` file **may** contain other information, and **may** also be completed by another documentation, possibly with more details, in text, PDF, HTML or any other format.

For a simple code, the license information in `README.txt` can be

```
This program is written by Jane Doe <jane.doe@example.org> and
distributed under the terms of the GPLv3 license.
```

A complex case (multiple authors, patents and licenses) can be:

```
This program is written by Taro Yamada <taro.yamada@example.jp>
and Juan Perez <juan.perez@example.es> with contributions from
Marie Untel, ENS Cachan <marie.untel@ens-cachan.fr>.
- mmatch.c and rot_tree.c may be linked to the pending EU patent
  123.456 by Taro Yamada and Juan Perez and are provided for
  scientific and education only.
- demoz.c may be linked to the US patent 65.43.21 by Jane Doe;
  see the file for license terms.
- eizo.c and linalg_lib.c are distributed under the terms
  of the BSD license.
- All the other files are distributed under the terms of the
  LGPLv3 license.
```

4.2. Readability

In an IPOL program, the source code is a primary material for the publication. It will be reviewed, published and read like any other part of the article. The authors **must** take care of the clarity of their program.

The source code of an IPOL program **must** be consistently indented and spaced. Lines **should** be limited to 80 characters and **should not** end with blank characters (spaces, tabs, ...). Files **should not** have more than 1000 lines. The line terminations **should** be the same (DOS/Windows `CRLF` or UNIX `CR` style) for all the files of the program.

Functions **should** be grouped by abstraction level in different source code files:

- the `main()` function, command-line processing and input/output calls in one file,
- the implementation of the algorithm described and reviewed in the IPOL article in one or more other files,
- and the implementation of auxiliary and external routines in one or more other files.

Annex D provides some examples of programs that can be used to improve the indentation, spacing and presentation of a source code.

4.3. Implementation and Comments

The source code of an IPOL program **must** be commented precisely and exhaustively. Authors **should** target the "1/8 comment/instruction ratio", but the quality of the comments is more important than the quantity. The source code **must** be written in English, including all variables, functions names and comments.

Authors **must** ensure that the code is understandable, to the satisfaction of the editor and reviewers, so that consistency between the description of the algorithm and its implementation can be verified. The relation between each part of the implementation and the respective part of the description of the algorithm **must** be explained in the comments.

Authors **should** apply simpler implementations when available, follow the conventions of the programming language, and use comments to explain implementation choices and every complicated or subtle point in the program. Clarity is more important than virtuosity.

Every function **must** be documented with at least one line explaining what the function is doing, and the meaning of its parameters and return value.

The Doxygen[39] source code documentation format is **recommended** for every IPOL Program.

Annex D provides some examples of programs that can be used to count the comment, instruction and blank lines.

4.4. Example Data

The authors **should** provide an example of input file to test the IPOL program and the result to expect when this input file is processed by the program.

Annexes

The annexes are not part of the guidelines. They are provided to help authors and editors follow the guidelines.

A. Key Words

The key words **must**, **must not**, **required**, **shall**, **shall not**, **should**, **should not**, **recommended**, **may**, and **optional** in this document are to be interpreted as described in IETF RFC2119[40].

must This word, or the terms **required** or **shall**, mean that the definition is an absolute requirement.

must not This phrase, or the phrase **shall not**, mean that the definition is an absolute prohibition.

[39] `http://www.stack.nl/\protect$\relax\sim$dimitri/doxygen/`
[40] `http://tools.ietf.org/html/rfc2119`

No article will be published in IPOL if a program included in this article doesn't follow such requirements or prohibitions.

should This word, or the adjective **recommended**, mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.

should not This phrase, or the phrase **not recommended** mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

The rationale for not following such recommendations **must** be agreed by the authors, reviewers and editor before an article is accepted for publication.

may This word, or the adjective **optional**, mean that an item is truly optional. One author may choose to include the item because a particular article requires it or because the author feels that it enhances the software while another author may omit the same item.

B. Compression and Archive Tools

IPOL authors can consider that the files produced by the following tools are correct zip and tar/gzip archives:

- on Linux systems, the `tar` (usually "GNU tar"), `zip` (usually "Info-Zip zip") and `gzip` programs;
- on Mac OS X systems, the "Create Archive" feature of the graphical interface and the `tar` (usually "BSD tar") and `gzip` programs;
- on Windows systems, the "Compressed Folder" feature of the graphical interface and the `7-zip`[41] program.

C. Coding Help

The authors of the IPOL article do not need to be the authors of all the implementation of their algorithm. They can use their own code or code from other IPOL programs or other software projects and libraries, or a combination of these options. This is encouraged when it helps improve the quality of the implementation. All the source code must follow the guidelines (be standard, portable, readable and documented), regardless of its origin. The original copyrights and licenses of the reused code parts must of course be respected.

Some help to read and write image files is provided in the IPOL wiki[42] with simplified interfaces to `libpng` and `libtiff` and some examples. IPOL authors can also find in this wiki a list of contributions and tools[43] from IPOL authors willing to share.

[41] `http://www.7-zip.org/`
[42] `http://tools.ipol.im/wiki/author/code/tools/`
[43] `http://tools.ipol.im/wiki/author/code/hatchery/`

The IPOL editorial board can provide some help to the authors to accelerate their code and guide them for performance profiling and parallel programming. The IPOL discussion list[44] is the good place for any questions related to IPOL, including the implementation work.

D. Source Code Tools

IPOL authors can use these programs to improve the presentation and clarity of their source code: `indent`[45] (C, Linux), `uncrustify`[46] (C/C++, Linux and Windows), `astyle`[47] (C/C++, Linux, Mac OS X and Windows) and `UniversalIndentGUI`[48] (cross-platform graphical frontend).

Tools like `cloc`[49], `ohcount`[50] and `sloccount`[51] can be used to count the comments, instructions and blank lines and evaluate the comment/instruction ratio.

[44] `http://tools.ipol.im/mailman/listinfo/discuss`
[45] `http://www.gnu.org/software/indent/`
[46] `http://uncrustify.sourceforge.net/`
[47] `http://astyle.sourceforge.net/`
[48] `http://universalindent.sourceforge.net/`
[49] `http://cloc.sourceforge.net/`
[50] `http://ohcount.sourceforge.net/`
[51] `http://sloccount.sourceforge.net/`

Chapter 10

Annex 2: Feedback Survey

Contents

The questions of the IPOL author survey are reproduced hereafter, after adaptation into a printable form. A supplement survey, with only one question addded after the analysis of the results of the main survey, in in the Annex B.

Author Feedback

If you have been involved in more than one IPOL article, please refer to most representative one to answer the questions.

Are you publishing in IPOL ...

- an original algorithm, never published before
- an algorithm you already published in a "classic" journal
- an algorithm already described in a "classic" journal by other authors, with your implementation
- an algorithm already described in a "classic" journal by other authors, with their implementation

For each item, answer "yes" or "no".

How much work is needed to write an article for IPOL, if compared with an article for a "traditional" journal?

This only includes the redaction of the text of the article, not the work on the software.

- $< \times 0.25$
- $\times 0.25$
- $\times 0.5$
- $\times 1$
- $\times 2$
- $\times 4$
- $> \times 4$

If writing an article for a traditional journal takes 1 week and writing an article on the same subject for IPOL takes 2 week, then the answer is $\times 2$.

How much work is needed to produce an implementation adapted to the IPOL requirements, if compared with a development only targeting "classic" publications?

- $\times 1$
- $\times 1.5$
- $\times 2$
- $\times 4$
- $> \times 4$

If a software was developed in 2 weeks and needed 1 more week to be adapted to IPOL, then the answer is "$\times 1.5$".

Are these restrictions on IPOL software a problem for you?

- code in standard C/C++
- complete code required
- portable (Win/Mac/Linux) implementation
- limited list of libraries
- usable in command-line
- few file formats allowed
- documentation and comments requirements

For each item, chose an answer in the following list:

1. yes, it is a big problem and requires a lot of extra work
2. yes, it is a problem
3. it has an impact on my work, but it is not an obstacle
4. no, I can adapt, I don't mind
5. no, absolutely no impact on my software

Which license did you use for your IPOL software?

- GPL/LGPL/AGPL
- BSD
- for research only

Is your IPOL article linked to a patented algrithm?

- yes, and I am the patent holder
- yes, but I am not the patent holder
- no, I have not heard of such patent

About the demo archive of your IPOL article...

- Do you regularly look at the archive?
- Did it reveal some properties of the algorithm?

For each item, answer "yes" or "no".

Are you satisfied with ...

- the edition interface for the articles (wiki-style web system)?
- the edition language for the articles (markdown)?
- the article format (web page)?
- the documentation and information about the review process?
- the speed of the review process?

For each item, choose an answer in a 1–5 scale between "1 - yes, very satisfied" and "5 - no, very dissatisfied".

Would you prefer...

- a web review interface instead of mails?
- a system to send your article instead of editing directly IPOL?
- to use LaTeX to write your IPOL articles?
- to publish the articles as PDF files?

For each item, answer "yes" or "no" or "uncertain".

Would you...

- cite an IPOL article in your research papers?
- suggest a colleague to visit IPOL?
- publish another article in IPOL?
- suggest a colleague to publish an article in IPOL?

For each item, answer "yes" or "no".

Please order these propositions of IPOL developments, the highest priority first

- better demonstration system and tools
- better demonstration archives
- better design of the ipol web site
- better edition interface
- better tools for the review process
- source code tools and libraries
- other data types (sound, video, ...)
- better indexation in scientific journal databases
- reader feedback (comments, forum) for every article

How would you qualify your IPOL experience as an author?

Choose an answer in a 1–5 scale between "1 - very positive" and "5 - very negative".

General Information

What is your current status?

- Master Student
- PhD Student
- Post-doc
- Junior Researcher
- Senior Researcher
- Other

What is the main operating system you use for tasks related to your research activity?

- Windows
- Mac OS X
- Linux
- Other

What is the main programming language you use for tasks related to your research activity?

- Java
- C
- C++
- Python
- MATLAB
- Fortran
- Other

What is the main browser you use for tasks related to your research activity?

- Internet Explorer
- Firefox
- Chrome
- Opera
- Other

Supplement Survey

This is a supplement to the "IPOL Author Feedback 2011" survey.

Almost every author rated their IPOL experience "positive" to "very positive" in the feedback survey. We would like to know more.

Why are you satisfied with IPOL? Why is it positive, and not just "more work than a normal paper"? What were the benefits of working on an IPOL paper? And if your experience is not positive, you can comment too.

Chapter 11

References

Bibliography

[1] Loi sur la liberté de la presse, 1881.
URL: http://www.legifrance.gouv.fr/affichTexte.do?cidTexte=LEGITEXT000006070722.

[2] Loi relative à l'informatique, aux fichiers et aux libertés, 1978.
URL: http://www.legifrance.gouv.fr/affichTexte.do?cidTexte=LEGITEXT000006068624.

[3] Loi sur la communication audiovisuelle, 1982.
URL: http://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000000880222.

[4] Loi relative à la liberté de comommunication, 1986.
URL: http://www.legifrance.gouv.fr/affichTexte.do?cidTexte=LEGITEXT000006068930.

[5] Loi sur la fraude informatique, 1988.
URL: http://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000000875419.

[6] Loi portant adaptation du droit de la preuve aux technologies de l'information et relative à la signature électronique, 2000.
URL: http://www.legifrance.gouv.fr/affichTexte.do?cidTexte=LEGITEXT000005629200.

[7] Loi relative à la sécurité quotidienne, 2001.
URL: http://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000000222052.

[8] Loi d'orientation et de programmation pour la sécurité intérieure, 2002.
URL: http://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000000780288.

[9] Loi sur la sécurité intérieure, 2003.
URL: http://www.legifrance.gouv.fr/affichTexte.do?cidTexte=LEGITEXT000005634107.

[10] Loi pour la confiance dans l'économie numérique, 2004.
URL: http://www.legifrance.gouv.fr/affichTexte.do?cidTexte=LEGITEXT000005789847.

[11] Loi relative au droit d'auteur et aux droits voisins dans la société de l'information (dadvsi), 2006.
URL: http://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000000266350.

[12] Loi relative à la lutte contre le terrorisme et portant dispositions diverses relatives à la sécurité et aux contrôles frontaliers, 2006.
URL: http://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000000454124.

[13] Loi favorisant la diffusion et la protection de la création sur internet, 2009.
URL: http://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000020735432.

[14] Loi relative à la protection pénale de la propriété littéraire et artistique sur internet, 2009.
URL: http://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000021208046.

[15] Loi d'orientation et de programmation pour la performance de la sécurité intérieure, 2011.
URL: http://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000023707312.

[16] 3tu.datacentrum.
URL: http://datacentrum.3tu.nl/.

[17] Loi tendant à valoriser l'activité inventive et à modifier le régime des brevets d'invention, 1968.
URL: http://legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000000317285.

[18] A. Buades, B. Coll and J.-M. Morel. A review of image denoising algorithms, with a new one. *Multiscale Modeling and Simulation*, 2006.
DOI: http://dx.doi.org/10.1137/040616024.

[19] Janet Abbate. *Inventing the Internet*. MIT Press, 1999.
ISBN: 0262511150.

[20] A. Almansa, A. Desolneux, and S. Vamech. Vanishing point detection without any a priori information. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 2003.

[21] Luis Alvarez, Luis Gomez, and J. Rafael Sendra. Algebraic Lens Distortion Model Estimation. *Image Processing on Line*, 2010.
DOI: http://dx.doi.org/10.5201/ipol.2010.ags-alde.

[22] Apple Inc. iOS ABI Function Call Guide, 2010.
URL: http://developer.apple.com/library/ios/documentation/Xcode/Conc%eptual/iPhoneOSABIReference/iPhoneOSABIReference.pdf.

[23] Apple Inc. OS X Lion Technical Specifications, 2011.
URL: http://www.apple.com/macosx/specs.html.

[24] Apple Inc. Software License Agreement for Mac OS X [Lion], 2011.
URL: http://images.apple.com/legal/sla/docs/macosx107.pdf.

[25] Apple Inc. Software License Agreement for Mac OS X Lion Server, 2011.
URL: http://images.apple.com/legal/sla/docs/macosxserver107.pdf.

[26] Antoine Aubert and Frank Macrez. "Brevet de logiciel" : quelle portée ?, 2001.
URL: http://www.droit-ntic.com/pdf/brevetlog.pdf.

[27] Simon Baker, Daniel Scharstein, J. P. Lewis, Stefan Roth, Michael Black Black, and Richard Szeliski. Flow accuracy and interpolation evaluation.
URL: http://vision.middlebury.edu/flow/eval/.

[28] D.H. Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern recognition*, 1981.

[29] C. Ballester, V. Caselles, and P. Monasse. The tree of shapes of an image. *ESAIM: Control, Optimisation and Calculus of Variations*, 2003.

[30] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *SIGOPS Operating Systems Review*, 2003.
DOI: http://dx.doi.org/10.1145/1165389.945462.

[31] Kobus Barnard, Brian Funt, and Adam Coath. A data set for colour research. *Color Research and Application*, 2002.
DOI: http://dx.doi.org/10.1002/col.10049,
URL: http://www.cs.sfu.ca/~colour/data/colour_constancy_test_images/.

[32] N. Barnes. Publish your computer code: it is good enough. *Nature*, 2010.
DOI: http://dx.doi.org/10.1038/467753a.

[33] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. *Computer Vision–ECCV 2006*, 2006.

[34] Gavin Bell, Anthony Parisi, and Mark Pesce. The Virtual Reality Modeling Language — Version 1.0 Specification, 1995.
URL: http://www.web3d.org/x3d/specifications/vrml/.

[35] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *Visualization and Computer Graphics, IEEE Transactions on*, 1999.

[36] Berne Convention for the Protection of Literary and Artistic Works, 1886. 1908 Berlin Revision, archived by the Internet Archive.
URL: http://www.archive.org/details/internationalco00offigoog.

[37] Berne Convention for the Protection of Literary and Artistic Works, 1886. 1971 Paris revision, archived by the Cornell University Legal Information Institute.
URL: http://www.law.cornell.edu/treaties/berne/overview.html.

[38] T. Berners-Lee. Tags used in html, 1992. archived by the World Wide Web Consortium.
URL: http://www.w3.org/History/19921103-hypertext/hypertext/WWW/MarkUp/Tags.html.

[39] T. Berners-Lee and D. Connolly. IETF draft: Hypertext Markup Language (HTML), 1993. archived by the World Wide Web Consortium.
URL: http://www.w3.org/MarkUp/draft-ietf-iiir-html-01.txt.

[40] T. Berners-Lee and D. Connolly. IETF RFC1866 — Hypertext Markup Language - 2.0, 1995.
URL: http://tools.ietf.org/html/rfc1866.

[41] Tim Berners-Lee. Answers to young people — What made you think of the WWW?, 2012.
URL: http://www.w3.org/People/Berners-Lee/Kids.html.

[42] Timothy Berners-Lee. Information management: a proposal. Technical Report CERN-DD-89-001-OC, CERN, 1989.
URL: http://www.w3.org/History/1989/proposal.html.

[43] Timothy Berners-Lee. The original http as defined in 1991, 1991.
URL: http://www.w3.org/Protocols/HTTP/AsImplemented.html.

[44] Timothy Berners-Lee and Robert Cailliau. WorldWideWeb: Proposal for a hypertexts Project, 1990.
URL: http://www.w3.org/Proposal.html.

[45] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. Image inpainting. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 2000.

[46] P. Bhat, B. Curless, M. Cohen, and C. Zitnick. Fourier analysis of the 2d screened poisson equation for gradient domain problems. *Computer Vision–ECCV 2008*, 2008.

[47] Andrew Binstock. Interview with Herb Sutter. *Dr. Dobb's*, 2011.
URL: http://drdobbs.com/cpp/231900562.

[48] G. Blanchet, A. Buades, B. Coll, JM Morel, and B. Rouge. Fattening free block matching. *Journal of Mathematical Imaging and Vision*, 2011.

[49] Budapest open access initiative.
URL: http://www.soros.org/openaccess/read.

[50] R.C. Bolles, H.H. Baker, and D.H. Marimont. Epipolar-plane image analysis: An approach to determining structure from motion. *International Journal of Computer Vision*, 1987.

[51] Jeff Bonwick. ZFS Deduplication. Jeff Bonwick's Blog at Oracle, 2009.
URL: http://blogs.oracle.com/bonwick/en_US/entry/zfs_dedup.

[52] J.Y. Bouguet. Camera calibration toolbox for matlab, 2004.
URL: http://www.vision.caltech.edu/bouguetj/calib_doc/.

[53] T.E. Boult and G. Wolberg. Correcting chromatic aberrations using image warping. In *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR'92., 1992 IEEE Computer Society Conference on*, 1992.

[54] Paul Bourke. PLY - Polygon File Format.
URL: http://paulbourke.net/dataformats/ply/.

[55] T. Boutell. IETF RFC2083 — PNG (Portable Network Graphics) Specification Version 1.0, 1997.
URL: http://tools.ietf.org/html/rfc2083.

[56] Gabriel Bouvigne. Patents and mp3, 2002.
URL: http://www.mp3-tech.org/patents.html.

[57] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 2004.

[58] S. Bradner. IETF RFC2119 — Key words for use in RFCs to Indicate Requirement Levels, 1997.
URL: http://tools.ietf.org/html/rfc2119.

[59] Grant R. Brammer, Ralph W. Crosby, Suzanne J. Matthews, and TiffaniL. Williams. Paper Mâché: Creating Dynamic Reproducible Science. In *Proceedings of the International Conference on Computational Science*, 2011.
DOI: http://dx.doi.org/10.1016/j.procs.2011.04.069.

[60] Robert Bringhurst. *The Elements of Typographic Style*. Hartley & Marks, 2nd edition, 2002.
ISBN: 0881791326.

[61] Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition*. Addison-Wesley Professional, 2nd edition, 1995.
ISBN: 0201835959.

[62] Mark Brown. Fastcgi specification, 1996.
URL: http://www.fastcgi.com/devkit/doc/fcgi-spec.html.

[63] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. Non-local Means Denoising. *Image Processing On Line*, 2011.
DOI: http://dx.doi.org/10.5201/ipol.2011.bcm_nlm.

[64] Antoni Buades, Bartomeu Coll, Jean-Michel Morel, and Catalina Sbert. Self-Similarity Driven Demosaicking. *Image Processing on Line*, 2011.
DOI: http://dx.doi.org/10.5201/ipol.2011.bcms-ssdd.

[65] Antoni Buades, Triet Le, Jean-Michel Morel, and Luminita Vese. Cartoon+Texture Image Decomposition. *Image Processing on Line*, 2011.
DOI: http://dx.doi.org/10.5201/ipol.2011.blmv_ct.

[66] T. Buades, Y. Lou, JM Morel, and Z. Tang. A note on multi-image denoising. In *Local and Non-Local Approximation in Image Processing, 2009. LNLA 2009. International Workshop on*, 2009.

[67] Jonathan B. Buckheit and David L. Donoho. Wavelab and reproducible research. Technical Report 474, Department of Statistics, Stanford University, 1995.
URL: http://www-stat.stanford.edu/~donoho/Reports/1995/wavelab.pdf.

[68] Bundesgerichtshof. Rentabilitätsermittlung, 2004. BGH X ZB 34/03.

[69] Bundesgerichtshof. Informationsübermittlungsverfahren, 2007. BGH X ZB 9/06.

[70] Vannevar Bush. As we may think. *The Atlantic Monthly*, 1945.
URL: http://www.theatlantic.com/magazine/archive/1945/07/as-we-may-think/3881/4/.

[71] Murielle Cahen. Responsabilité des forums de discussion.
URL: http://www.murielle-cahen.com/publications/p_forum.asp.

[72] E.J. Candès, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *Information Theory, IEEE Transactions on*, 2006.

[73] J. Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 1986.

[74] Jaime Carbonell. On man-computer interaction: A model and some related issues. Scientific Report No. 1, DARPA Project 8668, 1967.
URL: http://handle.dtic.mil/100.2/AD0666666.

[75] V. Caselles, R. Kimmel, and G. Sapiro. Geodesic active contours. *International journal of computer vision*, 1997.

[76] Davide P. Cervone. jsmath: A method of including mathematics in web pages, 2007.
URL: http://www.math.union.edu/~dpvc/jsMath/.

[77] Davide P. Cervone. Mathjax: a javascript-based engine for including tex and mathml in html, 2010.
URL: http://www.math.union.edu/~dpvc/talks/2010-01-15.mathjax/.

[78] T.F. Chan and C.K. Wong. Total variation blind deconvolution. *Image Processing, IEEE Transactions on*, 1998.

[79] Laurent Chemla. *Confessions d'un Voleur*. Denoël, 2002.
ISBN: 2207252167,
URL: http://www.confessions-voleur.net/.

[80] Jon Claerbout and Martin Karrenbach. Electronic documents give reproducible research a new meaning. 1992. Proceedings of the 62nd Annual International Meeting of the Society of Exploration Geophysics.

[81] CNRS. Propriété intellectuelle - Logiciels, 2010.
URL: http://www.dgdr.cnrs.fr/daj/propriete/logiciels/logiciels.htm.

[82] R.R. Coifman and D.L. Donoho. Translation-invariant de-noising. *Lecture Notes in Statistics*, 1995.

[83] Gilbert Colletaz, Christophe Hurlin, Christophe Pérignon, and Yvan Stroppa. Runmycode — la recherche académique en économie et gestion à portée de clic. *La lettre de l'INHSH*, 2012.
URL: http://www.cnrs.fr/inshs/Lettres-information-INSHS/lettre_infoINSHS_15.pdf.

[84] ISO/IEC JTC1/SC22/WG21 The C++ Standards Committee. 14882:1998 Programming languages — C++, 1998. unofficial archive.
URL: http://www.kuzbass.ru:8086/docs/isocpp/.

[85] P. Comon. Independent component analysis, a new concept? *Signal processing*, 1994.

[86] Neil G. Connelly, Ture Damhus, Hartshorn Richard M., and Alan T. Hutton, editors. *Nomenclature of Inorganic Chemistry: Recommendations 2005*. Royal Society of Chemistry, 2005.
ISBN: 0854044388,
URL: http://old.iupac.org/publications/books/rbook/Red_Book_2005.pdf.

[87] Stéphane Cordier, Konrad Hinsen, Christophe Hurlin, and Frédéric Loulergue. Rencontre de réflexion autour de la recherche reproductible, 2012.
URL: http://www.fdpoisson.fr/cascimodot/doc/RRRR/R4-050412.php.

[88] Oracle Corporation. Virtualbox website, 2011.
URL: https://www.virtualbox.org/.

[89] Cour d'appel de Paris. S.A. SAGEM c./ M. le directeur de l'INPI, 2003.
URL: http://www.softwarepatentnews.de/pdf/ca_paris_1.pdf.

[90] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian. Image denoising by sparse 3-D transform-domain collaborative filtering. *Image Processing, IEEE Transactions on*, 2007.

[91] Ronald S. Laurie Daniel Lin, Matthew Sag. Source code versus object code: Patent implications for the open source community. *Santa Clara Computer & High Technology Law Journal*, 2002.
URL: http://www.chtlj.org/sites/default/files/media/articles/v018/v018.i2.Lin.pdf.

[92] Data archiving and networked services.
URL: http://www.dans.knaw.nl/.

[93] Philip M. Davis. Does open access lead to increased readership and citations? A randomized controlled trial of articles published in APS journals. *The Physiologist*, 2010.
URL: http://view.ncbi.nlm.nih.gov/pubmed/21473414.

[94] Pablo Delbracio, Mauricio anmd Musé and Andrès Almansa. Non-parametric sub-pixel local point spread function estimation. *Image Processing on Line*, 2012.
DOI: http://dx.doi.org/10.5201/ipol.2012.admm-nppsf.

[95] J. Delon, A. Desolneux, J.L. Lisani, and A.B. Petro. A nonparametric approach for histogram segmentation. *Image Processing, IEEE Transactions on*, 2007.

[96] Forum des droits sur l'internet. Pas de cadeaux pour les opposants au père-noël, 2002.
URL: http://www.foruminternet.org/specialistes/veille-juridique/actualites/pas-de-cadeaux-pour-les-opposants-au-pere-noel.html.

[97] A. Desolneux, L. Moisan, and J.M. Morel. Edge detection by helmholtz principle. *Journal of Mathematical Imaging and Vision*, 2001.

[98] Agnès Desolneux, Lionel Moisan, and Jean-Michel Morel. *From Gestalt Theory to Image Analysis: A Probabilistic Approach*. Springer-Verlag, 2008.
ISBN: 0387726357.

[99] P. Deutsch. IETF RFC1952 — GZIP file format specification version 4.3, 1996.
URL: http://tools.ietf.org/html/rfc1952.

[100] F. Devernay and O. Faugeras. Straight lines have to be straight. *Machine Vision and Applications*, 2001.

[101] J. Digne, J.M. Morel, N. Audfray, and C. Lartigue. High fidelity scan merging. In *Computer Graphics Forum*, 2010.

[102] J. Digne, J.M. Morel, N. Audfray, and C. Mehdi-Souzani. The level set tree on meshes. In *Proceedings of the Fifth International Symposium on. 3D Data Processing, Visualization and Transmission, Paris, France*, 2010.

[103] J. Digne, J.M. Morel, C.M. Souzani, and C. Lartigue. Scale space meshing of raw data point sets. In *Computer Graphics Forum*, 2011.

[104] Julie Digne, Nicolas Audfray, Claire Lartigue, Charyar Mehdi-Souzani, and Jean-Michel Morel. Farman Institute 3D Point Sets - High Precision 3D Data Sets. *Image Processing on Line*, 2011.
DOI: http://dx.doi.org/10.5201/ipol.2011.dalmm_ps.

[105] Dijkstra, Edsger W. Notes on Structured Programming. 1972.

[106] Directory of open access journals.
URL: http://www.doaj.org/.

[107] Ø. Due Trier, A.K. Jain, and T. Taxt. Feature extraction methods for character recognition-a survey. *Pattern recognition*, 1996.

[108] Nebel E. and Masinter L. IETF RFC1867 — Form-based File Upload in HTML, 1995.
URL: http://tools.ietf.org/html/rfc1867.

[109] A.A. Efros and T.K. Leung. Texture synthesis by non-parametric sampling. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, 1999.

[110] Elsevier. Executable paper grand challenge.
URL: http://www.executablepapers.com/.

[111] Elsevier. Executable paper grand challenge — knowledge enhancement in the computational sciences, 2011.
URL: http://www.executablepapers.com/.

[112] Douglas Engelbart, William English, et al. A research center for augmenting human intellect. In *AFIPS Fall Joint Computer Conference*, 1968.
URL: http://sloan.stanford.edu/mousesite/1968Demo.html.

[113] England and Wales Court of Appeal. Fujitsu's Application, 1997. EWCA Civ 1174.

URL: http://www.bailii.org/ew/cases/EWCA/Civ/1997/1174.html.

[114] England and Wales Court of Appeal. Aerotel Ltd v Telco Holding Ltd and others, and Neal William Macrossan's application, 2006. EWCA Civ 1371at para. 16.
URL: http://www.bailii.org/ew/cases/EWCA/Civ/2006/1371.html#para16.

[115] Maître Eolas. Blogueurs et responsabilité reloaded, 2008.
URL: http://www.maitre-eolas.fr/post/2008/03/24/905-blogueurs-et-responsabilite-reloaded.

[116] European Patent Convention, 1973. revised in 1991 and 2000.
URL: http://www.epo.org/law-practice/legal-texts/epc.html.

[117] University of Southampton eprints. What if the publisher forbids preprint self-archiving?
URL: http://www.eprints.org/openaccess/self-faq/#publisher-forbids.

[118] European Council Directive on the Legal Protection of Computer Programs, 1991.
URL: http://ec.europa.eu/internal_market/copyright/docs/docs/1991-250_en.pdf.

[119] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*, 2010.
DOI: http://dx.doi.org/10.1007/s11263-009-0275-4.

[120] H. Farid. Blind inverse gamma correction. *Image Processing, IEEE Transactions on*, 2001.

[121] Jeanne Farrington. Seven plus or minus two. *Performance Improvement Quarterly*, 2011.
DOI: http://dx.doi.org/10.1002/piq.20099.

[122] O. Faugeras, Q.T. Luong, and S. Maybank. Camera self-calibration: Theory and experiments. In *ECCV 92*, 1992.

[123] Olivier Faugeras. *Three-Dimensional Computer Vision: A Geometric Viewpoint*. MIT Press, 1994.
ISBN: 0262061589.

[124] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, 2003.

[125] John Ferraiolo et al. Scalable Vector Graphics 1.0 Specification, 2001.
URL: http://www.w3.org/TR/SVG10/.

[126] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Ietf rfc2616 — hypertext transfer protocol – http/1.1, 1999.
URL: http://tools.ietf.org/html/rfc2616.

[127] F. Fleuret, T. Li, C. Dubout, E.K. Wampler, S. Yantis, and D. Geman. Comparing machines and humans on a visual categorization test. *Proceedings of the National Academy of Sciences*, 2011.

[128] Robert W. Floyd and Ronald L. Rivest. Algorithm 489: the algorithm select - for finding the ith smallest of n elements. *Communications of the ACM*, 1975.
DOI: http://dx.doi.org/10.1145/360680.360694.

[129] National Center for Biotechnology Information. All resources.
URL: http://www.ncbi.nlm.nih.gov/guide/all/.

[130] ISO/IEC JTC1/SC22/WG14 Working Group for the programming language C. 9899:1999 Programming languages — C, 1999. public draft archive.
URL: http://www.open-std.org/jtc1/sc22/WG14/www/docs/n1256.pdf.

[131] Free Software Foundation. What is free software?
URL: http://www.gnu.org/philosophy/free-sw.html.

[132] Open Knowledge Foundation. Open definition.
URL: http://opendefinition.org/.

[133] Code de la propriété intellectuelle, 1992.
URL: http://www.legifrance.gouv.fr/affichCode.do?cidTexte=LEGITEXT000006069414.

[134] Free Software Foundation. What is free software?
URL: http://www.gnu.org/philosophy/free-sw.html.

[135] Matteo Frigo and Steven G. Johnson. The Design and Implementation of FFTW3. *Proceedings of the IEEE 93*, 2005.
DOI: http://dx.doi.org/10.1109/JPROC.2004.840301.

[136] Jacques Froment, Lionel Moisan, Jean-Michel Morel, et al. MegaWave, 2011.
URL: http://megawave.cmla.ens-cachan.fr/.

[137] Ann Gabriel and Rebecca CApone. Executable Paper Grand Challenge Workshop. In *Proceedings of the International Conference on Computational Science*, 2011.
DOI: http://dx.doi.org/10.1016/j.procs.2011.04.060.

[138] Bruno Galerne, Yann Gousseau, and Jean-Michel Morel. Micro-Texture Synthesis by Phase Randomization. *Image Processing on Line*, 2011.
DOI: http://dx.doi.org/10.5201/ipol.2011.ggm_rpn.

[139] Matan Gavish and David Donoho. A Universal Identifier for Computational Results. *Procedia Computer Science*, 2011. Proceedings of the International Conference on Computational Science, ICCS 2011.
DOI: http://dx.doi.org/10.1016/j.procs.2011.04.067.

[140] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*.

[141] Thierry Géraud, Pierre-Yves Strub, and Jérô Darbon. Color image segmentation based on automatic morphological clustering.

[142] Pascal Getreuer. Gunturk-Altunbasak-Mersereau Alternating Projections Image Demosaicking. *Image Processing on Line*, 2011.
DOI: http://dx.doi.org/10.5201/ipol.2011.g_gapd.

[143] Pascal Getreuer. Image Interpolation with Contour Stencils. *Image Processing on Line*, 2011.
DOI: http://dx.doi.org/10.5201/ipol.2011.g_iics.

[144] Pascal Getreuer. Image Interpolation with Geometric Contour Stencils. *Image Processing on Line*, 2011.
DOI: http://dx.doi.org/10.5201/ipol.2011.g_igcs.

[145] Pascal Getreuer. Linear Methods for Image Interpolation. *Image Processing on Line*, 2011.
DOI: http://dx.doi.org/10.5201/ipol.2011.g_lmii.

[146] Pascal Getreuer. Malvar-He-Cutler Linear Image Demosaicking. *Image Processing on Line*, 2011.
DOI: http://dx.doi.org/10.5201/ipol.2011.g_mhcd.

[147] Pascal Getreuer. Roussos-Maragos Tensor-Driven Diffusion for Image Interpolation. *Image Processing on Line*, 2011.
DOI: http://dx.doi.org/10.5201/ipol.2011.g_rmdi.

[148] Pascal Getreuer. Zhang-Wu Directional LMMSE Image Demosaicking. *Image Processing on Line*, 2011.
DOI: http://dx.doi.org/10.5201/ipol.2011.g_zwld.

[149] Pascal Getreuer. Image Demosaicking with Contour Stencils. *Image Processing on Line*, 2012.
DOI: http://dx.doi.org/10.5201/ipol.2012.g-dwcs.

[150] Herman Geuvers. Proof assistants: History, ideas and future. *Sadhana*, 2009.
DOI: http://dx.doi.org/10.1007/s12046-009-0001-5.

[151] James Gillies and Robert Cailliau. *How the Web was Born: The Story of the World Wide Web*. Oxford University Press, 2000.
ISBN: 0192862073.

[152] Gomez-Diaz, Teresa. FAQ : licence & copyright pour les développements de logiciels libres de laboratoires de recherche. *Plume*, 2009.
URL: http://www.projet-plume.org/fr/ressource/faq-licence-copyright.

[153] Gomez-Diaz, Teresa. Diffuser un logiciel de laboratoire : recommandations juridiques et administratives. *Plume*, 2010.
URL: http://www.projet-plume.org/ressource/diffuser-logiciel-recomm-juridiques-admin.

[154] Gomez-Diaz, Teresa. Article vs. Logiciel : questions juridiques et de politique scientifique dans la production de logiciels. *Plume*, 2011.
URL: http://www.projet-plume.org/ressource/article-vs-logiciel.

[155] Juan Gabriel Gomila Salas and Jose-Luis Lisani. Local Color Correction. *Image Processing on Line*, 2011.
DOI: http://dx.doi.org/10.5201/ipol.2011.gl_lcc.

[156] Ulf Grenander. *General Pattern Theory: A Mathematical Study of Regular Structures*. Clarendon Press, 1993.
ISBN: 0198536712.

[157] Rafael Grompone von Gioi, Jeremie Jakubowicz, Jean-Michel Morel, and Gregory Randall. Lsd: A fast line segment detector with a false detection control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010.

[158] Rafael Grompone von Gioi, Jeremie Jakubowicz, Jean-Michel Morel, and Gregory Randall. LSD: a Line Segment Detector. *Image Processing on Line*, 2012.
DOI: http://dx.doi.org/10.5201/ipol.2012.gjmr-lsd.

[159] Rafael Grompone von Gioi, Pascal Monasse, Jean-Michel Morel, and Zhongwei Tang. Self-consistency and universality of camera lens distortion models. *CMLA Preprint, ENS-Cachan*, 2010.

[160] Rafael Grompone von Gioi, Pascal Monasse, Jean-Michel Morel, and Zhongwei Tang. Lens distortion correction with a calibration harp. In *Image Processing (ICIP), 2011 18th IEEE International Conference on*, 2011.

[161] Philip J. Guo. CDE: Run Any Linux Application On-Demand Without Installation, 2011. Proceedings of the 2011 USENIX Large Installation System Administration Conference (LISA).
URL: http://www.usenix.org/events/lisa11/tech/full_papers/Guo.pdf.

[162] Philip J. Guo and Dawson Engler. Cde: Using system call interposition to automatically create portable software packages, 2011. Proceedings of the 2011 USENIX Annual Technical Conference.
URL: http://www.stanford.edu/~pgbovine/projects/pubs/guo_usenix11_camera_ready.pdf.

[163] Katie Hafner and Matthew Lyon. *When Wizards Stay Up Late — The Origina Of The Internet*. Touchstone, 1998.
ISBN: 0684812010.

[164] Henry Edward Hardy. The history of the net. Master's thesis, Grand Valley State University, 1993.
URL: https://w2.eff.org/Net_culture/net.history.txt.

[165] S. Harnad and T. Brody. Comparing the impact of open access (oa) vs. non-oa articles in the same journals. *D-lib Magazine*, 2004.
URL: http://eprints.ecs.soton.ac.uk/10207/.

[166] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey vision conference*, 1988.

[167] R.I. Hartley. Theory and practice of projective rectification. *International Journal of Computer Vision*, 1999.

[168] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2004.
ISBN: 0521540518.

[169] D.J. Heeger and J.R. Bergen. Pyramid-based texture analysis/synthesis. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, 1995.

[170] Rolf Henkel. Web-based Image Processing, 2002.
URL: http://axon.physik.uni-bremen.de/online_calc/.

[171] Rolf Henkel. Web-based image processing — internet archive wayback machine, 2002.
URL: http://wayback.archive.org/web/*/http://axon.physik.uni-bremen.de/online_calc/.

[172] Rolf Henkel. Web-based image processing — results — internet archive wayback machine, 2002.
URL: http://wayback.archive.org/web/*/http://axon.physik.uni-bremen.de/online_calc/storage/*.

[173] Hermann Hesse. *Das Glasperlenspiel*. Suhrkamp, 2002.
ISBN: 351841335X.

[174] Konrad Hinsen. A data and code model for reproducible research and executable papers. *Procedia Computer Science*, 2011. Proceedings of the International Conference on Computational Science, ICCS 2011.
DOI: http://dx.doi.org/10.1016/j.procs.2011.04.061.

[175] Jan Hoffman. Free software, big business? *Deutsche Bank Research*, 2002.
URL: http://www.dbresearch.de/PROD/999/PROD0000000000047931.pdf.

[176] Gerard J. Holzmann. *The Early History of Data Networks*. Wiley - IEEE Computer Society Press, 1994.
ISBN: 0818667826.

[177] B.K.P. Horn and B.G. Schunck. Determining optical flow. *Artificial intelligence*, 1981.

[178] D.H. Hubel and T.N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 1962.

[179] Fraunhofer IIS-A. Early MP3 Patent Enforcement, 1998.
URL: http://www.chillingeffects.org/N/464.

[180] The Insight Journal.
URL: http://www.insight-journal.com/.

[181] Citrix Systems Inc. Xen hypervisor website, 2011.
URL: http://www.xen.org/.

[182] The MathWorks Inc. Compatibility Summary for MATLAB Software, 2011.
URL: http://www.mathworks.co.jp/help/techdoc/rn/bqsrae0.html.

[183] VMWare Inc. Vmware virtualization products website, 2011.
URL: http://www.vmware.com/virtualization/.

[184] Open Source Initiative. The open source definition.
URL: <http://opensource.org/docs/osd.

[185] Image Processing On Line (IPOL).
ISSN: 2105-1232,
DOI: http://dx.doi.org/10.5201/ipol,
URL: http://www.ipol.im/.

[186] IPOL usage report for 2010/09 - 2011/08, 2011.
URL: http://www.ipol.im/news/20110923_stats/.

[187] IPOL Software Guidelines — version 1.00.
URL: http://tools.ipol.im/wiki/ref/software_guidelines/.

[188] D.J. Jobson, Z. Rahman, and G.A. Woodell. A multiscale retinex for bridging the gap between color images and the human observation of scenes. *Image Processing, IEEE Transactions on*, 1997.

[189] Aurélie Jung. La brevetabilité des logiciels. Master's thesis, Université Robert Schuman de Strasbourg, 2006.
URL: http://www.ceipi.edu/pdf/memoires/M%C3%A9moire_Jung.pdf.

[190] J.T. Kajiya. The rendering equation. *ACM SIGGRAPH Computer Graphics*, 1986.

[191] Gaetano Kanizsa. *Vedere e pensare*. il Mulino, 1991.
ISBN: 9788815029218.

[192] Gaetano Kanizsa. *Grammatica del vedere : saggi su percezione e gestalt*. il Mulino, 1997.
ISBN: 9788815060907.

[193] Phillip Katz et al. .ZIP File Format Specification, 2007. version 6.3.2.
URL: http://www.pkware.com/documents/casestudies/APPNOTE.TXT.

[194] Brendan Kehoe. *Zen and the Art of the Internet: A Beginnerś Guide*. Prentice Hall, 1992.
ISBN: 0130107786,
URL: http://www.cs.indiana.edu/docproject/zen/zen-1.0_toc.html.

[195] Brian W. Kernighan. *D is for Digital*. DisforDigital.net, 2011.
ISBN: 1463733895.

[196] Brian W. Kernighan and Rob Pike. *The Practice of Programming*. Addison-Wesley, 1999.
ISBN: 020161586X.

[197] Brian W. Kernighan and Phillip J. Plauger. *The Elements of Programming Style*. McGraw-Hill, 2nd edition, 1978.
ISBN: 0070342075.

[198] Leonard Kleinrock, Robert Kahn, David Clark, et al. *Toward a National Research Network*. National Academy Press, 1988.
ISBN: 6610260524,
URL: http://books.nap.edu/openbook.php?isbn=NI000393.

[199] Leonard Kleinrock, Robert Kahn, David Clark, and other. *Realizing the Information Future*. National Academy Press, 1994.
ISBN: 0309050448.

[200] Donald Knuth. Literate programming. *The Computer Journal*, 1984.
DOI: http://dx.doi.org/10.1093/comjnl/27.2.97.

[201] Donald Knuth. *Literate Programming*. Stanford University Center for the Study of Language and Information, 1992.
ISBN: 0937073806.

[202] Donald E. Knuth. *The TeXbook*. Addison-Wesley, 1984.
ISBN: 0201134470.

[203] J.J. Koenderink. The structure of images. *Biological cybernetics*, 1984.

[204] J.J. Koenderink, A.J. Van Doorn, et al. Affine structure from motion. *JOSA A*, 1991.

[205] G. Koepfler, C. Lopez, and J.M. Morel. A multiscale algorithm for image segmentation by variational method. *SIAM journal on numerical analysis*, 1994.

[206] Michael Kohlhase, Joseph Corneli, Catalin David, Deyan Ginev, et al. The planetary system: Web 3.0 & active documents for stem. *Procedia Computer Science*, 2011. Proceedings of the International Conference on Computational Science, ICCS 2011.

DOI: http://dx.doi.org/10.1016/j.procs.2011.04.063.

[207] V. Kolmogorov and R. Zabih. Computing visual correspondence with occlusions using graph cuts. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, 2001.

[208] Ed Krol. *The Whole Internet User's Guide & Catalog — Academic Edition*. OŔeilly, 1996.
ISBN: 0534506747.

[209] L. Alvarez, L. Gomez and J. R. Sendra. An algebraic approach to lens distortion by line rectification. *Journal of Mathematical Imaging and Vision*, 2009.
DOI: http://dx.doi.org/10.1007/s10851-009-0153-2.

[210] Leslie Lamport. *LaTeX: A Document Preparation System : user's guide and reference manual*. Addison-Wesley, 2nd edition, 1994.
ISBN: 0201529831.

[211] Arthur Leclaire and Marc Lebrun. An implementation and detailed analysis of the K-SVD image denoising algorithm. preprint, accessed March 23, 2012.

[212] L. Lee and WEL Grimson. Gait analysis for recognition and classification. In *Automatic Face and Gesture Recognition, 2002. Proceedings. Fifth IEEE International Conference on*, 2002.

[213] Timothy B. Lee. Does not compute: court says only hard math is patentable. *Ars Technica*, 2011.
URL: http://arst.ch/qlf.

[214] Friedrich Leisch. Sweave: Dynamic generation of statistical reports using literate data analysis. In *Proceedings in Computational Statistics*, 2002.
URL: http://www.ci.tuwien.ac.at/~leisch/sweave.

[215] Friedrich Leisch. Sweave: Dynamic generation of statistical reports using literate data analysis. In *Compstat 2002 - Proceedings in Computational Statistics*, 2002.
ISBN: 3-7908-1517-9.

[216] Friedrich Leisch, Manuel Eugster, and Torsten Hothorn. Executable papers for the r community: The r2 platform for reproducible research. *Procedia Computer Science*, 2011. Proceedings of the International Conference on Computational Science, ICCS 2011.
DOI: http://dx.doi.org/10.1016/j.procs.2011.04.065.

[217] Lawrence Lessig. *The Future of Ideas*. Vintage Books, 2002.
ISBN: 0375726446.

[218] Randy Leveque, Ian Mitchell, and Victoria Stodden. Reproducible research: Tools and strategies for scientific computing, 2011.
URL: http://www.stodden.net/AMP2011/.

[219] A. Levin, P. Sand, T.S. Cho, F. Durand, and W.T. Freeman. Motion-invariant photography. In *ACM Transactions on Graphics (TOG)*, 2008.

[220] M. Li and J.M. Lavest. Some aspects of zoom lens camera calibration. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 1996.

[221] Xin Li. Reproducible research in computational science.
URL: http://www.csee.wvu.edu/~xinl/source.html.

[222] Sébastien Li-Thiao-Té. The lepton project, 2012.
URL: http://www.math.univ-paris13.fr/~lithiao/ResearchLepton/Lepton.html.

[223] Libvirt website, 2011.
URL: http://libvirt.org/.

[224] Joseph C. R. Licklider. Man-computer symbiosis. *IRE Transactions on Human Factors in Electronics*, 1960.

[225] Joseph C. R. Licklider and Robert W. Taylor. The computer as a communication device. *Science and Technology*, 1968.

[226] Nicolas Limare, Jose-Luis Lisani, Jean-Michel Morel, Ana Belén Petro, and Catalina Sbert. Simplest Color Balance. *Image Processing On Line*, 2011.
DOI: http://dx.doi.org/10.5201/ipol.2011.llmps-scb.

[227] Nicolas Limare and Jean-Michel Morel. The ipol initiative: Publishing and testing algorithms on line for reproducible research in image processing. *Procedia Computer Science*, 2011. Proceedings of the International Conference on Computational Science, ICCS 2011.
DOI: http://dx.doi.org/10.1016/j.procs.2011.04.075.

[228] Nicolas Limare, Ana Belén Petro, Catalina Sbert, and Jean-Michel Morel. Retinex Poisson Equation: a Model for Color Perception. *Image Processing on Line*, 2011.
DOI: http://dx.doi.org/10.5201/ipol.2011.lmps_rpe.

[229] Tim Lindholm and Franck Yellin. *The Java Virtual Machine Specification*. Prentice Hall, 2nd edition, 1999.
ISBN: 0201432943,
URL: http://java.sun.com/docs/books/jvms/.

[230] JL Lisani, L. Moisan, P. Monasse, and JM Morel. On the theory of planar shape. *SIAM Multiscale Modeling and Simulation*, 2003.

[231] Jose Luis Lisani, Antoni Buades, and Jean-Michel Morel. Image Color Cube Dimensional Filtering and Visualization. *Image Processing on Line*, 2011.
DOI: http://dx.doi.org/10.5201/ipol.2011.blm-cdf.

[232] C. Liu, W.T. Freeman, R. Szeliski, and S.B. Kang. Noise estimation from a single image. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, 2006.

[233] Linux kernel coding style. Chapter 2: Breaking long lines and strings.
URL: http://www.kernel.org/doc/Documentation/CodingStyle.

[234] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 2004.
DOI: http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94.

[235] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 2004.

[236] Patrick J. Lynch and Sarah Horton. *Web Style Guide*. Yale University Press, 3rd edition, 2009.
ISBN: 0300137370.

[237] J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman. Non-local sparse models for image restoration. In *Computer Vision, 2009 IEEE 12th International Conference on*, 2009.

[238] D. Marr. Vision: A computational approach, 1982.

[239] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings of the 8th International Conference on Computer Vision*, 2001.
DOI: http://dx.doi.org/10.1109/ICCV.2001.937655.

[240] C.C.D. Massachusetts. Whittemore v. Cutter, 1813. Fed. Cas. 1120.

[241] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image and Vision Computing*, 2004.

[242] Matteo Frigo. Upgrading from FFTW version 2, 2004.
URL: http://fftw.org/fftw3_doc/Upgrading-from-FFTW-version-2.html.

[243] Steve McConnell. *Code Complete*. Microsoft Press, 2nd edition, 2004.
ISBN: 0735619670.

[244] Rob McCool, John Franks, Ari Luotonen, George Phillips, and Tony Sanders. The Common Gateway Interface, 1993. archived by the Internet Archive.
URL: http://web.archive.org/web/1993/http://hoohoo.ncsa.uiuc.edu/cgi/.

[245] Z. Merali. Computational science: ...error. *Nature*, 2010.
DOI: http://dx.doi.org/10.1038/467775a.

[246] W. Metzger. Gesetze des sehens (die lehre vom sehen der formen und dinge des raumes und der bewegung). *Frankfurt/M.: Kramer*, 1975.

[247] Microsoft. Compiler options listed alphabetically.
URL: http://msdn.microsoft.com/en-us/library/fwkeyyhe.aspx.

[248] Microsoft Corporation. Windows 7 system requirements, 2011.
URL: http://windows.microsoft.com/en-US/windows7/products/system-req%uirements.

[249] George A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 1956.
URL: http://cogprints.org/730/1/miller.html.

[250] L. Moisan. Periodic plus smooth image decomposition. *Journal of Mathematical Imaging and Vision*, 2011.

[251] L. Moisan and B. Stival. A probabilistic criterion to detect rigid point matches between two images and estimate the fundamental matrix. *International Journal of Computer Vision*, 2004.

[252] P. Monasse and F. Guichard. Fast computation of a contrast-invariant image representation. *Image Processing, IEEE Transactions on*, 2000.

[253] Pascal Monasse. Quasi-Euclidean Epipolar Rectification. *Image Processing on Line*, 2011.
DOI: http://dx.doi.org/10.5201/ipol.2011.m_qer.

[254] Marco Mondelli and Adina Ciomaga. Finite Difference Schemes for MCM and AMSS. *Image Processing on Line*, 2011.
DOI: http://dx.doi.org/10.5201/ipol.2011.cm_fds.

[255] J.M. Morel and G. Yu. Is sift scale invariant? *Inverse Problems and Imaging*, 2011.

[256] Pierre Mounier. Diffamation dans les forums de discussion : quelle responsabilité pour les webmestres ?, 2002.
URL: http://homo-numericus.net/spip.php?article169.

[257] Mathematical Programming Computation.
URL: http://mpc.zib.de/.

[258] D. Mumford and A.Ã. Desolneux. Pattern theory: The stochastic analysis of real-world signals (applying mathematics). 2010.

[259] D. Mumford and J. Shah. Optimal approximations by piecewise smooth functions and associated variational problems. *Communications on pure and applied mathematics*, 1989.

[260] K.P. Murphy, Y. Weiss, and M.I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, 1999.

[261] Nicholas Negroponte. *Being Digital*. Vintage, 1996.
ISBN: 0679762906.

[262] R. Ng, M. Levoy, M. Brédif, G. Duval, M. Horowitz, and P. Hanrahan. Light field photography with a hand-held plenoptic camera. *Computer Science Technical Report CSTR*, 2005.

[263] M. Nguyen, G. Gimel'farb, and P. Delmas. Web-based on-line computational stereo vision. In *Proceedings of the 23rd International Conference on Image and Vision Computing*, 2008.
DOI: http://dx.doi.org/10.1109/IVCNZ.2008.4762147.

[264] N. Nguyen, P. Milanfar, and G. Golub. A computationally efficient superresolution image reconstruction algorithm. *Image Processing, IEEE Transactions on*, 2001.

[265] Jakob Nielsen. *Usability Engineering*. Morgan Kaufmann, 1993.
ISBN: 0125184069.

[266] Piotr Nowakowski, Eryk Ciepiela, Daniel Harezlak, Joanna Kocot, et al. The collage authoring environment. *Procedia Computer Science*, 2011. Proceedings of the International Conference on Computational Science, ICCS 2011.
DOI: http://dx.doi.org/10.1016/j.procs.2011.04.064.

[267] NVIDIA Corporation. NVIDIA CUDA Compute Unified Device Architecture Programming Guide, version 1.1, 2007.
URL: http://developer.download.nvidia.com/compute/cuda/1_1/NVIDIA_CU%DA_Programming_Guide_1.1.pdf.

[268] United States Court of Appeals. Apple Computer, Inc. v. Franklin Computer Corp., 1983. 714 F. 2d 1240.

[269] United States Court of Appeals. In re Kuriappan P. Alappat, Edward E. Averill and James G. Larsen, 1994. 33 F.3d 1526.

[270] United States Court of Appeals. Bernstein v. United States Dept. of Justice, 1997. 176 F.3d 1132.

[271] United States Court of Appeals. State Street Bank & Trust Co., Plaintiff-Appellee, v. Signature Financial Group, Inc., Defendant-Appellant, 1998. 149 F.3d 1368.

[272] United States Court of Appeals. In re Bernard L. Bilski and Rand A. Warsaw, 2008. 545 F.3d 943.

[273] United States Court of Appeals. Cybersource Corp. v. Retail Decisions Inc., 2011. 620 F. Supp. 2d 1068.

[274] University of Chicago Press Staff. *The Chicago Manual of Style*. University of Chicago Press, 16th edition, 2010.
ISBN: 0226104206.

[275] Supreme Court of the United States. Le Roy v. Tatham, 1852. 55 U.S. 156.

[276] Supreme Court of the United States. Gottschalk, Acting Commissioner of Patents v. Benson, et al., 1972. 409 U.S. 63.

[277] Supreme Court of the United States. Parker, Acting Commissioner of Patents and Trademarks v. Flook, 1978. 437 U.S. 584.

[278] Supreme Court of the United States. Diamond, Commissioner of Patents and Trademarks v. Diehr, et al., 1981. 450 U.S. 175.

[279] Supreme Court of the United States. Bernard L. Bilski and Rand A. Warsaw v. David J. Kappos, Under Secretary of Commerce for Intellectual Property and Director, Patent and Trademark Office, 2010. 130 S. Ct. 3218.

[280] ANSI X3J11 Technical Committee on the C Programming Language. ANSI X3.159-1989 Programming Language C, 1989. unofficial archive.
URL: http://flash-gordon.me.uk/ansi.c.txt.

[281] Openvz website, 2011.
URL: http://www.openvz.org/.

[282] Open Research Computation.
ISSN: 2042-5767,
URL: http://www.openresearchcomputation.com/.

[283] P. Van Gorp and P. Grefen. Supporting the internet-based evaluation of research software with cloud infrastructure. *Software and Systems Modeling*, 2010.
DOI: http://dx.doi.org/10.1007/s10270-010-0163-y.

[284] S.C. Park, M.K. Park, and M.G. Kang. Super-resolution image reconstruction: a technical overview. *Signal Processing Magazine, IEEE*, 2003.

[285] United States Patent and Trademark Office. Interim Examination Instructions for Evaluating Subject Matter Eligibility Under 35 U.S.C. § 101, 2009.
URL: http://www.uspto.gov/web/offices/pac/dapp/opla/2009-08-25_interim_101_instructions.pdf.

[286] P. Pérez, M. Gangnet, and A. Blake. Poisson image editing. In *ACM Transactions on Graphics (TOG)*, 2003.

[287] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 1990.

[288] S.M. Pizer, E.P. Amburn, J.D. Austin, R. Cromartie, A. Geselowitz, T. Greer, B. ter Haar Romeny, J.B. Zimmerman, and K. Zuiderveld. Adaptive histogram equalization and its variations. *Computer vision, graphics, and image processing*, 1987.

[289] J. Portilla, V. Strela, M.J. Wainwright, and E.P. Simoncelli. Image denoising using scale mixtures of gaussians in the wavelet domain. *Image Processing, IEEE Transactions on*, 2003.

[290] Jef Poskanzer and Bryan et al. Henderson. Netpbm — The Netpbm Formats, 2009.
URL: http://netpbm.sourceforge.net/doc/#formats.

[291] Debian Project. The debian free software guidelines.
URL: http://www.debian.org/social_contract#guidelines.

[292] James J. Quirk. Amrita ebook, 2011.
URL: http://www.amrita-ebook.org/.

[293] J. Rabin, J. Delon, and Y. Gousseau. A contrario matching of sift-like descriptors. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, 2008.

[294] R. Raskar, A. Agrawal, and J. Tumblin. Coded exposure photography: motion deblurring using fluttered shutter. *ACM Transactions on Graphics*, 2006.

[295] Red Hat Inc. Red Hat Enterprise Linux Technology capabilities and limits, 2011.
URL: http://www.redhat.com/rhel/compare/.

[296] J. Rissanen. A universal prior for integers and estimation by minimum description length. *The Annals of statistics*, 1983.

[297] D. Robinson and K. Coar. Ietf rfc 3875 — the common gateway interface (cgi) version 1.1, 2004.
URL: http://tools.ietf.org/html/rfc3875.

[298] O. Rodeh and A. Teperman. zfs - a scalable distributed file system using object disks. In *Proceedings. 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies*, 2003.
DOI: http://dx.doi.org/10.1109/MASS.2003.1194858.

[299] Rodin, Josip and Aoki, Osamu and. Debian New Maintainers' Guide, 2011.
URL: http://www.debian.org/doc/manuals/maint-guide/.

[300] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 1958.

[301] Retinex Poisson Equation: a Model for Color Perception, 2011. IPOL demo.
URL: http://www.ipol.im/pub/demo/lmps_retinex_poisson_equation/.

[302] Simplest Color Balance, 2011. IPOL demo.
URL: http://www.ipol.im/pub/demo/lmps_simplest_color_balance/.

[303] L.I. Rudin, J.L. Lisani, and J.M. Morel. Registration and comparison of three dimensional objects, 2010. WO Patent WO/2010/093,824.

[304] L.I. Rudin and S. Osher. Total variation based image restoration with free local constraints. In *Image Processing, 1994. Proceedings. ICIP-94., IEEE International Conference*, 1994.

[305] L.I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 1992.

[306] N. Sabater, A. Almansa, and J.M. Morel. Rejecting wrong matches in stereovision. *preprint*, 2008.

[307] N. Sabater, J.M. Morel, A. Almansa, et al. How accurate can block matches be in stereo vision? *SIAM Journal on Imaging Sciences*, 2011.

[308] Reed Saltzer and Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 1984.
DOI: http://dx.doi.org/10.1145/357401.357402.

[309] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, 2002.

[310] Robert C. Seacord. *The CERT C Secure Coding Standard*. Addison-Wesley, 2008.
ISBN: 0321563212,
URL: https://www.securecoding.cert.org/confluence/display/seccode/CERT+C+Secure+Coding+Standard.

[311] Stephen Segaller. *Nerds 2.0.1: A Brief History of the Internet*. TV Books, 1999.
ISBN: 1575000881.

[312] Peter Seibel. *Coders at Work: Reflections on the Craft of Programming*. Apress, 2009.
ISBN: 1430219483,
URL: http://www.codersatwork.com/.

[313] S.M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, 2006.

[314] Jean Serra. *Image Analysis and Mathematical Morphology*. Academic Press, 1982.
ISBN: 0126372403.

[315] James A. Sethian. *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge University Press, 1999.
ISBN: 0521645573.

[316] C.E. Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 2001.

[317] E. Shechtman and M. Irani. Matching local self-similarities across images and videos. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, 2007.

[318] J. Siciarek and B. Wiszniewski. Ioda - an interactive open document architecture. *Procedia Computer Science*, 2011. Proceedings of the International Conference on Computational Science, ICCS 2011.
DOI: http://dx.doi.org/10.1016/j.procs.2011.04.070.

[319] SIAM Journal on Imaging Sciences (SIIMS).
ISSN: 1936-4954,
URL: http://www.siam.org/journals/siims.php.

[320] Sparc europe.
URL: http://www.sparceurope.org/resources/hot-topics/open-journals.

[321] Richard M. Stallman. *Free Software, Free Society*. Createspace, 2009.
ISBN: 1441436855.

[322] Tom Standage. *The Victorian Internet: The Remarkable Story of the Telegraph and the Nineteenth Century's On-line Pioneers*. Walker & Company, 2007.
ISBN: 0802716040.

[323] J.L. Starck and F. Murtagh. Automatic noise estimation from the multiresolution support. *Publications of the Astronomical Society of the Pacific*, 1998.

[324] Guy L. Steele and Eric S. Raymond. *The New Hacker's Dictionary*. 3rd edition, 1996.
ISBN: 0262680920,
URL: http://catb.org/jargon/html/I/indent-style.html.

[325] Neal Stephenson. *In the Beginning...was the Command Line*. William Morrow Paperbacks, 1999.
ISBN: 0380815931,
URL: http://www.cryptonomicon.com/beginning.html.

[326] Victoria Stodden. Enabling reproducible research: Open licensing for scientific innovation. *International Journal of Communications Law and Policy*, 2009.
URL: http://www.ijclp.net/files/ijclp_web-doc_1-13-2009.pdf.

[327] Victoria Stodden. The legal framework for reproducible research in the sciences: Licensing and copyright. *IEEE Computing in Science and Engineering*, 2009.
DOI: http://dx.doi.org/10.1109/MCSE.2009.19.

[328] Victoria Stodden. The scientific method in practice: Reproducibility in the computational sciences, 2010. MIT Sloan Research Paper No. 4773-10.
DOI: http://dx.doi.org/10.2139/ssrn.1550193.

[329] Victoria Stodden. The credibility crisis in computational science: An information issue, 2012. Dean's Lecture, UC Berkeley School of Information, Berkeley, CA.
URL: http://www.stanford.edu/~vcs/talks/BerkeleyFeb2012-STODDEN.pdf.

[330] William Jr. Strunk and E. B. White. *The Elements of Style*. Longman, 4th edition, 1999.
ISBN: 0205313426.

[331] Jeremy Sugarman, Ganesh Venkitachalam, and Beng-Hong Lim. Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. In *Proceedings*

of the 2001 USENIX Annual Technical Conference, 2001.
URL: http://www.usenix.org./publications/library/proceedings/usenix01/sugerman/sugerman.pdf.

[332] Sun Microsystems. Code Conventions for the Java Programming Language. Section 4.1: Line Length.
URL: http://www.oracle.com/technetwork/java/codeconventions-136091.html#313.

[333] Adobe Systems. Encapsulated PostScript 3.0 Specification, 1992.
URL: http://partners.adobe.com/public/developer/en/ps/5002.EPSF_Spec.pdf.

[334] Adobe Systems. TIFF 6.0 Specification, 1992.
URL: http://partners.adobe.com/public/developer/tiff/.

[335] Mark Taylor. Lame technical faq, 2000. [online; accessed 19-October-2011].
URL: http://lame.sourceforge.net/tech-FAQ.txt.

[336] Y. Tendero, J. Gilles, S. Landeau, and JM Morel. Efficient single image non-uniformity correction algorithm. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, 2010.

[337] the Debian Installer team. Meeting Minimum Hardware Requirements, 2010.
URL: http://www.debian.org/releases/stable/i386/ch03s04.html.en.

[338] The Debian Project. The debian free software guidelines.
URL: http://www.debian.org/social_contract#guidelines.

[339] The Debian Project. Debian Policy Manual, 2011.
URL: http://www.debian.org/doc/debian-policy/.

[340] The IEEE and The Open Group. The Open Group Base Specifications Issue 7, IEEE Std 1003.1-2008, 2008. POSIX.1-2008, section utilities/pax/ustar.
URL: http://pubs.opengroup.org/onlinepubs/9699919799/utilities/pax.html#tag_20_92_13_06.

[341] The IEEE and The Open Group. The Open Group Base Specifications Issue 7, IEEE Std 1003.1-2008, 2008. POSIX.1-2008, section utilities/c99.
URL: http://pubs.opengroup.org/onlinepubs/9699919799/utilities/c99.html.

[342] the `comp.compression` Usenet group users. Where can i find lenna and other images?
URL: http://www.faqs.org/faqs/compression-faq/part1/section-30.html.

[343] TOP500.Org. K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect, 2011.
URL: http://www.top500.org/system/10810.

[344] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon. Bundle adjustment-a modern synthesis. *Vision algorithms: theory and practice*, 2000.

[345] Kate L. Turabian. *A Manual for Writers of Research Papers, Theses and Dissertations*. University of Chicago Press, 7th edition, 2007.
ISBN: 0226823377.

[346] M. Unser, A. Aldroubi, and M. Eden. Fast b-spline transforms for continuous image representation and interpolation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1991.

[347] United States Code - Title 17. Chapter 1 - Subject Matter And Scope of Copyright.
URL: http://www.law.cornell.edu/uscode/17/usc_sup_01_17_10_1.html.

[348] United States Code - Title 35. Chapter 2 - Patentability of Inventions and Grant of Patents.
URL: http://www.law.cornell.edu/uscode/35/usc_sup_01_35_10_II.html.

[349] Pieter Van Gorp and Paul Grefen. Supporting the internet-based evaluation of research software with cloud infrastructure. *Software and Systems Modeling*, 2009.
DOI: http://dx.doi.org/10.1007/s10270-010-0163-y.

[350] Pieter van Gorp and Steffen Mazanek. SHARE A Web Portal for Creating and Sharing Executable Research Papers. 2011.
DOI: http://dx.doi.org/10.1016/j.procs.2011.04.062.

[351] van Rossum, Guido. Python 3000, 2006.
URL: http://www.python.org/dev/peps/pep-3000/.

[352] van Rossum, Guido and Warsaw, Barry. Style Guide for Python Code. Section: Maximum Line Length.
URL: http://www.python.org/dev/peps/pep-0008/.

[353] Patrick Vandewalle, Jelena Kovacevic, and Martin Vetterli. Reproducible research in signal processing — what, why and how? *IEEE Signal Processing Magazine*, 2009.
DOI: http://dx.doi.org/10.1109/MSP.2009.932122.

[354] Patrick Vandewalle, Sabine Süsstrunk, and Martin Vetterli. A frequency domain approach to registration of aliased images with application to super-resolution. *EURASIP Journal of Applied Signal Processing*, 2006.
DOI: http://dx.doi.org/10.1155/ASP/2006/71459.

[355] L. Vincent. Fast grayscale granulometry algorithms. In *EURASIP Workshop ISMM*, 1994.

[356] L. Vincent. Morphological area openings and closings for grey-scale images. *NATO ASI Series F Computer and Systems Sciences*, 1994.

[357] L. Vincent and P. Soille. Watersheds in digital spaces: an efficient algorithm based on immersion simulations. *IEEE transactions on pattern analysis and machine intelligence*, 1991.

[358] P. Viola and M.J. Jones. Robust real-time face detection. *International journal of computer vision*, 2004.

[359] Virtanen, Perttu. Latest Software Patent Law Developments in the US and EU. *script-ed*, 2010.
DOI: http://dx.doi.org/10.2966/scrip.070310.562,
URL: http://www.law.ed.ac.uk/ahrc/script-ed/vol7-3/virtanen.asp.

[360] W3C HTML Working Group. Xhtml 1.0 the extensible hypertext markup language (second edition).
URL: http://www.w3.org/TR/xhtml1/.

[361] Bertrand Warusfel. La brevetabilité des inventions logicielles dans les jurisprudences européenne et américaine. In *Actes du colloque de l'AFDIT*, 2002.
URL: http://www.droit.univ-paris5.fr/warusfel/articles/JurInvLog_warusfel03.pdf.

[362] Jon Watson. Virtualbox: bits and bytes masquerading as machines. *Linux Journal*, 2008.
URL: http://www.linuxjournal.com/article/9941.

[363] Web Hypertext Application Technology Working Group (WHATWG). HTML Living Standard, 2012.
URL: http://www.whatwg.org/specs/web-apps/current-work/multipage/.

[364] L.Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 2000.

[365] Weinberger, Benjy and Silverstein, Craig and Eitzmann, Gregory and Mentovai, Mark and Landray, Tashana. Google C++ Style Guide. Section: Line Length.
URL: http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml#Line_Length.

[366] M. Wertheimer. Untersuchungen zur Lehre von der Gestalt. II. *Psychological Research*, 1923.

[367] David A. Wheeler. Secure programming for linux and unix howto, 2003.
URL: http://www.dwheeler.com/secure-programs/.

[368] David A. Wheeler. Free-libre / open source software (floss) is commercial software, 2006.
URL: http://www.dwheeler.com/essays/commercial-floss.html.

[369] David A. Wheeler. Why open source software / free software (oss/fs, floss, or foss)? look at the numbers!, 2007.
URL: http://www.dwheeler.com/oss_fs_why.html.

[370] Wikimedia. Wikimedia Traffic Analysis Report - Operating Systems - September 2011, 2011.
URL: http://stats.wikimedia.org/archive/squid_reports/2011-09/SquidR%eportOperatingSystems.htm.

[371] Wikipedia. Usage share of operating systems — wikipedia, the free encyclopedia, 2011. [Online; accessed 31-October-2011].
URL: http://en.wikipedia.org/w/index.php?title=Usage_share_of_operating_systems&oldid=458256172.

[372] Wikipedia. Virtual disk image - wikipedia, the free encyclopedia, 2011. [Online; accessed 20-October-2011].
URL: http://en.wikipedia.org/w/index.php?title=Virtual_disk_image&oldid=447383468.

[373] World Wide Web Consortium (W3C). Html5, 2012.
URL: http://www.w3.org/TR/html5/.

[374] Akira Yanagawa, Alexander C. Loui, Jiebo Luo, Shih-Fu Chang, Dan Ellis, Wan Jiang, Lyndon Kennedy, and Keansub Lee. Kodak consumer video benckmark data set: concept definition and annotation. Technical Report 246-2008-4, Columbia

University ADVENT, 2008.
URL: http://www.ee.columbia.edu/ln/dvmm/consumervideo/.

[375] L.P. Yaroslavsky. Digital picture processing. an introduction. *Springer Series in Information Sciences*, 1985.

[376] G. Yu, G. Sapiro, and S. Mallat. Solving inverse problems with piecewise linear estimators: from gaussian mixture models to structured sparsity. *Image Processing, IEEE Transactions on*, 2010.

[377] Guoshen Yu and Jean-Michel Morel. ASIFT: An Algorithm for Fully Affine Invariant Comparison. *Image Processing On Line*, 2011.
DOI: http://dx.doi.org/10.5201/ipol.2011.my-asift.

[378] Guoshen Yu and Guillermo Sapiro. DCT image denoising: a simple and effective image denoising algorithm. *Image Processing On Line*, 2011.
DOI: http://dx.doi.org/10.5201/ipol.2011.ys-dct.

[379] C. Zach, T. Pock, and H. Bischof. A duality based approach for realtime tv-l 1 optical flow. *Pattern Recognition*, 2007.

[380] Robert Hobbes Zakon. Hobbes' internet timeline, 2011.
URL: http://www.zakon.org/robert/internet/timeline/.

[381] Z. Zhang. A flexible new technique for camera calibration. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 2000.

[382] Shaobin Zhu Zhu. Patent Rights Under FOSS Licensing Schemes. *Shidler Journal of Law, Commerce & Technology*, 2007.
URL: http://www.lctjournal.washington.edu/Vol4/a04zhu.html.

[383] Rusen Öktem, Leonid Yaroslavsky, Karen Egiazarian, and Jaakko Astola. Transform based denoising algorithms: Comparative study, 1999.
URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.55.5616.

TODO: split by categories (books, law, standards, journals, IPOL articles, other articles, www, misc.)