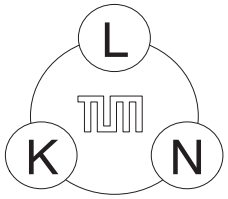


Technische Universität München
Lehrstuhl für Kommunikationsnetze
Fachgebiet Medientechnik
Prof. Dr.-Ing. Eckehard Steinbach

Master's Thesis

Compression of images and videos by geometrization

Author:	STOIBER, Nicolas
Matriculation Number:	2835082
Address:	1 avenue Aristide Briand F33110 Le Bouscat France
Email Address:	nicolas.stoiber@mytum.de
Supervisors:	Prof. Dr.-Ing. Eckehard Steinbach (TUM) Bruno Lévy (INRIA)
Begin:	05. March 2007
End:	05. September 2007

Abstract

For technical and historical reasons, the raster graphics representation of visual information, like an image or a video, has become predominant over the years. For compression purposes in particular, the superiority of JPEG and MPEG schemes has strengthened the dominant position of pixel-based representations of images and videos. The drawback of these description techniques is that the pixels that carry the visual information have no extrinsic cohesion with one another. This complicates the execution of higher-level processing related to the semantics of the information. In this Diplomarbeit, an alternative, higher-order image and video description technique is considered. Image and videos will be represented by respectively two or three dimensional meshes, in which the color information is modelized by polynomial functions. This representation can be obtained from a bitmap source through a fully automatic conversion process. The visual characteristics of the new format as well as its compression performances are investigated and compared with the traditional raster graphics techniques.

Contents

Chapter 1

Introduction

Background

An image says more than a thousand words. For this exact reason, technologies have been developed to create, capture, analyze and process images about as far as we can recall. The digital revolution of the late 20^{th} century has stimulated this research field and its technological applications, and it becomes rare to meet somebody who has never seen an image on a computer. Moreover with the development of digital portable devices like PDAs, cellular phones, game and music devices, digital pictures have extended their presence and can now be captured and visualized on a growing variety of platforms, each of them having its own requirements in terms of screen resolution, computational power and storage. The relations between the different actors of the image manipulation landscape become more complex as their number and the differences between them grow. A challenge nowadays is certainly the versatility of image and video formats, and their ability to adapt to different viewing conditions as well as computational and bandwidth limitations.

Historically, digital images and videos are represented and stored as "bitmaps", also called "raster graphics". Bitmaps are regular grids of rectangular elements called pixels[1]. Each of these elements carries a single color information (usually in the form of a RGB vector), and form a digital picture when a sufficient number of them are put side by side. Digital videos are nothing but a sequence of digital pictures displayed one after another at an appropriate pace (usually 24 to 30 images per second). This representation of images and videos is straightforward, because it is the representation delivered by digital picture recording devices like CCD and CMOS cameras. These sensors capture light as grid of small independent photosensitive elements that correspond to the later pixels in the digital representation of the image. Additionally to the histori-

[1]Pixel is the abbreviation of "picture element"

cal arguments, the pixel-based representation of visual information presents interesting advantages as far as analyzing and processing are concerned ; namely the regular and simple structure of a pixel grid, and the possibility to theoretically study digital pictures by relying on the quantization and sampling theories.

The topic of image and video formats is also strongly related to the concept of information compression. Except for special processing tasks, it does not make sense to store or transmit visual information in its raw form now that powerful compression tools can be easily used, and that computational power has become cheaper than transmission bandwidth. For the last 20 to 30 years, the study of the compression of images and videos has been a very active research and developement field. The incredible progress of hardware capabilities along with the decrease of memory price have made both the development of the research effort and of industrial applications possible. The still image coding scheme "JPEG" (Joint Photographic Expert Group, 1994) and the video coding scheme MPEG (Motion Picture Expert Group, 1996), relying on the pixel-based description of an image, represent major breakthrough in the way visual information is handled. Processing the frequency spectrum of the visual signal, and using time redundancy and motion compensation (MPEG) lead to unprecedented results in terms and compression performances. Since then, the performances and features of this paradigm has kept on being improved, either by an increase of complexity and computational capabilities (MPEG4:AVC/H.264, 2003) or by the introduction of different spectral spaces (JPEG2000 and its Wavelet transformation, 2000). Because of this ability to be efficiently compressed, the pixel-based representation of visual information has become the uncontested dominant form of picture description and JPEG and MPEG have set the tone in terms of data compression efficiency. As a result, the compactness of an image or a video format is nowadays one of its most important characteristics, and definitely one that needs to be considered when planning to work on an alternative image representation.

Motivation

Despite the indisputable qualities of raster graphics, one could argue that there may be cases and applications where alternative representations of visual information are more appropriate. The growing success of vector graphics for special applications like web browsing is a good example. In vector graphics the numerous pixel coefficients are replaced by a smaller group of higher-order equations describing the content of the image. This can be seen as an abstraction of an image's representation: instead of manipulating small fragments that hold no intrinsic coherence, we work with analytic equations that contain higher level information.

This dualism in image and video representations can be compared to an issue that has been deeply investigated in the research team that hosted this Diplomarbeit: the rep-

resentation of geometry for computer graphics applications. Let us point out the similarities of these two subjects. The geometry capture devices, mainly 3D scanners, often deliver the captured geometry as a triangulated surface in a 3D space. This data representation consists of a fine sampling of the original continuous geometry which is approximated by piecewise linear variations (each triangle is a portion of a 2D plane). Like for images and its pixel-based representation, the more elements[2] we use, the better the approximation become. Due to this capture format, and because it has other interesting advantages, computer graphics processing techniques based on triangulated surfaces have been widely developed and used. Nevertheless, it is far from being the most appropriate representation for several tasks, like manual geometry editing, or the application of texture onto geometry. For the latter, it is much more interesting to have a description of the geometry consisting of wider and regular patches of bicubic equations (N.U.R.B.S. for example[3]). The geometry is not described by a great amount of small and simple elements anymore, but by larger, higher-order shapes. Like for images and videos, this can be seen as an abstraction of the geometrical information.

In the computer graphics field however, the "abstracted" representation is a usual working basis. Many computer graphics algorithms and software rely on this structure. As a result, between the triangle-based description delivered by a 3D scanner and the higher-order description desired at the application side, a *conversion* is required. Due to the extensive use of both types of geometry representation, the conversion issue has been the subject of an important research effort. The research is still open, but a collection of efficient solutions have been proposed, and are now widely adopted. In comparison, the popularity of vector graphics and alternative image description schemes in image processing applications seems pretty low. The ambition of this Diplomarbeit is to investigate the possibilities concerning the same kind of representation *conversion* for images and videos. In the image processing community where the raster representation is dominant, we want to explore an alternative way of describing the visual information of a picture, based on higher level primitives. Several interesting features can be expected from this change in image representation:

- Equation-based pictures are easily scalable unlike pixel-based pictures. Unlike pixel-based formats they store a resolution-independent information.
- The description of the higher-order information (equations), can be more compact than an exhaustive pixel grid description. In that case a conversion from pixels to equations would perform a *compression* of the image information.
- Equations hold a more global description of the image part they describe. They can be more easily related to the semantics of what has being displayed than the coefficients of a pixel grid. This may facilitate tasks like image segmentation or object recognition.

[2]Pixels in the case of images, and triangles in the case of geometry
[3]Non-Uniform Rational B-Splines

- Content-altering, stylization and other non-photorealistic rendering processes would certainly benefit from a higher level image description.

The goal of this Diplomarbeit is not to fully develop all the features listed above. We do not aim at proposing a new universal image representation that would perfectly meet their needs and replace the pixel-based image description. This will probably not happen anytime soon. Our purpose in this work, is to explore the capabilities of an alternative image and video representation: study how visual information description based on higher order primitives can be constructed from the traditional pixel-based description, and point out its advantages and its drawbacks. What we expect to achieve is to develop our experience on problems that are bound to arise when such a change of representation is performed, and to evaluate the gain or loss of information and possibilities of further processing with the new format. As previously exposed, the data size is an important aspect and conditions the viability of a format. Thefore the main focus of this *prospective* work has been the study of the compression performances of the considered image description method.

Please note that it was from the beginning considered a prospective research work. Indeed, the research team that supervised this work did not consider it the contribution to the solving of a well-known problem, but as an exploration of a new prospect direction that could open doors to further interesting image processing tasks.

Structure of this report

The report begins with the description of the context of the Diplomarbeit in part 2. Practical informations like the work environment and the available tools are presented, as well as the scientific context with references to previous works belonging to the scope of the report. Then in part 3, the core of our image and video representation is exposed. Part 3 is the backbone of this report since it contains the principles of the representation as well as the method used to construct it. All these informations have been kept in the same part for coherence and comprehension purposes, however a large variety of subjects are covered in it. Thus in part 3 the methods are only described qualitatively. In parts 4 and 5, the technical details of respectively the conversion and the compression schemes touched on in part 3 can be found. The results are exposed in part 6. They concern both the quality of the representation of images and videos and the compression performances. Finally general comments, conclusions and outlooks to further investigations are presented in part 7.

Chapter 2

Context of this work

2.1 Work environment

This Diplomarbeit takes places at the public research institute INRIA Lorraine[1] in Nancy, France. INRIA stands for "national research institute for information technologies and its applications". INRIA Lorraine is one of the five research centers forming the INRIA institute (the others are located in Lille, Rennes, Grenoble, Rocquencourt and Sofia-antipolis). The research effort of the institute is segmented in so called projects, each project developing a particular research theme related to information technologies. This Diplomarbeit was affiliated to the ALICE project. The members of the ALICE research team are specialists of geometry and light processing belonging to the computer graphics community. Their main research themes are related to geometry processing. Among them we find the automatic design of vector fields onto surfaces, the segmentation and parametrization of three-dimensional meshes and the approximation of geometrical shape. The team also devoted research and programming efforts to light processing algorithms with the development of global illumination methods based on the concepts of radiosity and photon mapping.

Despite being mainly focused on computer graphics topics, the work of the ALICE team involves concepts that are sometimes worth extending to other applications. Thus, a reflexion on mobile function basis and higher order representations, first initiated for the processing of geometry, was found to be also interesting for images. The recent work of ALICE members was oriented on the parametrization of triangulated meshes. This special task needs an efficient method to segment the geometry, and that is what Lévy tried to improve when exploring these subjects. Therefore, when the idea came out to translate these thoughts to the image processing field, the intent was to use them for image segmentation purposes. A region-growing segmentation method called

[1] also known as "Loria"

ARDECO[2][LL06] was then developed and achieved interesting results. It was designed to work at the pixel-level, but for performances reasons, an intermediate triangulated structure was introduced. The source image was first to be segmented into a set of triangular cells, and the segmentation process was then executed using the triangles as elementary entities. The final detected regions are groups of triangular cells. The concept of image triangulation originally introduced as an intermediate structure to accelerate ARDECO showed interesting visual characteristics, and the idea to use it as an alternative representation of visual information came out: images and even videos may be well described by geometrical objects and higher order colorimetric primitives[3]. To explore this possibility and state whether such an approach is worth developing important research effort, it was decided to dedicate this Diplomarbeit to the question.

2.2 Previous works

Trying to describe images with higher-order mathematical objects is not new. Although it is not the most popular format for visual information, vector graphics has made his way through specific applications where their particular graphics style and their compactness are appreciated. Internet users are familiar with the look-and-feel of websites designed with Flash, and vector graphics images are well-suited for the creation and storage of abstracted, computer-generated visual objects like logos or writing fonts. Yet, since these formats rely on mathematical formulas, they are not really appropriate to represent real visual signals, which are made of irregularities and discontinuities. Many methods that have introduced the use of higher-order components to depict images and video are thus classified in the category of image abstraction and non-photorealistic rendering (NPR). A few of them are presented in the next section. Methods using higher order image and video descriptions in compression scenarios are introduced in 2.2.2.

2.2.1 Image abstraction and stylizing

The purpose of the following methods is to obtain a stylized or abstracted image from a natural or synthetic raster graphics source. Obviously, the perception theory is the key aspect in judging the result. An abstract image should enhance important features, and make the structure of the image clear to the viewer. The different schemes differ by the type of primitives they use, and the possibilities they offer in terms of accuracy and stylistic rendering. The popularity of image abstraction began with Haerberli's *Paint by numbers: abstract image representations* [Hae90]. He proposed the decomposition of images into a set of simple primitives filled with constant colors, to create a impressionist

[2]Automatic Region DEtection and COnversion.

[3]the description of the representation is exposed in part 3

paintings out of raster images. Nevertheless to really be efficient, a method needs to be able to understand the structure and detect meaningful shapes and shadings from the original images pictures. The term "meaningful" is, of course, closely related to the human perception of visual information. In [DS02], DeCarlo *et al.* have presented a method to reveal this structure of regions and boundaries using computer vision concepts. The important elements of a picture are located using a model of human perception, and are preserved and highlighted in the final stylized image. This way, *unsupervised* segmentation of an image into regions can be performed. Like in [Hae90], the detected regions are filled with constant colors only, which delivers once again a very stylized result. Similar schemes have also been developed for videos: in [WXSC04], Wang *et al.* look to turn videos into cartoon animations. In this *video tooning* paper, videos are treated as space-time volumes of image data[4]. The visual objects in the video sequence are semi-automatically segmented. The user needs to outline the objects on selected key frames and a mean shift algorithm interpolates the segmentation throughout the whole sequence. The semantic regions can then be rendered according to the user stylization desire. Other video abstraction methods have preceded this one, like [Lit97] and [Her01] , which work with the calculation of an optical flow and energy minimization at the pixel level. These methods are less likely to produce a clean semantic segmentation of the video sequence, and aim at giving an artistic painting style to the video.

The previously presented works were mainly focused on abstract representations of visual information. The resemblance between the original picture and the stylized one is an issue, but the goal is not to faithfully represent the original, rather to produce an artistic impression out of it. Trying to render realistic images with these methods typically output over-segmented images consisting of irregular regions with constant colors. However, other research efforts have been devoted to more faithful geometrical representations of image as we will see in the following.

More realistic impressions are typically obtained by introducing higher-order colorimetric primitives to describe the shading of a region. The regions themselves have a simpler form to allow the use of more powerful algorithms. In [SP06] for example, the segmentation relies on a constraint Delaunay triangulation on which a linear spline set is fit to approximate the image, yielding a polygon based-vectorization. The ARDECO method [LL06], which has been mentioned in 2.1 also belongs to this category. Indeed it detects a region set delimited by cubic splines, in which the colors are modeled by constant, linear or circular gradients. Even more recently, [SLWS07] introduced a visually very convincing semi-automatic vectorization method based on optimized gradient meshes. The optimized gradient meshes consists of rectangular patches whose boundaries are cubic Bezier splines. Colors coefficients are carried by the vertices, and the region shadings are interpolated from them.

These advanced vectorization methods rely on different geometrical and mathematical primitives, and use different optimization techniques requiring human assistance

[4]This is also how we will handle videos. See 3.1.3.1 for more details.

or not, but they all have in common the purpose of reaching the best possible output quality, yet this quality is mostly judged subjectively. Methods like [SLWS07] are really efficient at representing a simplified image with a compact information, but objective quality measures also care about visual details that are hardly captured. This is why, despite what is sometimes claimed in the concerned papers, the compression abilities of these methods are not that high when dealing with actual rate/distortion measures.

2.2.2 Geometrical images and compression

Pure vectorization methods are not competitive image and video compression methods, but a few previous studies have already introduced the use of higher-order image description for compression purposes. At a low-level, E. Le Pennec and S. Mallat have improved the spectral image coding by using orthogonal bandelet bases [PM05]. In JPEG and JPEG2000, the intensity variations are analyzed with respect to the main axis X and Y (and Z for videos) which is justified by complexity issues. In these approaches the segmentation is canonical, so no special signalization is required since the decoder knows how the data will be transmitted. On the other hand, no matter what the pictures hold for objects and colors, they will be segmented and processed exactly the same way. Orthogonal bandelet bases are spectral analysis functions meant to show more adaptation to the image content. They are constructed by dividing the image into regions inside of which the geometric flow of the image is parallel, and thus easy to describe frequentially. Simply put the geometric flow of the image is recovered, and the spectral coding is performed along this flow, where the image has simple regular variations. The rate/distortion performance of bandelet coding are better than what JPEG2000 can achieve, especially for images with a significant geometric flow like the Barbara picture[5]. These good results are obtained at the expense of a higher theoretical and computational complexity.

The bandelet structure remains a limited adaptation to the geometry of the image. We want to introduce more adaptation freedom by using simplicial complexes (2D triangular meshes for images, and 3D tetrahedrized meshes for video). Attempts of using geometrical objects like meshes to compress visual information have already been made. An important and visible contribution is the support of two-dimensional meshes in the MPEG4 standard [6]. The standard fully describes how such an information stream has to be coded to be efficiently compressed, but does not standardize a method to create this kind of information from raster graphics source. Consequently, this feature is hardly used in practice because of the difficulty to generate mesh-based image and video content. Another contribution to the topic "mesh-based image and video compression", is the research effort dedicated by a team from the IRISA institute to mesh-based image

[5] *http://www.mathematik.uni-ulm.de/numerik/research/wavelets/barbara.gif*
[6] MPEG4 part 2: ISO/IEC 14496-2

and video coding [MPL00a, MPL00b, CPM05]. [MPL00a] is the closest research contribution to what we have tried to develop during this Diplomarbeit. The paper, called *Mesh-based scalable image coding with rate-distortion optimization* [MPL00a], describes a still image coding scheme based on the construction of a two-dimensional hierarchical mesh. The structure is a quincunx triangular mesh that can be adaptively refined by splitting cells hierarchically, on a multi-layer scale. This adaptive splitting scheme is driven by a rate-distortion criterion. The vertices of the mesh carry a color information and the image intensity is reconstructed by performing a linear interpolation on the mesh. The other work from this team [MPL00b, CPM05] concern the extension of this idea to video coding with 2D deformable meshes.

The principles we base our work on is actually similar to theirs, although we became aware of their contribution after beginning our study. Indeed, we also use triangulated meshes to segment images and interpolation techniques to encode color variations. Yet important theoretical and technical points differ: [MPL00a] describes an image coding scheme based on a strict hierarchical 2D mesh whereas we choose to work with unconstrained and flexible Delaunay meshes ; for videos they use 2D deformable triangular meshes while we experiment tetrahedral mesh-based video coding ; and finally the representation of the color information is different: they opt for a piecewise affine representation whereas we choose a discontinuous quadratic description.

As we have seen throughout this section, the idea of a geometrical representation of images and videos is not new. Yet out of the references that have been named, only a few stress the importance of the compactness of these data, and treat it as a rate/distortion issue. The similarity between our approach and the one exposed in [MPL00a], as well as their technical differences would make it interesting to compare both schemes.

2.3 Tasks and tools

2.3.1 Nature of the work

As mentioned several times, the main task associated with this Diplomarbeit is to test the performances of an alternative visual information representation. This vague description of the work involved several more precise subtasks:

- clarify the basis of the picture representation scheme,
- design a conversion method to translate traditional pixel-based images and videos into the desired representation,
- optimize the representation and the conversion step with respect to a rate/distortion criterion (the reasons why we focus on the compression aspect are exposed in),

- and evaluate the performances of the scheme.

The nature of the work required from these tasks was double-sided:

The basis activity was a research type of work: since the approach was new to us, a way to apprehend the conversion and the optimization steps had to be developed. Indeed, if the structure of the new representation was quite clear from the beginning, its construction remained an open issue. It was then necessary to segment this construction process into steps that could be formalized. The formalization helped relating our problems to equivalent tasks that had already been investigated either in the image processing field or in the computer graphics community. A method to solve each task could then be imagined by adapting the most interesting ideas to our needs. A theoretical answer could be given to the question of the construction and the optimization of the geometric representation of visual information. As important as it is to formalize a problem before trying to solve it, the actual performances of the representation cannot easily be predicted from the theory. A real implementation of the process is necessary to judge the relevancy of the theoretical approach. This leads to the second main activity of this Diplomarbeit: development.

The implementation of the representation conversion was realized mainly using the C++ language. Our purpose was to construct a working prototype based on the theoretical background. The disadvantages of C++ for prototyping activity like ours are well-known: the complexity, a rather steep learning curve and, as a result, long development times required to translate an idea into a working program. From the prototyping point of view, the choice of C++ is not optimal compared to other solutions like Matlab. Nevertheless in our case, other factors had to be accounted for: The large algorithmic complexity requiring fast execution speed, the possibility to use third party libraries and tools already optimized to perform general task like geometry processing or linear systems solving, and the production of a program-independent code. Therefore, C++ was probably the wisest choice one could have made in this situation. It executes faster, reducing the waiting time due to computing, but also introduces low level problems in the design process that are sometimes not easy to spot and solve, and that slows the process of reflexion.

2.3.2 Tools

When developing programs associated with a research project in the team ALICE, a working platform called “Graphite” is used. Graphite has been developed, used and maintained by the team throughout the years and has become a large project of more than 100000 code lines. It is a versatile environment that allows the simplified integration of tools and third party code to a project, and a semi-automatic generation of user-interfaces. All the presented visual and numerical results derives from data generated by the C++ project, embedded in the Graphite environment.

Specialized software libraries linked with the Graphite environment were used for this Diplomarbeit:

- OpenGL[7]: The famous open graphics library to manipulate and display 2D and 3D graphical objects.
- CGAL: Computational Geometry Algorithms Library[8]. This library proposes the implementation of general geometry processing tasks like the manipulation of triangulated and tetrahedral meshes.
- LAPACK[9]: Its numeric solver has been used for the resolution of linear equations systems.

Interfaces to other languages have also been used to perform special operations:

- Recently, it has become possible to manually program the operations of the processing unit located on the graphics board (GPU). We have taken advantage of this possibility by assigning part of the treatments required for the rendering of the images to the GPU. This accelerates the rendering step, and frees the central processing unit for pure conversion operations.
  The GPU programming has been done with GLSL (Graphics Library Shader Language), a language maintained by the supporters of OpenGL.
- Interfaces between C++ and Python have also been established to use the scripting abilities of the latter.

2.3.3 Time allocation

Six month have been dedicated to this study on the *image and video compression by geometrization*. This time period includes the theoretical work and the software development. It is important to note that this work benefited from existing material. The ARDECO image segmentation method, mentioned in 2.1, basically set the basic ideas at the origin of this Diplomarbeit. The concept of a an image representation based on triangulated meshes, which we will study in report, originates from ARDECO. In addition to fundamental theoretical leads, software code developed for ARDECO was used as a starting point in this Diplomarbeit. Nevertheless, the triangulation used by ARDECO was thought has an intermediate processing step, and thus has a very different purpose than the one we will present later in this report. Ultimately, the conversion of a raster graphics image into our alternative representation uses a very different mesh generation process than what was implemented for ARDECO. Although we got a substantial experience out of the existing methods, important research and implementation efforts

[7] *http://www.opengl.org/*
[8] *http://www.cgal.org/*
[9] *http://www.netlib.org/lapack/*

were made, even (and especially) for images. Among the tasks of this Diplomarbeit was also the goal to extend the geometrical interpretation of visual information to videos, for which nothing concrete had been developed before. Consequently, the study of geometrical video representation and the development of the associated programs represented an important share (about the half) of the 6 month-period. The other half of the Diplomarbeit was dedicated to the design of a rate-distortion optimizing conversion scheme, and to the development of an efficient compression method. As we will see along this report, these general concepts comprise multiple intermediate methods and processes, that all had to be studied and implemented. To keep this report reasonably short, only the most visible and interesting points will be developed. There are also cases, where multiple approaches have been investigated to solve a specific problem. They are not all exposed in the report, but also demanded a substantial time investment. In pure development effort, the size of the code written within the framework of this Diplomarbeit is estimated to more than 20000 lines of code.

Chapter 3

System description

3.1 Representation of images and videos

3.1.1 Overview

We aim at providing an alternative representation of images, based on higher level primitives. These primitives consist of larger patches on which mathematical equations describe the variations of the visual information. This can be seen as an abstraction process: from multiple pixel informations, which are captured and stored independently from each other and consequently hold no semantic informations about the image, we want to reconstruct areas where the intensity and the colors are defined by a common information, which consequently carries an indication of the evolution of the image in that area. In 2.2 we have seen that orthogonal bandelet bases, takes the geometrical flow into account to improve the image description. We now want to take this idea a little further: the tasks during the image conversion will be to isolate areas in images and videos where the signal has a simple form, in order to describe the area with little information. In a sense, it is a continuation of the principle initiated by the bandelets, where we try to give even more freedom to the geometric shapes that can be identified.

Generally speaking, the representation of images and videos will be the following. The visual object structure will have the form of a mesh. Two-dimensional images will rely on surfacic triangular meshes while videos will be handled as 3D space/time volumes, and will be represented by a three-dimensional tetrahedral mesh. The meshes are not constructed by a direct method, but in an iterative fashion by successive approximations. The elements of these meshes (cells, edges, vertices) will carry the visual information in the form coefficient sets. These coefficients define a polynomial approximation of the color variation information inside each cell. The original pixel-based intensity variations describing the visual content of an area spanned by a cell will be replaced by a set of polynomial equations modelling this intensity. As we have seen in 2.2, this way

of representing images is not new. ARDECO [LL06] used it as a segmentation working basis, and [MPL00a] tested the potential of hierarchical mesh-based visual information for compression purposes. In the same scenario, we are willing to test a somewhat different and more flexible geometrical architecture, and extend it to videos.

More details on the structure we use to represent images and videos can be found respectively in 3.1.2 and 3.1.3., In 3.2, the stress is put on how this representation can be computed from a raster graphics source. Two tasks arise from this process: building a suitable mesh and expressing the color variations of each cell of this mesh. These aspects are presented respectively in 3.2.1 and 3.2.2. Finally in 3.3, we concentrate on the compression aspect of the conversion.

3.1.2 Geometrical representation of images

As introduced above, our purpose is to segment an image into areas that can bear higher-order information. Since we want to allow a fair degree of freedom for the positioning of the cells, they need to be easy to handle. Moreover, the geometric segmentation of the image will need to be signaled to the decoder, but in a compression context we cannot afford to generate long bitstreams only to describe the image geometry. As a result, the form of the cells must remain simple so that the geometric description of a picture can be done in a compact way. The structure that combines the most advantages is the 2D triangulated mesh structure. Although it may not be optimal to represent shapes like squares usually present in images, it has the capability to adapt to a whole lot of geometric shapes, more or less regular. On top of that, if handled properly, it can rely on methods and properties like the Delaunay triangulation and the dual Voronoi diagram which make their use practical, and allow us to bind the expectations and results with a mathematical theory. This gives the structure an important advantage over, for example, quadratic meshing techniques[1]. We also impose the triangulated mesh to be Delaunay[2]. This means that for a given set of vertices, the connections between them to form the triangulated mesh are dictated by the Delaunay conditions and cannot be explicitly chosen. From a flexibility point of view, we lose control of a degree of freedom, but this postulate was motivated by attractive compensations. Her are the two main interests of using Delaunay meshes:

[1] [SLWS07] approximates the geometry with quadratic meshes, but in their case, a fair share of the operations as to be performed manually

[2] Delaunay is a special property of meshes, that carries the name of the mathematician that first studied it (*Boris Nikolaevich Delaunay*). A simplex mesh in dimension n is composed of cells, which are n-simplices (a 2-simplex is a triangle, and a 3-simplex is a tetrahedron). For every cell we can construct its circumcircle which the circle passing through the n vertices of the cell. A mesh is said to be Delaunay if, for all cells, the circumsphere contains no other vertices than the ones forming the cell. The exact definition of Delaunay's features and their justification will not be developed in this report. See [Krä95] for more informations on the subject.

- Delaunay meshes can guarantee a certain degree of quality for the cells. In a iterative scenario where the mesh is repeatedly modified, the management of cell's shape and degeneration can become complicated. The Delaunay condition prevents important cell distortion from occuring, thus optimizing the use of the given resources.
- Given a set of vertices, the Delaunay triangulation is unique[3]. In an image storage scenario, this means that the reconstruction of the image structure requires only the vertex set. The connection information is reproductible at the decoder side without additional signaling. A saving of bitrate is realized.

The basic structure of the image format is then set. The primary elements replacing the pixels of a traditional image format are the triangles of a 2D Delaunay mesh. In traditional pixel-based image representations, the pixels hold a single color coefficient, usually a RGB or YCbCr color vector. The agglomeration of these pixels altogether creates the impression of a continuous visual information. Our larger basis elements are meant to carry more information than a single color vector, they must indeed be able to describe more or less simple forms of image variations. In theory, one could think of several ways to represent the colorimetric informations for our cells. This can be formalized by the use of functions basis. A function basis is a family of functions defining a function space. All functions located in this space can be written as a combination of the basis functions. As an example, let us remind that every periodic function in $\mathbb{R}$ can be written as a combination of cosine and sine functions (Fourier decomposition). Our purpose will be to choose a function basis and model the color variations in each cell by a combination of the basis functions. Let us use mathematics to clear this all.

Let a function basis be:

$$(\phi) = (\phi_1, \phi_2, \phi_3, ..., \phi_n)$$

each vector ϕ_i being a bivariate real-valued function, $\phi_i = \phi_i(x, y) \in \mathbb{R}$.

Inside each triangular cell, each $\phi_i(x, y)$ defines a particular intensity pattern. The modeled intensity in the cell $\hat{I}(x, y)$ will be a weighted sum of these patterns. The color fitting step will consists of finding the weighting coefficients $(\alpha_1, \alpha_2, \alpha_3, ..., \alpha_n)$, so that the original cell color variations $I(x, y)$ are properly approximated by the following combination:

$$I(x, y) \approx \hat{I}(x, y) = \alpha_1.\phi_1(x, y) + \alpha_2.\phi_2(x, y) + ... + \alpha_n.\phi_n(x, y) = \sum_{i=1}^{n} \alpha_i.\phi_i(x, y)$$

The ϕ_i functions determine only a real-valued intensity variation. As usual color are determined by three color channels (Red, Green and Blue). The color modelization process will be repeated for each of the color channel, and the final color model will be defined by $[\hat{I}_R(x, y), \hat{I}_G(x, y), \hat{I}_B(x, y)]$.

[3]This is not exactly true: see 5.1.2 for more details

The process of functional-based representation is actually common in image processing and in image and video compression in particular. Indeed in JPEG and MPEG compression a space-frequency transformation is used, generally the DCT[4]. In these schemes, each block (typically 8×8 or 4×4 blocks) is represented by a bunch of parameters called the frequency coefficients, which are actually the weighting factors of a linear combination of basis functions (the basis functions depend on the space/frequency transformation being used). The aim of this operation is to be able to represent a group of pixels more efficiently in the new vector space formed by the functions. This is exactly what we want to do here, except that instead of regular pixel blocks we project more general groups of pictures elements onto the function basis.

One could think of several relevant function basis that could be used here. For simplicity reasons, as a first working basis, we turned to the polynomial function basis. What we call the polynomial basis is a function basis whose vectors are the canonical polynomial functions. The color variations in the cells of the Delaunay mesh will be modeled by weighted sums of these basis functions. In the special case of a polynomial basis, since the sum of polynomial functions remains a polynomial, it simply means that we will try to find the polynomial that best represents the original intensity variations in each cell. From now on, we will refer to the process of finding the weighting coefficients defining the polynomial as "color fitting". As we will explain later on, our goal in the image segmentation process is that the Delaunay cells avoid to straddle intensity discontinuities. If done so, the color variations in the cells should remain smooth, and the polynomial that is supposed to model it is simple. Practically speaking we will limit the degree of the polynomial functions in the basis, and work with the following function basis:

$$\begin{cases} \phi_1 = 1 \\ \phi_2 = x \\ \phi_3 = y \end{cases}, \begin{cases} \phi_4 = x^2 \\ \phi_5 = y^2 \\ \phi_6 = x.y \end{cases}, \begin{cases} \phi_7 = x^3 \\ \phi_8 = y^3 \\ \phi_9 = x^2.y \\ \phi_{10} = y^2.x \end{cases}$$

In practical applications, the use of 3rd degree components leads to insignificant visual improvement in the image compared to the number of additional coefficients that are to be fitted and encoded, so ϕ_7 to ϕ_{10} are mostly put aside. Degree 0 and degree 1 image approximations can be used if reduced sets of the function basis, respetively (ϕ_1) and (ϕ_1, ϕ_2, ϕ_3), are used. These low-degree approximations provide interesting coarse approximations of the image at a low expense, but the best rate/distortion performance however is almost systematically obtained with the second degree color modelization (ϕ_1 to ϕ_6). All visual and numerical results presented in this report rely on a second degree color model. The polynomial approach combines simplicity and reasonable results. They are particularly efficient when it comes to representing smooth varying intensity

[4]Discrete Cosine Transformation

with a low bitrate. Their representation abilities are yet limited when textures appear in an image. The introduction of more evolved function bases might improve the performance in those cases. For smoothly varying area however, the polynomial function basis, combined with a appropriate segmentation, is well-adapted representation in terms of rate/distortion performance. Discussions on the relevancy of the chosen function can be found in 7.2. For now we stick to a complete polynomial modelization.

Figure 3.1 illustrates the structure used to represent images:

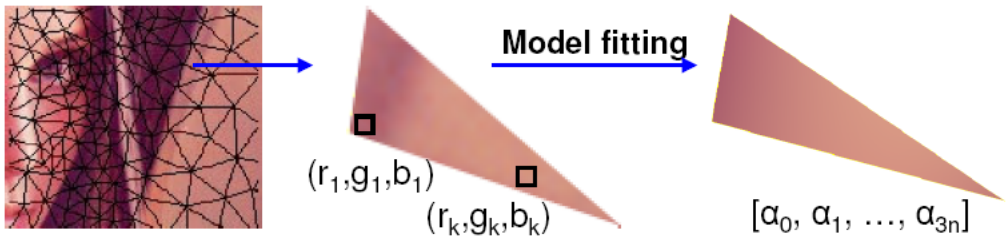


Figure 3.1: The geometric representation of a 2D image. The picture consists of triangular cells forming a Delaunay triangulation. Each triangular cell possess 3 sets of values $(\alpha_1, \alpha_2, \alpha_3, ..., \alpha_n)$ which are the weighting coefficients of the polynomial basis functions for each color channel.

3.1.3 Geometrical representation of videos

3.1.3.1 Video object structure

Usually, videos are considered as a temporal sequences of 2D frames. From this point of view, the adaptation of the still image geometric representation described above would result in using deformable triangular meshes. This approach is not new, the support of deformable meshes for video object has even been standardized in MPEG4. If it benefits from obvious advantages, like an efficient and cheap motion compensation and a strong semantic representation of the video content. Important practical difficulties unfortunately arise when it comes to converting a pixel-based video into a 2D deformable mesh. Alone the problems of objects' apparitions, disappearance and overlapping are a very challenging tasks that still has not been efficiently solved. This object-based video representation is probably the best theoretical representation we have so far, but its construction remains a problem. What we propose as a new video representation has a weaker semantic link with the video content, but can be more intuitively constructed as a geometrical object. This representation is actually the exact adaption of the still image representation (see 3.1.2) for videos, except they are not considered as sequences of images, as mentioned in the beginning of this section, but as a real three-dimensional

object. Intuitively, the first two dimensions are the spatial dimensions X and Y describing a frame, and the third one Z is the time. Despite their semantic differences, we treat them equally during the conversion process. The object we want to convert is a video volume composed with voxels[5]. Two adjacent voxels in the X or the Y directions are the two adjacent pixels of the corresponding 2D frame of the video, and two adjacent voxels in the Z direction are the value of the same pixel position at two successive time instants. Figure 3.2 visually describe how the spatiotemporal video volume is formed out of the consecutive frames.

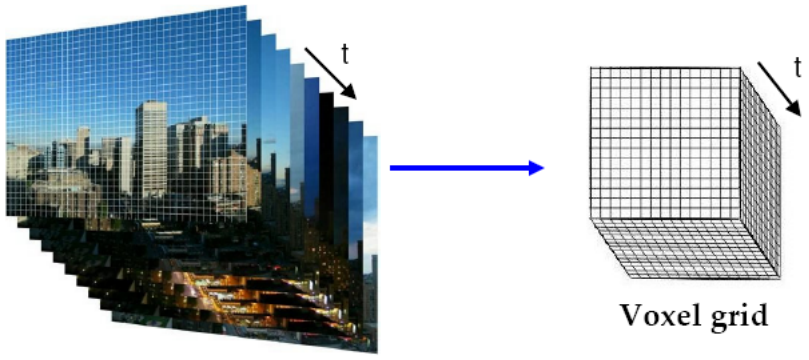


Figure 3.2: The video volume. All frames of a video sequence put one after another form a volume of intensity variations. The pixels of the frames become the voxels of the video volume.

We will now process the video volume exactly how we processed the image surfaces in the previous section: We will segment it into cells, and model the original color variations by a vector in a polynomial function basis. For images we used triangles, which are the simplices in a 2D space. The motivation for this have been exposed in 3.1.2. For the same reasons, we will use 3-simplices to segment the video volume. the three-dimensional simplex is a tetrahedron. We will rely on the theoretical basis of the Delaunay tetrahedrization of space to build up the segmentation of the video volume, as illustrated on figure 3.3. As far as the modelization of the colorimetric information is concerned, the technique will be similar as the 2D case. We will project the intensity functions on a polynomial function space, and store the coefficients weighting the contributions of the basis functions. The only difference is that we now handle trivariate functions.

[5]Voxel = VOlume ELement: derived from the denomination “pixel” = PICTure ELement in image processing.

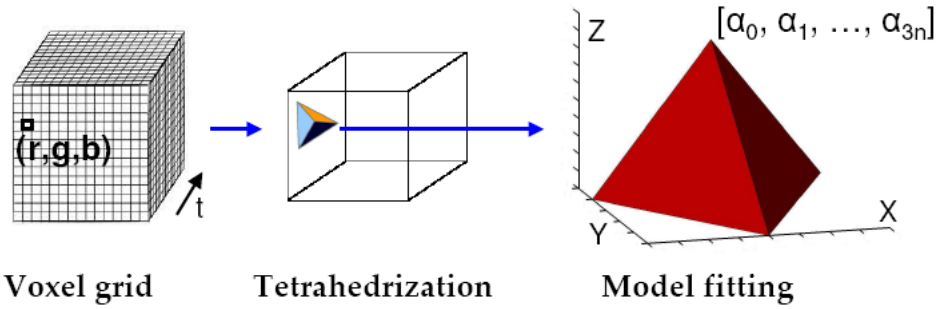


Figure 3.3: The geometric representation of a 3D video volume. The volume is composed with tetrahedral cells forming a Delaunay tetrahedrization. Each tetrahedron possess 3 sets of values $(\alpha_1, \alpha_2, \alpha_3, ..., \alpha_n)$ which are the weighting coefficients of the polynomial basis functions for the three color channels.

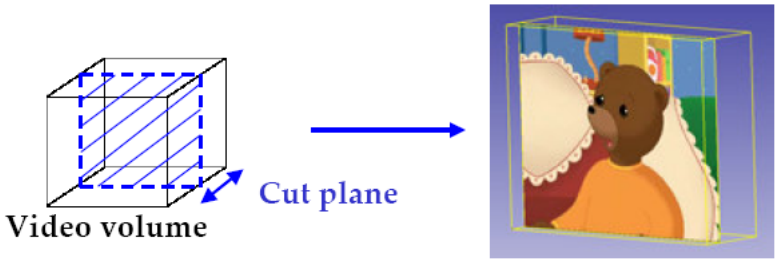


Figure 3.4: playing a video. The frames of the video are the intersection between a cutplane (equation Z=t) and the video volume. We "play" the video by translating the cutplane back and forth. Note that a temporal interpolation automatically occurs: The video is continuous in space and time.

3.1.3.2 The motion issue

Even if it is treated like one by the conversion process, the third dimension of the video volume does not behave like a spatial dimension. The major breakthrough that occurred in video coding with the design of MPEG codecs rely precisely on their ability to take advantage of time redundancies. In natural scenes, objects have continuous movements, which means that their positions in two adjacent captured frames are relatively close. The information needed to reconstruct the concerned part of the image can then be reduced if the redundancy is identified and correctly signaled (through a block-matching algorithm for example). The gain in rate/distortion performance is so important compared to Motion-JPEG that no video coding scheme would stand a chance without taking good care of the motion compensation process. Here we want to point out here that this tetrahedral representation of videos implicitly performs a form of motion compensation. Indeed, as we will see in 3.2, the cells that compose the image and video object are not randomly placed throughout the available domain, but follow a placement rule meant to provide the best possible quality. This rule places the edge of the cells on the discontinuities of the source picture intensity. When an object, a ball for example, moves throughout a video sequence, its trajectory forms a volume in our $X, Y, Z = t$ space. What the segmentation scheme will try to do is place the edges of the tetrahedral cells on the boundaries of this volume. Thus the cells representing the inside of this volume shall follow the object's trajectory in the Z direction (temporal dimension). One cell then implicitly models a part of the moving object over several time instants, which could be regarded as "motion adaptation".

As we will see, however, the addition of a third coordinate and the representation of motion is the source of particular problems (see 6.2 for more details).

The debate we had about triangular cells to segment images is also relevant in the 3D case: one can wonder if tetrahedrons are the best way to segment a 3D volume, especially when motion is concerned. Indeed, tetrahedrons have the advantage that they can easily adapt to non-regular shapes, and their simplicity prevent them from creating anomalies like non-convex cells, which is not the case for quads for example. On the other hand, their structure limits them in representing moving objects in the 3D volumes since their intersection surface with a moving cutplane cannot be constant in size (contrary to quads). Thus more tetrahedral cells than cubic cells are necessary to correctly display a static rectangular shape in a video, but again tetrahedral meshes like triangular 2D meshes benefit from the Delaunay contribution and the relation to Voronoi diagrams. This makes them practical to generate, control and optimize in an iterative scenario.

3.2 The conversion process

In the above subsections, we have described the image and video representation we would like to use. This description was basically focused on the end result, but not much on what must be actually done to get it. The main challenge is to construct these objects from the traditional pixel-based image and video representation, and see how they behave and what we can do with them. The main operations required to perform the conversion will be first presented, and then the conversion process itself as a combination is these operations, will be exposed. The description will be generic, and thus concern both the image and the video conversion process.

Figure 3.5: A basic conversion pipeline. The different operations are executed sequentially and independently from each other.

3.2.1 Mesh construction

Probably the most crucial point in the conversion is the segmentation. It has been said in 3.1.2 that an image or a video consists of Delaunay cells baring color information. The placement of the cells is of course seriously supervised. The color and intensity variations inside a cell will be approximated by polynomial functions. The dynamic of such functions is limited, and is only exact when the original colors obey to a polynomial law. As one can imagine, this does not often occur. The polynomial approximation will then create visual distortion, which we want to keep as low as possible with respect to the available bitrate. Polynomials, which are smooth functions, are particularly inadequate to cope with intensity discontinuities whether spatial or temporal. To ensure that the cells cover intensity variations that are as smooth as possible, an adapted strategy is to attract the edges of the Delaunay cells on the images' discontinuities. This prevents this discontinuities from falling right in the middle the cells causing heavy distortions. For 2D images, the edges of the cells are line segments and the discontinuities are curved lines and we aim at placing these segments on the discontinuity lines. The 3D video case is somewhat more complicated: the frontiers of the Delaunay cells are the faces of the tetrahedrons, and the discontinuities are 2D surfaces in the 3D volume. The faces of the tetrahedrons have to be snapped on the discontinuity surfaces.

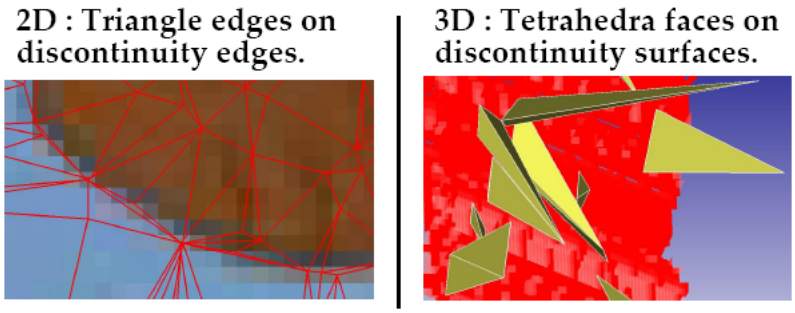


Figure 3.6: Cells placement strategy: the edges of the cells need to snap on the images' discontinuities. This way they are likely to cover smoother areas and create lower distortion.

To be able to snap the edges on the discontinuities, they must first be detected. This can be done by using classic edge detection filters with the pixel-based representation of the image or the video. Details on the principles and use of edge detection filter can be found in 4.1.1. The result of an edge detection filter on a still image is well-known, but it is less common to filter a video volume. In that case, it is important to use a complete 3D edge detection scheme, similar to the one used in 3D medical imaging for example. Indeed, a static or moving object in a video spans a volume in the 3D video object, whose outer surface we want to detect. This surface is the frontier between the inside and the outside of the objects trajectory in the 3D volume, and that is exactly where we want to place the edges of the Delaunay cells to minimize the presence of color discontinuities in them.The outcome of a 2D edge filter that would simply be repeated for each frame is not the same as a real 3D edge filter, and would not correctly detect the discontinuities as 2D surfaces in our 3D volume, which is why a full 3D edge has to be used at the expense of a higher execution time.

The outcome of our edge detection step we call the "saliency map". It will be the basis of the cells placement strategy. When constructing a 2D or 3D mesh we dispose of two degrees of freedom: the position of the vertices and the connections between these vertices. As mentioned earlier, we chose to work with Delaunay meshes only, which fully dictates the way to points are going to be connected with each other in the final mesh. The choice of the Delaunay graph was actually made to avoid the necessity of encoding the mesh's connections as a coding parameters. This means we are left with only one degree of freedom to entirely determine the mesh: the positions of the vertices. In order to maximize the probability of having cells' edges on the discontinuities of the pictures, the vertices themselves have to be placed on these discontinuities. Placing a defined number of points on a surface, or in a volume, so that they are reasonably scattered on the whole domain and obey to our strategy is not a simple task. We decided to use a stochastic approach: The Centroidal Voronoi Tessellation (CVT). The way of constructing a CVT is very similar to a Lloyd-Max relaxation scheme as used in the design of optimal quantizers [Llo82]. The principle of this algorithm is the following: Points are

first placed at strategic initial positions in the domain, then their positions are independently updated within a local neighborhood[6]around the vertex. The neighborhood, and thus the update efficiency, depends on the distribution of the vertex's nearest neighbors. The update process is of course supposed to let discontinuities present in the neighborhood attract the vertices. One update step is not enough, and we have to run through this process several times in order for the vertices to reach a stable position. Our goal is fulfilled since the vertices have been attracted to the discontinuities, and we also ensured a fair egalitarian scattering of the vertices throughout the domain since vertices can only move inside a limited zone (the neighborhood) within a relaxation step, while no other vertex has access to this zone. This prevents the vertices from all converging towards the biggest saliencies in the domain while the smallest but also important ones are left unrepresented. The details of the CVT are described with more precisions in 4.1.2.

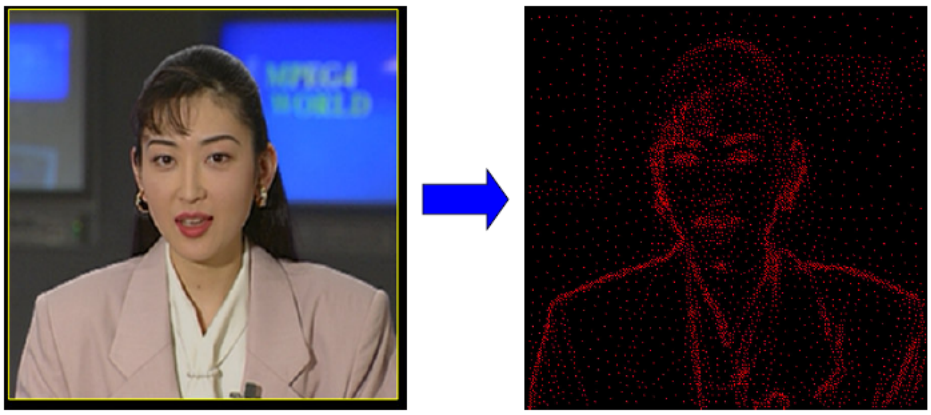


Figure 3.7: The vertices of a 2D mesh after a Centroidal Voronoi Tessellation (CVT). They are scattered all over the image and snapped on its discontinuities

We recall that our goal was to place the edges of the mesh on the discontinuities of the source signal. Influencing the positioning of the vertices and placing them on the discontinuities of the images will tend to form edges at the right spots, but it does completely guarantee it. Indeed the Delaunay triangulation or tetrahedrization we use is completely unconstrained and must remain so, otherwise the modifications to the Delaunay algorithm would represent information that would have to be signaled to the decoder. No additional constraint can be added to really force the edges of the cells to correspond to the source's discontinuities. In most cases however, using a good method for the placement of the vertices results in a satisfying result for the final mesh, but problematic cases can occur, leading to unwanted distortion in the decoded visual information. See 6.2 for more details on that type of artifacts.

[6]The neighborhood mentioned here is the Voronoi cell of the vertex. See 4.1 for more details

3.2.2 Color fitting

Once the mesh is constructed, the last step is to generate the color information that will approximate the original pixel intensities of the source. A mentioned in 3.1.2, the pixel-based intensity variations are to be approximated by a weighted sum of polynomial basis functions. the approximated intensity of one color channel in a cell is then (for the 2D case):

$$\hat{I}(x,y) = \alpha_1.\phi_1(x,y) + \alpha_2.\phi_2(x,y) + ... + \alpha_n.\phi_n(x,y) = \sum_{i=1}^{n} \alpha_i.\phi_i(x,y)$$

The purpose, of course, is for the approximation to be as close as possible to the original intensity $I(x,y)$, for all (x,y) pairs inside the cell. Assuming we have $I(x,y) = \hat{I}(x,y)$ at the pixel (x,y), the above formula is an equation with n unknowns, with n being the number of functions in the polynomial basis. A cell generally contains several pixel positions, each of them delivering a linear equation with the same unknowns $(\alpha_1, \alpha_2, \alpha_3, ..., \alpha_n)$. By grouping these equations, we obtain a linear system of the form $A.\overrightarrow{x} = \overrightarrow{b}$:

$$\begin{cases} \alpha_1.\phi_1(x_1,y_1) + \alpha_2.\phi_2(x_1,y_1) + ... + \alpha_n.\phi_n(x_1,y_1) = I(x_1,y_1) \\ \alpha_1.\phi_1(x_2,y_2) + \alpha_2.\phi_2(x_2,y_2) + ... + \alpha_n.\phi_n(x_2,y_2) = I(x_2,y_2) \\ ... \\ \alpha_1.\phi_1(x_K,y_K) + \alpha_2.\phi_2(x_K,y_K) + ... + \alpha_n.\phi_n(x_K,y_K) = I(x_K,y_K) \end{cases}$$

where

$$A = \begin{bmatrix} \phi_1(x_1,y_1) & \phi_2(x_1,y_1) & ... & \phi_n(x_1,y_1) \\ \phi_1(x_2,y_2) & \phi_2(x_2,y_2) & ... & \phi_n(x_2,y_2) \\ ... & & & ... \\ \phi_1(x_K,y_K) & \phi_2(x_K,y_K) & ... & \phi_n(x_K,y_K) \end{bmatrix}, \overrightarrow{x} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ ... \\ \alpha_n \end{bmatrix}, \overrightarrow{b} = \begin{bmatrix} I(x_1,y_1) \\ I(x_2,y_2) \\ ... \\ I(x_K,y_K) \end{bmatrix}$$

Generally, the matrix A is a tall matrix, meaning it has more rows than columns. The most commonly used function basis is the 2nd degree approximation, where the canonical polynomials up to degree 2 are used. This implies $n = 6$ for the image case, and $n = 10$ for the video case. The Delaunay cells usually contain a lot more than 10 pixels, making a tall matrix out of A. The problem to solve here is an overdetermined linear system. A straightforward approach to retrieve the alpha coefficients of the cell is to use the least squares solution of the linear system. It is well adapted to this problem

and benefits from a direct solving method: the pseudo-inverse. Despite all these advantages, the least squares solution has weaknesses too. The most important is certainly the lack of robustness towards outliers. If, for some reason, a pixel that is not supposed to be in a cell brings its contribution to it, the whole least squares solution will be affected. This means that a local outlying pixel globally influences visual representation of the cell. This situation occurs because the cells placement during the mesh construction step inevitably implies discontinuities pixels being assigned to a cell, as it can be seen on figure 3.8.

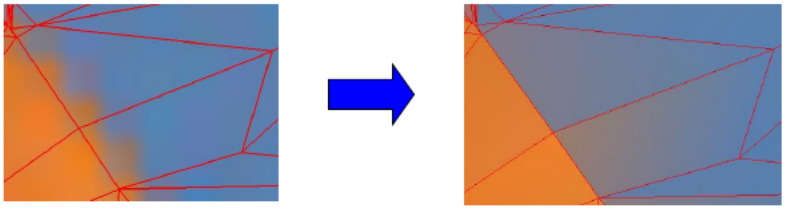


Figure 3.8: Fitting artifact: since edges in an image are not infinitely thin unlike the edges of the mesh (between two Delaunay cells), the discontinuity pixels necessarily fall inside cells that are not meant to represent them. This results in inappropriate shadings in the cells, impairing the final quality.

To overcome this problem, a more sophisticated color fitting scheme had to be introduced. It is based on the M-estimator technique, and provides an enhanced robustness towards outlying pixels, thus improving the rendering of the cells a good deal. The details on the successive color fitting approaches are detailed in 4.2.

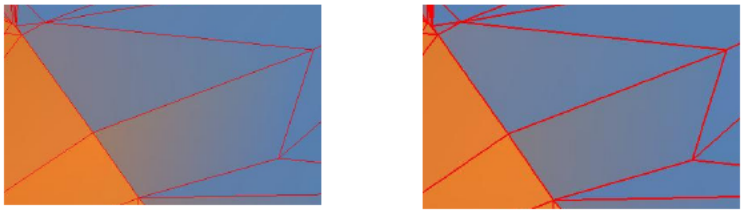


Figure 3.9: The consequences of using a robust estimation scheme instead of the straightforward least-squares solution. Left: the least-squares fitting solution. Right: the robust fitting solution. The influence of outlier pixels from the discontinuity is reduced.

We note that, during the color fitting process, we chose to process the cells independently from each other. Nothing explicitly guarantees that new color values at the common edge between two cells are equal to each other. The consequences of this choice are discussed in 3.3.2.

3.2.3 A rate-distortion driven conversion

A purely sequential conversion process has been graphically presented in figure 3.5. This method has the advantage of being fast, but is far from being optimal with respect to the use that is made of the cells. We see on the conversion pipeline that the geometrical part (mesh construction) and the colorimetric part (color fitting) are completely separated. This implies that the number of cells that will be used to segment the domain has to be given *a priori*[7]. The computation of this number can be driven by the dimensions of the image or the video , or by a quick analysis of its content, but it will not necessarily be the optimal number with respect to the rate/distortion performance. The other problem occurring in the mesh construction process is the question of the initial positions of the vertices. The relaxation method used converges towards a local minimum as stochastic methods usually do. The starting state of the mesh has an important influence on the final relaxed result. When one has n points at disposal, the way they will be scattered around the domain before starting the relaxation makes a big difference for the final mesh topology, and thus indirectly for the quality of the representation. There might be too many cells at places in the domain where fewer would have done just as good in terms of quality, and consequently too few cells where a finer segmentation is required. The purpose is obviously to have as few cells as possible for a given visual quality. Any additional cell induces a non-negligible cost, especially at low rates, and we cannot afford wasting resource by placing cells at the wrong places. Therefore a different conversion approach has to be considered. It will aim at introducing more segmentation cells where they are needed, and if they are needed.

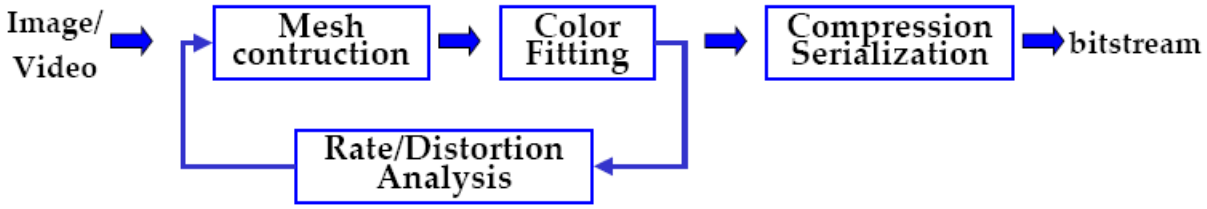


Figure 3.10: Rate/distortion driven conversion. The image is approximated successively with a growing number of cells. At each step, new cells are introduced where the representation quality is not good enough.

An initial number of vertices still has to be given in this scheme, but it will not stay fixed since the process is supposed to adapt the number of cells to the complexity of the image to convert. The initial number of vertices is then chosen to give a very coarse version of the geometrized object we can start working on. Once the mesh has been constructed and the colors fitted, a rate/distortion factor of each individual cell is evaluated. The

[7] It can be determined through different criteria: the maximum number of vertices, the maximum area that can be covered by a cell, etc.

factor is given by:

$$\lambda = \frac{cell\ mean\ square\ error}{cell\ bitrate}$$

Since the number of coefficients used per cells for the polynomial approximation is known, the number of pixels in a cell is proportional to the mean bitrate generated by this very cell. The measure of the error relies on the pixel-based SSD[8] between the original data and the current approximation. In every cell where the rate/distortion factor λ is above a given threshold λ_T a new vertex will be introduced. After a final relaxation the iteration ends, and the process starts all over again until convergence is reached. At equal number of cells, the rate/distortion driven process makes a better use of the coding resources, and delivers a better visual quality:

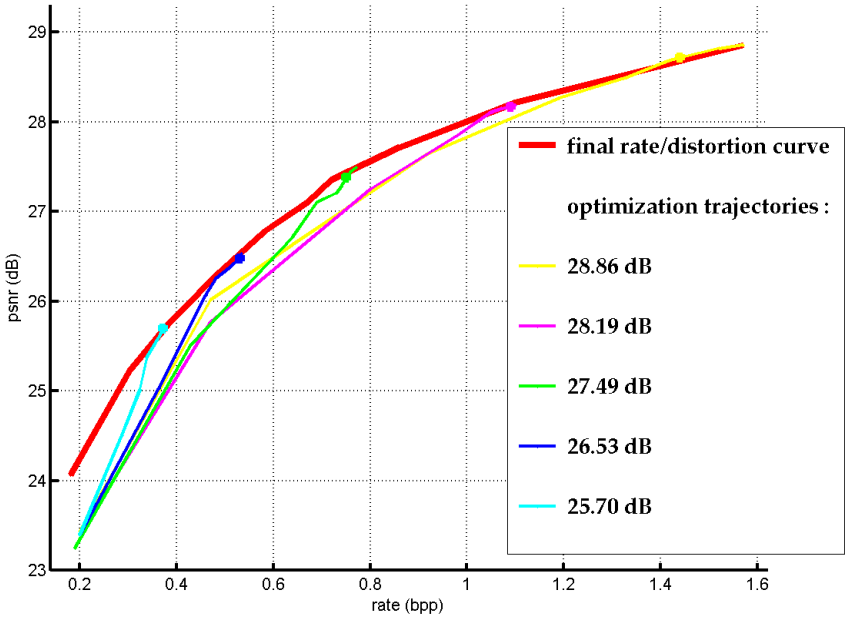
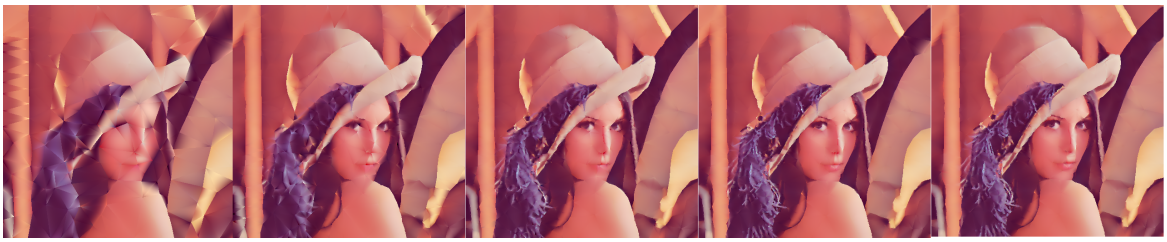


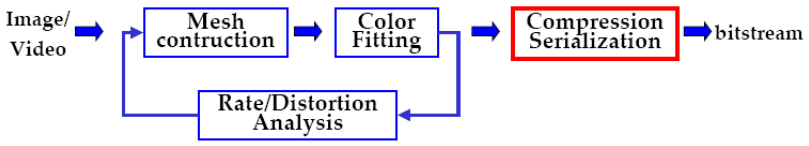
Figure 3.11: Rate/distortion trajectories of the conversion process. The curves describe the rate/distortion evolution of the image during the conversion. The dots on the trajectories represent the image state after the 6^{th} iteration.



[8]Sum of Squared Differences

Figure 3.12: Left: a sequential conversion process (no feedback), using the pipeline of figure 3.5. Right: the result of the rate/distortion driven process from figure 3.10 with the same number of cells.

3.3 The compression aspect



At this very point, we have a complete conversion method able to generate geometric image and video representations starting from the traditional pixel-based data. As it has been said in the introduction of this report, the new representation is likely to possess several interesting features, but there is one that we particularly focus on: the information compression performances of the format. Indeed, the whole approach is based on the fact that areas in a picture, or space/time volumes in a video, that contain low-complexity visual information should be awarded less precision, and thus less bits, than high-complexity ones. Moreover, compared to structured approaches like [MPL00a], the unconstrained Delaunay mesh geometry is supposed to give as much freedom as possible to the segmentation process to approximate the features of the pictures. Everything is done so that the description resources are placed where they are

most needed. This is an interesting rate/distortion trade-off strategy, but the data flow representing a converted image or video is still not optimized for compression. Let us see how we can reduce the size of the structure some more. A geometrized image or video consists of the principal data streams:

- The geometrical information (mesh),
- The colorimetric information (color polynomials coefficients).

The proportion of the contribution of each entity in the final file size may varying depending on the source data[9], but these variations are small. Before compression the ratio of bitrate occupation is approximately: 10% for the geometry and 90% for the colorimetric information. The raw rate/distortion performances at this point are still far from traditional compression methods like JPEG (there is a factor 8 between the bitrates).

From a compression point of view, the two streams will be processed separately. The geometry will be compressed using an adapted lossless compression process, whereas the colorimetric information will undergo a lossy compression step including quantization. The principle of each compression scheme is described below. For more specific details see 5.1 and 5.2.

3.3.1 Compression of the geometry

The information essential to any reconstruction of the geometrized image implies being able to reconstruct the mesh. It can be seen as an overhead without whom the actual visual reconstruction of the image cannot start. Thus the bitrate required for the coding of the mesh has to be as low as possible. Nevertheless, we cannot afford loss of information concerning the topology of the mesh. Any difference between the original mesh and the decoded mesh would right away create visible image distortion. The compression scheme used for the geometry has to be efficient, and it has to be lossless. Meshes are geometrical objects that are widely used in computer graphics applications. Although compression aspects are generally not the main concern in these applications, a few research works have investigated this topic ([TR96],[TG98],[COLR99]). The problem with these algorithms is that they are generally design to compress an unstructured mesh, which means that the position of the vertices and the connections between the vertices are to be encoded[10]. We remember from 3.1.2 that we constrain our meshes to be Delaunay which gives them a certain structure. More precisely, it makes the encoding of the connections completely irrelevant because the mesh can be entirely reconstructed

[9]Depending on the topology of the mesh, a given number of vertices can generate a varying amount of cells

[10]Actual mesh codecs often use the following approach: The vertices are encoded in a specific order such that it partially contains the topology of the mesh. If one considers encoding only the vertices positions, the constraint vertices' order induces by these codecs reduces the degrees of freedom and obviously does not lead to an optimal compression of the geometry.

from its vertices. The problem of the mesh compression is actually reduced to a point set compression task. One of the competitive approach we have chosen was designed by P.M Gandoin and O. Devillers [DG00], and performs a hierarchical geometrical lossless coding of a set of points with integer coordinates in a n-dimensional space. The integer coordinates requirement is a rather strong condition, but imposing the mesh relaxation scheme (3.2.1) to work with integer positions does not affect the results in a disturbing way. This can be regarded as a lossy step in the geometry coding process since the vertices cannot be placed wherever we want but are "quantized" at integer coordinates. However, the practical difference it makes is, in all encountered cases, not even visible. The fact that the coding of the point set is hierarchical, which allows progressive coding and decoding of a point set, is a very nice feature that can be a crucial advantage in some applications. In our case unfortunately, it does not play much of a role since we need to have the mesh entirely decoded before beginning to decode the colors. The hierarchical aspect was not a criterion in our choice, the scheme was chosen because of its good coding performances. The compression method is presented in section 5.1.1. On table 3.1 we display the quantitative compression results:

	number of vertices	original size (bytes)	final size (bytes)	compression ratio	bitrate (bpp)
lena (image)	735	1654	938	1.763	$3.58.10^{-3}$
lena (image)	1009	2271	1214	1.87	$4.63.10^{-3}$
peppers (image)	729	1641	954	1.72	$3.64.10^{-3}$
monarch (image)	1516	3601	1902	1.89	$3.22.10^{-3}$
mother (video)	37554	110128	45243	2.43	$3.95.10^{-3}$
little bear (video)	32822	95266	36782	2.59	$3.80.10^{-3}$

Table 3.1: Geometry compression performances for different 2D and 3D meshes.

The Gandoin and Devillers method is an interesting point set compression scheme that combines simplicity and efficiency. Nevertheless, if we take for granted that the coordinates of the points are integer value, we can actually forget about the semantics of the data, namely that it is a n-dimensional mesh, and establish a connection with a research field where the compression effort has been more active. Indeed, our goal is to encode a quantity of point located on a regular grid in a 2D or 3D space. The regular grid is similar to a pixel grid (in 2D) or a voxel grid (in 3D), and the information related to every pixel of this grid is whether there is a vertex at that location or not. This is a binary information. Thus, coding our mesh is equivalent to encoding a binary 2D or 3D image in a lossless fashion. Powerful compression formats have been developed for this purpose, especially for the transmission of binary image data with faxes. The Joint Bi-level Image experts Group (JBIG) has come up with the JBIG1 and JBIG2 compression methods for this purpose. They are however very specialized schemes. JBIG1 for instance is meant to send the image progressively and thus handles the data in scanline order which limits the compression possibilities. JBIG2 is more versatile and adapts its coding to the

image's content (figures, text, etc), which obviously improves the codec's performances if the semantics of the data is coherent with what the codec can handle. In our case the image data originally come from geometrical data. Text- and graphics-specialized manipulations on these data are not likely to provide the gain that is expected in normal use cases. In average over randomly generated binary images, JBIG2 and the hierarchical geometry codec have about the same compression performances. On real geometry data sets however, our hierarchical geometry codec tends to use less bits to encode the mesh, and benefits from a lower computational complexity.

3.3.2 Compression of the colors: reduction of the amount information to encode

As it has been mentioned earlier in this section, the biggest share of the uncompressed bitstream is occupied by the color information. In order for the format to be somewhat competitive, the bitrate dedicated to the color needs to be significantly reduced. For this reason, and because the quality of the color information is linked with the characteristics of the human visual system, the compression of this stream will be lossy. As a matter of fact, every high-performance image or video codec discards information from the source with regards to the specificities of the human eye. This is the only way to achieve competitive compression.

In the raw form, the colors are represented by a set of coefficient , which are the weighting factors of the functions in the polynomial basis[11]. These coefficients are floating-points values, and in the second degree approximation, 6 of them are required per color channel and per cell to completely describe the latter. If we assume a 32-bit floating point precision, we will then have $3 \times 6 \times 32 = 576$ bits per cells. At a reasonable quality, the cells segmenting the lena picture contains about 200 pixels in average, which correspond to an average bitrate of 2.88 bpp. This is completely out of proportion compared with the standard rates we obtain with other compression method. We need to lose information here, and as often in image and video compression we will use quantization as an information-removing process to achieve better rates. The problem with the alpha coefficients space, is that their dynamic is completely out of control. Indeed they are coefficients related to the constant, linear, quadratic or cubic shading of the color intensity. Depending on the cell's spread and intensity variations, their value range is very large and not easily quantizable. Very small alpha coefficients can occur as often as big alpha coefficients, both with the same importance in terms of final quality. Therefore, the first step of the compression pipeline consists of shifting the color representation towards another space: the three-dimensional digital color space. This will immediately reduce the number of bits needed to store a cell's color variations with no visible degradation, and will place us in a space where it is easier to quantize the data.

[11] See 3.1.2

The alpha coefficients of a cell define a polynomial as the weighting factors of the basis functions. Yet the canonical coefficients of a polynomial are not the only way it can be represented. Thanks to interpolation techniques, the polynomial can also be uniquely defined by a set of n sample points located throughout the cell. In our case, the polynomial defined by the alpha coefficients represents colors variations, and the colors originate from the source pictures, so they are supposed to be in the [0 .. 255] range. Instead of using the alpha coefficients to define a polynomial color variation pattern, we can just as well pick n sample points and store their intensity value. The polynomial in the current cell is uniquely defined by this set of n color values. The advantage of this strategy is that we now work with actual digital colors, which live in a limited space with known relations to the human visual system. The details of the conversion from polynomial alpha coefficients into colors are derived in 5.2.1. Instead of $3 \times n$ polynomial coefficients[12], we now have n three dimensional color vectors to encode. As it has been said earlier, second degree polynomial are the best rate/distortion compromise to represent the cells. In that case we need $n = 6$ sample points in the cell to represent the whole color variations. The points are taken on the edges of the cells, as it is usually done in finite elements methods:

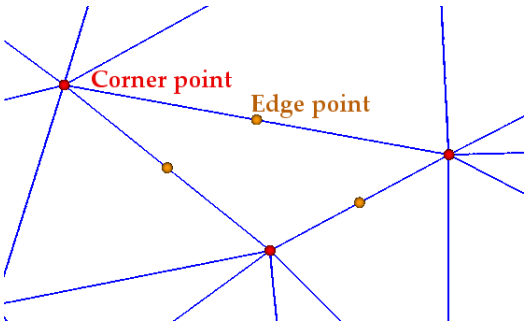


Figure 3.13: From polynomial coefficients to sample color points: For a second degree color model on images, we need 6 sample points per cell (10 for 3D cells). 3 Points are taken at the cells corner (*corner points*), and the other three in the middle of the cells' edges (*edge points*).

Since the polynomial coefficients were floating points, the sample color points values are computed with a floating point precision. The very first trivial bitrate improvement we can make is to quantize them the way it is usually done in image processing: at integer values in the range [0 ... 255]. This 8-bit representation was originally designed in a way that no visual difference between two adjacent level can be seen. Thus, not surprisingly, quantizing the sample points at integer coordinate in the RGB space introduces no visible changes. A simple change of representation causes a bitrate reduction of a factor 4. Instead of a mean bitrate of 2.88 bpp per cell, we have a 0.72 bpp mean rate for the same image quality.

[12] n alpha coefficients per color channel and 3 channels

Another process we call "selective color merging" is likely to reduce the amount of color information that has to be sent. Its purpose is to take advantage of the inter-cell color redundancy. As we see on figure 3.13, the color points of a cell share the same location as other color points from surrounding cells. It is important here to point out the difference between a vertex and a cell corner. A vertex is a geometrical point defined by its coordinates only: at one precise location in space there can be only one vertex. The corner points are defined by their location and by the cell they belong to, thus two distinct corner points can have the same coordinates but different color values. As an example: when k cell corners coincide at one geometrical location, there is one vertex but k corner points. The same goes for edge points: In the middle of an edge of the mesh there are 2 coincident egde points, each of them attached to one of the cell adjacent to the edge. This distinction is important because it allows the representation of color discontinuities between cells. If the color where attached to vertices and not to cell points as in [MPL00a], the color variations would necessarily be smooth everywhere, and color discontinuities could be correctly represented only with a very fine mesh. Yet there are areas in an image where the intensity does vary smoothly. In those areas, it is interesting to have a smooth intensity transition between adjacent cells. It is unfortunately not always the case since the color fitting step handle cells independently, and does not care about inter-cell color transitions. This can result in visible color gaps at the frontier between adjacent cells[13]. The selective color merging is a post-fitting step meant to merge the colors located at the border between cells in low-varying areas. This has two advantages: first of all it limits the visual impact of the blocking artifact, delivering nicer-looking pictures, and second of all it reduces the amount of color information to encode. Indeed if a group of k coincident cell points are merged, there is only one remaining color to encode instead of k. The disadvantage is that an overhead we call "discontinuity map" has to be spent in the bitstream to signal to the decoder where the color have been merged. The details of the merging process are developed in 5.2.2.

As one can see on figure 3.14, the subjective visual quality is improved. Other merging examples can be found on figure 5.5 in section 5.2.2. An observation we can make is that the selective color merging does not improve the PSNR (without regards to the rate improvement), as it often is when using smoothing operators. The perceptual image quality however does not meet the quantitative quality measure in that case, and is significantly increased. We can now zoom in on the compressed pictures without being bothered by the otherwise annoying cell transition artifacts. Let us now compare the rate/distortion improvement when using the selective color merging (the additional bitrate due to the discontinuity map is included).

The scheme's rate/distortion performance still lacks a compression ratio between 2 and 3 to really be competitive. Such an improvement cannot rely on lossless entropy coding alone: an additional quantization method had to be introduced. In the following, we

[13]This phenomenon is similar to the "blocking artifact" well-known in JPEG where the 8×8 blocks are processed separately, like our cells.

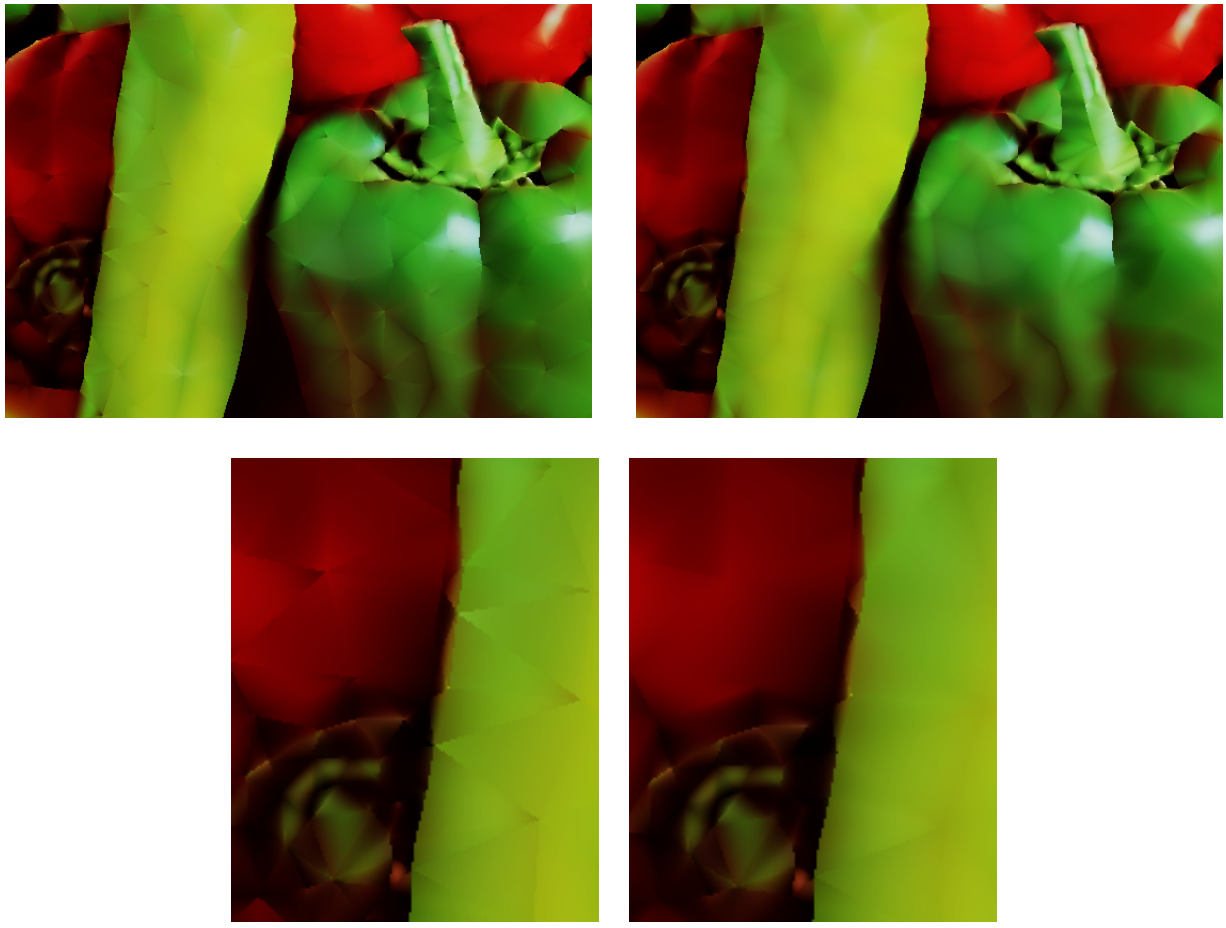


Figure 3.14: Selective color merging: the visual impact. Left: before the merging. Right: after the merging

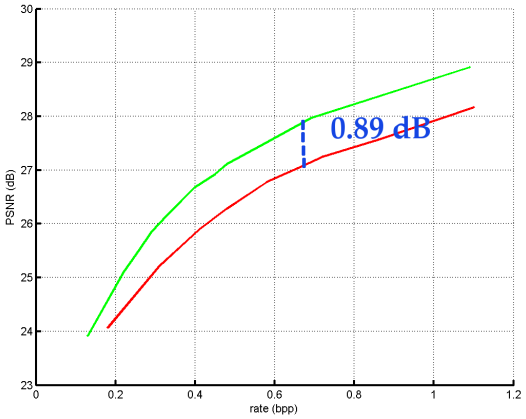


Figure 3.15: Rate/distortion performance comparison: true color non-merged vs. true color merged as RD plot

describe briefly the different approaches that were explored to further reduce the bitrate.

3.3.3 Compression of the colors: Color compression algorithms

Intra-cell differential coding

Differential coding is often the first thing that comes to one's mind when information redundancy is to be reduced in a bitstream. Redundancy can be efficiently exploited when the structure of the data is known and likely to possess redundancy at predictable places[14]. The only structure we have here is the segmentation in Delaunay cells. As seen on figure 3.13, the color information for a cell consists of n 24-bit colors. Can these be coded in a differential fashion ? A value that can act as a predictor for the colors of a cells is the cell's mean color. The approach that was attempted was to store a mean color value for each Delaunay cell, and then store the actual cells colors in the difference between the color and the cell's mean color. A possible refinement consist of predicting the edge points' color not only with the cell's mean color, but with the colors of the adjacent corner points.

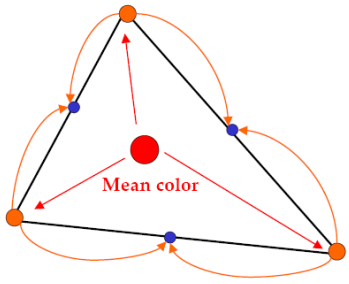


Figure 3.16: Prediction strategy. The arrows symbolize the prediction.

From the decoding angle, the mean colors of the cells is the information that is first needed. Moreover, placing the mean colors sequence at the beginning of the bitstream allows gradual decoding at the decoder side: first the mean colors are decoded and a coarse version of the image can be displayed, then the colors of the corner points can be differentially decoded and the image's preview is refined, finally the edge points are decoded with respect to the corner points' prediction, and the image is completely reconstructed. The mean colors stream itself is differentially coded, as it is done in JPEG for the DC coefficients of the DCT blocks. A way of ordering the cells to ensure maximal correlation between the adjacent mean colors in the stream had to be found.

[14] In JPEG for example, the DC coefficients sequence is likely to be correlated, and the high frequency AC coefficients are likely to form a profitable sequence of zeros.

In order for the bitrate to be reduced, the differential coefficients must be quantized. A uniform quantization schemes was first implemented, but quickly led to disappointing results: Since the colors of a cell are quantized independently from the others, two adjacent cells with close colors before the quantization step might end up being quantized to very different color values. This strongly degrades the rendering quality in uniform area of the images, which happen to be places where the difference coefficients are small. It is the reason why a non-uniform quantizer that emphasizes the precision on small difference coefficients was introduced. Its rate/distortion performances made it the most judicious choice in that case.

Unfortunately, after implementing what intuitively seemed to be a good idea, the obtained compression gain induced by differential coding was not up to the expectations. As it has been said earlier, differential coding is interesting when we have a structure that facilitates the exploitation of redundancy. Our mesh structure does not do that. Image areas with low variations (and thus high redundancy) tend to be covered by large cells and the cells will try their best to capture as much variations as they can, making the difference between the color points as big as possible. This is not compatible with an efficient differential coding scheme.

Color space quantization

Another possibility of bitrate reduction is the straightforward quantization of the color values themselves. A color value is a vector in a three-dimensional color space, so what is meant here is the vector quantization (VQ) of the colors samples from the converted image. Color VQ is used in traditional image representation, in the internet-oriented format GIF in particular: instead of signaling a color by its color triplet (R,G,B) using 24 bits, only 8 bits are used to encode a color index. The color index points an entry in a color palette where the actual true color value is stored. An 8-bit color index bounds the number of color that can be used to 256. This reduction might be transparent for simple synthetic images that uses only a few colors. For natural images where generally numerous color gradations are used, the color VQ is generally a source of strong distortion in spite of the introduction of elaborated palette generation methods and additional tricks like noise dithering. In our context, the situation is somewhat different: all the colors present in a cell are not specified by explicit coefficients. As we have seen in 3.3.2, the color shading of a cell is defined by a set of colors samples. Only the color values of these samples are explicitly coded as a 24-bit true colors, the intermediate color values resulting from the polynomial interpolation between the samples are not. Thus introducing a colorimetric VQ here does not have the same consequences as in the usual raster graphics application: it does not limit the precision on color gradients. The intermediate colors of a cell interpolated from the colors samples eventually go though the whole color spectrum that separates the samples with an infinite precision. The distortion that is created by a color palette quantization concerns the color samples

themselves: we preserve the form of the color gradients but the reference points of these gradients are modified. This visually results in a change of color tone[15] in certain parts of the image.

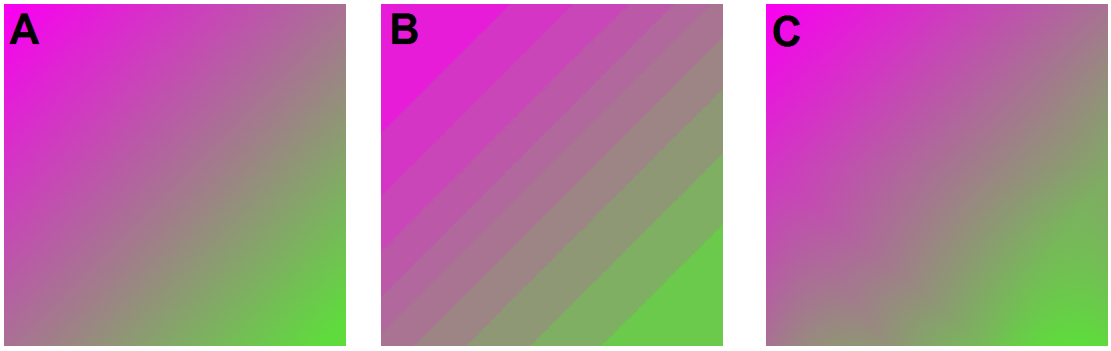


Figure 3.17: Color quantization using a color palette. A: original raster image. B: image coded with a color palette with the classic method. C: *mesh-based* image coded with a color palette.

On the figure 3.17, we see that the precision of the gradient is strongly reduced in B causing important distortion (33.57dB). In C which is coded with our mesh-based scheme, the whole color gradation remains despite the reduction of the color set. C obviously has a better quality both visually and numerically (38.80dB), and a *much* smaller bitrate (for this simple image, the compression factor between C and B is more than 1000 !).

A color shift is, from a numerical point of view, regarded as distortion. Yet, although the general tone of the image is modified, the color gradation coherence tends to be preserved. That is why, in this very context, the colorimetric VQ solution is an interesting one when pursuing additional bitrate reduction. Two important questions arise at this point: How much colors do we keep in the palette, and how are they determined. GIF and other formats historically used 256 color entries for an 8-bit color index. Here we plan on using entropy coding to encode the steam of color indices, so that each of them can be coded on a fractional number of bits. Thus, an arbitrary number of colors can be used. Ideally, we would like this number to be adaptive, which means that the content and the size of the color palette depend of the pictures' content. The color palette design process is driven by a rate/distortion aware criterion, and automatically generates a color palette optimized for the demanded quality. The technical details of the color quantization can be found in 5.2.3.

[15]By color tone, we mean we the hue and the brightness of a color

Figure 3.18: Visual influence of the color quantization. Left: unquantized image. Right: quantized image (245 colors). The visual difference between the two images is very small.

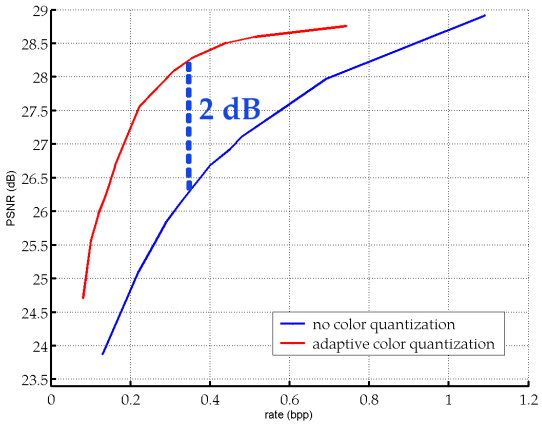


Figure 3.19: Rate/distortion influence of the adaptive color quantization step.

3.3.4 Transmission and decoding.

Transmission / storage

The information that has to be put into a bitstream to make the decoding possible consists of two main parts: the geometrical information and the colorimetric information. The geometrical information consists of the compressed mesh and the discontinuity map[16]. The mesh geometry has already been successfully compressed[17] and only need to be written in the bitstream. The discontinuity map is a binary sequence that can be further compressed with arithmetic coding. These two elements have to be known by the decoder before any image reconstruction can start. Therefore they are placed at the very beginning of the bitstream, and act as an overhead necessary for the actual reconstruction part. What remains of the bitstream is dedicated to the color information. If color quantization is used, the color palette has to be signaled to the decoder. These data are also an overhead to reconstruction. Finally the color indices complete the stream. The overhead we mentioned, consisting of the mesh geometry, the discontinuity map, and the color palette, occupies between 20% and 25% of the final file. While this overhead is processed when decoding an image, no image preview is possible since no actual color information has yet been read, but once the overhead has been receive, does the scheme allow progressive reconstruction ?

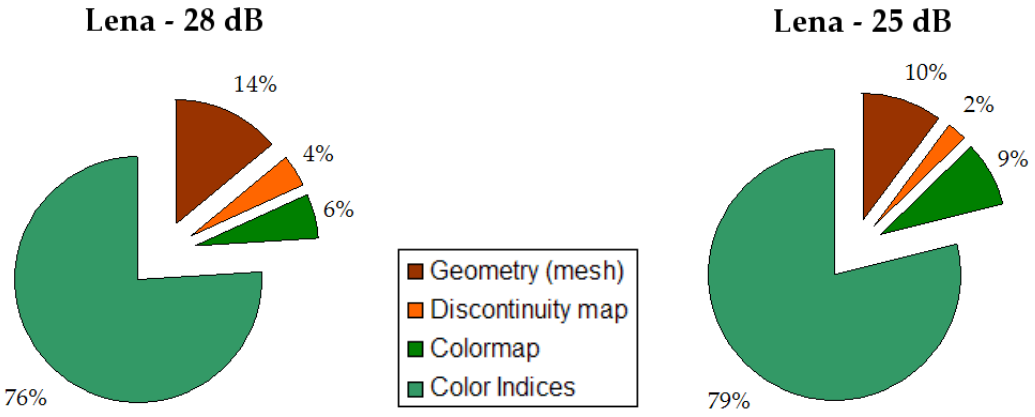


Figure 3.20: Typical image bitstream at two different quality levels. The geometrical information and the color palette form the overhead of the stream. Over the whole possible quality range, the overhead occupies between 20% and 25% of the total filesize.

What is impossible with the our structure is a hierarchical reconstruction. Our mesh-based structure, contrary to [MPL00a], has been kept at a one-level structure in order to allow the maximum cell positioning flexibility. This unfortunately prevents us from

[16]The discontinuity map comes from the selective color merging. see 3.3.2
[17]see 3.3.1

using a multilayer coding and decoding like the one present in JPEG2000. Yet this does not mean that the scheme is completely static, as JPEG is. A careful choice of the sending order of the color coefficients and an adapted interpolation mechanism at the receiver side permit the formation of refining image approximations while the bitstream is being decoded. More details about the sending and reconstruction strategy can be found in 5.2.4.

Decoding and visualization

The decoding of the information introduces no specific difficulty. It basically consists of reading the informations from the bitstream and placing them at the right spots. The geometry and color coefficients stream underwent arithmetic coding before being written in the bitstream, so the first operation performed by the decoder is arithmetic decoding. Then specific reconstruction algorithm are run for the geometrical information and the colors. First in the bitstream comes the geometry overhead. The decoding algorithm of the mesh is very similar to the one present at the encoder side[18], and allows complete and lossless reconstruction of the mesh. The discontinuity map is then decoded and read, and is used as a basis for the progressive color reconstruction described in 5.2.4. The color palette followed by the actual color indices form the rest of the stream. The true 24-bit colors are retrieved through color palette lookups and fed into the progressive reconstruction process.

When the reconstitution of visual object is entirely or partially completed, the visualization can begin. The color shadings inside the cell are interpolated from the color samples and form the visual content of the image or the video. This looks a bit like computing the visual appearance of a triangulated object in a 3D graphics scene. Computer graphics libraries dedicated to this task[19] rely on linear interpolation while we allow the use of higher order terms. To accomplish the higher-order interpolation in the most efficient way, we can directly rely on the programming features of the recent GPUs. A simple fragment shader carries out this task in record time.

For the visualization of video objects, an additional operations helps improving the visual quality of the result: an *artificial motion blur*. The way the tetrahedrization of a video is realized, small tetrahedral cells appears in motion area of the picture. When playing the video, the succession of these cells occurs with a high-frequency which is visually disturbing. A perceptive quality is improved when the cell rendering of these area is averaged over time. This obviously introduces a blur in fast-moving parts of the pictures, but the degradation can be kept reasonably low by using an adapted blurring scheme. The presence of a motion blur is far more natural to the human eye than high-frequency color changes.

[18] see 5.1.1 for more details
[19] OpenGL, Direct3D

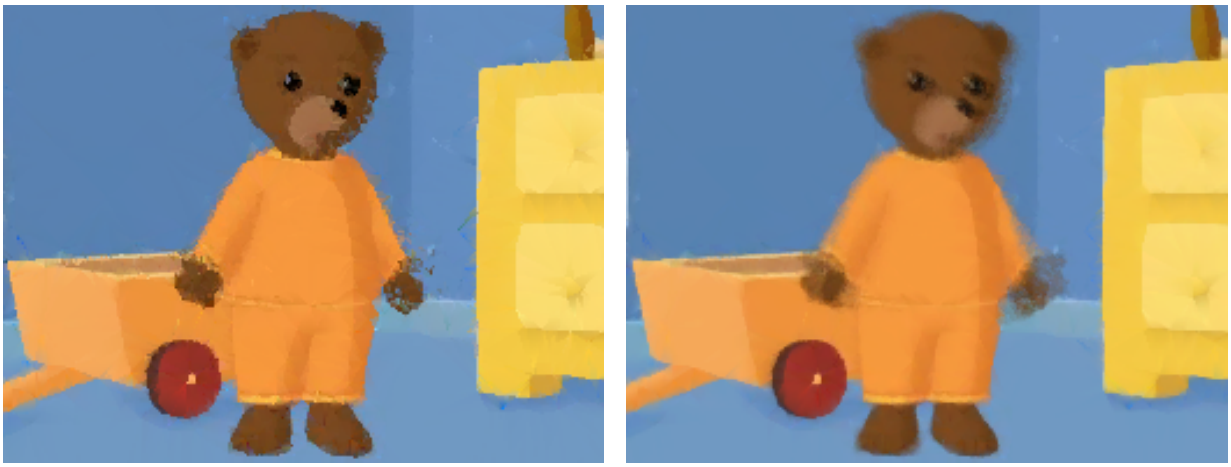


Figure 3.21: Visual influence of the artificial motion blur on videos. Left: no motion blur. Right: motion blur.

As we see, the operations performed at the decoder side are very simple compared to what goes on in the encoder. From a computational point of view, the decoding process is a lot lighter than the encoding. This is something noteworthy in this compression scheme: the computational complexity is concentrated in the encoder.

Chapter 4

The conversion modules

In the previous chapter, a high-level view of the system has been presented. The main steps are described but each one of them was, until now, considered as a black box. In other words, we have been studying the *what*, but not yet the *how*. Since this Diplomarbeit is not only about the theory and the use of abstract methods but also about the development of a test prototype, we will now focus on the lower-level description of its fonctionalities. As we have seen in 3.2, the conversion system can be segmented into blocks of processes. Each of the sections that follow handle one of these processes. They are presented in the execution chronological order.

4.1 Mesh construction

The whole compression process starts with the construction of a two or three dimensional mesh. The mesh is the corner stone of the format's performances, since the final "object" that is encoded is the mesh and its attached informations. The mesh construction takes place at the very beginning of the conversion task and all following processes implicitly depend on the mesh quality. The construction of a good mesh is thus a critical point, and it has been keeping us busy during a fair share of this Diplomarbeit. The mesh constructing entity had to be reviewed several times because it had become the limiting element of the whole system. Thus several version of the mesher have been tested and modified, with the purpose of improving the performances. The first question to consider is how we did evaluate its performances, and what were the important criteria in this matter. Among the main concerns, we can list:

Precision: Since the overall quality depends a lot on the feature of the mesh, it is important to ensure that the iterative construction is meant to converge towards a nearly optimal mesh.

Repeatability: The mesh construction system involves a stochastic operation. Yet it is important that the process remains "stable", in the sense that their must be reproductible for multiple conversion of the same input data, and that small variations in the input data must not change the quality of the final mesh.

Speed: Even if computation speed is not our major concern for the design of this codec, it is important to consider the speed factor for each modules. The convergence can sometimes be accelerated at the expense of a negligible loss in quality.

As presented in 3.2, the mesh construction is driven by a relaxation method. For the quality to improve within each step, we need to provide the relaxation algorithm with an information that will guide the optimization of the positions of the mesh's sites. We understand how important this information is, for it is responsible of the mesh's coherence with the original data, and thus of the overall quality of the final visual object. We call this information the "saliency" information. The computation of this information is the very first first process that is executed in the pipeline. It is presented in the following section.

4.1.1 Saliency computation

The saliency of the original images is computed, and passed on to the relaxation algorithm as a guide to optimize the mesh quality. The name "saliency" is due to the fact that it mainly consists of the edges and discontinuities detected in the input images. Edges and other important informations like rapid movement in a video are features we would like to detect and pack together to signal it to the relaxation algorithm. The latter will then use these so that the vertices of the mesh are attracted to these features in order for them to be well represented. Thus, the respect of the three quality criteria listed in the previous section clearly depend of how well we handle the saliency computation: The precision of the vertices positions obviously depend on the precision of the localization of the saliencies, the repeatability depend on how good the saliency computation resists to changes and noise, and finally the saliency computation is not instantaneous so its execution time has to be taken into account.

The saliency computation like the relaxation algorithm has been subject to several revisions. We will present only the two main methods that have been used and their respective results. The first one is simple and very common, it is the Sobel edge detection method. The Sobel filters are well-known in image processing and frequently integrated in more general processes for the detection of edges in images. It works well enough for high-quality material (high resolution and low noise), nevertheless it reveals its limits when working with noisy images, or containing highly textured elements. In those situation, a finer edge detection scheme is worth being used: the Canny edge detector.

Let us now briefly describe these edge detection algorithms.

Sobel edge detection

the famous Sobel operator along the X axis is:

$$Sx = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

It derives from the continuous first derivative $\frac{\partial}{\partial x}$. The simplest use of this operator is to apply it along the main directions, X and Y for images, thus getting the contribution G_x along X and G_y along Y, and outputting $G = \sqrt{G_x^2 + G_y^2}$ as a result. From a theoretical point of view, this makes sense since G_x is the local derivative of the intensity along X: $\frac{\partial I}{\partial x}$, and G_y represents $\frac{\partial I}{\partial y}$. $\left[G_x, G_y\right]^T$is then the local gradient of the image intensity, and G, our output, is its norm. The same principle is used for the 3D filtering of videos, adding a third component G_z representing $\frac{\partial I}{\partial z}$ in the equations. In that case we have $G = \sqrt{G_x^2 + G_y^2 + G_z^2}$. A generalization of this method also known as compass edge filter extends the computation of the first derivative's norm. Instead of taking the gradient's component only in the direction of the main axis, which are separated by a 90 angle, hybrid components, like the 45 directions, are injected in the result G as well. As far as directions are concerned, it acts like a compass that not only has North, South, East and West marks but also North-East, South-West, etc. Compass filtering improves the detection quality of edge forming an angle with the main axis close to 45 degree at the expense, of course, of execution speed.

A compass-version of the Sobel filter works reasonably good for low noise and non-textured images. However the mesh construction process that will use the saliency information needs precision and robustness, which becomes challenging for more complicated images. The Sobel edge detection did not quite behave as desired in those cases. Another edge detection scheme was thus investigated to improve the quality and reliability of the saliency detection process.

Canny edge detection

The Canny operator was developed later to overcome a few weaknesses of simpler methods like Sobel. The method description that follows is inspired from [Cse], the algorithm itself though has been entirely written for this Diplomarbeit. This method is meant to improve the quality of two main requirements of edge detections schemes:

- **Detection:** multiple-pass processing is used to enhance to detection of real continuous edges, and reduce the contribution of noise and textured areas.
- **Localization:** edges are reduced to a one-pixel wide border.

The 1st pass consists of a low-pass filtering using a Gaussian filter kernel. The goal of this preprocessing filter is to isolate edges as the maxima of the Gaussian smoothing function. Due to the discrete representation of images, edges often forms local chaotic discontinuities which, with a one-pass edge detection method like Sobel, forms a noise-like edge information. The Gaussian filter pass cleans this information before trying to detect edges. The exact position of edges are then precisely localized where the maxima of the smoothed edge intensity lies.

The 2nd pass is the edge detection pass itself. The purpose here is to detect the presence of edges according to a usual criterion but also to detect the *orientation* of the edges at the positions where they are detected. This is usually done by measuring the intensity variations in the two (or three) principal direction X and Y (and Z) from which both the presence of an edge and its orientation can be deduced. The Sobel filter for example is well adapted for this task. The directional gradient contribution G_x, G_y and G_z are extracted, the edge strength is related to the norm of the gradient $G = \sqrt{G_x^2 + G_y^2 + G_z^2}$, and the orientation of the edge is deduced from the ratios between G_x,G_y and G_z, and stored as angle information (θ and φ for 3D video volumes, θ only for 2D still images). At the end of this pass we known the strength of the discontinuity at each pixel (or voxel) and its orientation. Note that for the 2D case, the edge orientation is given by a one-dimensional line whereas in 3D, since the edges are surfaces, it is given by a 2D plane.

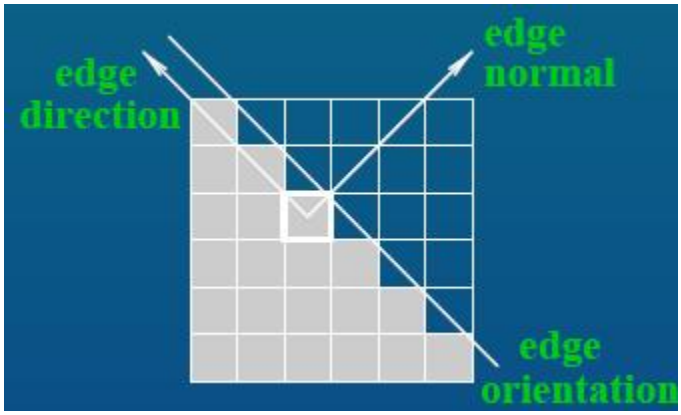


Figure 4.1: The edge orientation and the the edge normal.

The 3rd pass is called non-maximum suppression and aims at reducing the width of the edges to one pixel. Indeed, a discontinuity in a digital image is rarely infinitely thin and perfectly localized at a given pixel, but often spread over a few pixels perpendicularly

to the edge's direction . Adjusting the threshold of a classic edge detection scheme like Sobel will mostly lead to either detecting all this group of pixels, or detecting none of them, which means we either detect wide, not precisely localized edges, or we simply miss them. The non-maximum suppression is here to select from the group of pixels the one that represent the most exact position of the edge and only this one, thus marking one-pixel-wide edges. To do this, for each edge detected in the second pass, the orthogonal direction (along the normal of the edge) to the edge is determined, based on the measured edge orientation. Pixels along this normal direction are compared with one another, and only the maximum is marked as an edge pixel.

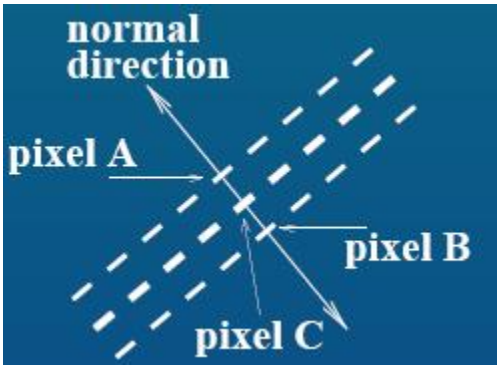


Figure 4.2: Illustration to the non-maximum suppression. Given that $G(C) > G(A)$ and $G(C) > G(B)$, pixels A and B are suppressed and only the pixel C is kept as an edge pixel.

The 4th and final pass is called "hysteresis thresholding". It is meant to remove isolated maxima, and thus indirectly emphasizing discontinuity that belong to a real continuous contour in the image. This pass is eventually the one claiming the longest execution time. Remark that it is not mandatory but it efficiently removes edges due to strong noise and textured areas. It is based on the research of the connectivity between the detected maxima, and parametrized by two thresholds T_{low} and T_{high}. Each remaining maxima G from the 3rd pass goes through the following process:

- If $G < T_{low}$, G is suppressed, and is called a "weak edge".
- If $G > T_{high}$, G remains a maximum, and is called a "strong edge".
- If $G \in [T_{low}, T_{high}]$, G is a candidate. In this case, we follow the chain of maxima along the edge direction, so long they are greater than T_{low}, and if one of them is greater than T_{high}, meaning that the candidate is connected to a strong edge, the candidate is kept as a maximum. Otherwise, the candidate is suppressed.

Strong edges and successful candidates are the final output of the canny edge detector. The hysteresis thresholding does a good job isolating actual contours from fake edges, but is a time-consuming task, particularly in the 3D case where the connectivity search is a 2D search.

Ideally, after all these steps, only intensity variations belonging to real objects' contours are output as detected edges. Let us visually compare the two implemented scheme Sobel and Canny:

Figure 4.3: The edges detected with the Canny approach benefit from a better localization and continuity compared to Sobel, at the expense of more complex and time-consuming processing.

4.1.2 Relaxation

The relaxation scheme is the process that will actually place the vertices and set the connections between them to form the final mesh. The saliency information (see previous section) will be its principal interaction with the source data. As explained in 3.2, the saliency information will serve to attract the mesh's vertices on the discontinuities on the image, thus preventing these discontinuities from appearing in the middle of the mesh's cells. The strategy we use to generate the mesh as also already been exposed: Out of the two degrees of freedom at disposal, the position of the vertices and the connections, we will control only the first one, leaving the handling of the connections to a Delaunay algorithm.

Vertices positions

Let us assume we dispose of V vertices to scatter in the object space. The object space consists of a 2D surface for the image codec, and of a 3D volume for the video codec. The saliency computation step has, at this point, provided the discontinuity detection results as a pixel-based function in the object space (pixel grid for images, and voxel grid for volumes). Like often in tasks where a small number of sites have to be placed intelligently in a larger domain, a stochastic scheme based on relaxation will be used to overcome the NP-hard theoretical complexity. The vertices will be placed at initial

positions and then iteratively moved until a satisfying result is obtained. There exists several types of relaxation depending on how the position of the vertices is updated within each round. They all have the same objective but differ in terms of characteristics like the execution speed of each round, the quality and speed of convergence and the complexity. The most relevant ones were tested for our application, and the one presenting the most attractive compromise is the Centroidal Voronoi Tessellation (CVT). In theory, it is very similar to a Lloyd relaxation [Llo82], since it is based on the Voronoi[1] partition of the object's space.

Let us go through one iteration and list the operations involved in a point position update.

At iteration k:

- 1: A Delaunay mesh is constructed out of the current positions of the vertices.
- 2: From the Delaunay mesh, we extract the Voronoi diagram of the vertices. We use the fact that the vertices of the Voronoi cells are the center of the circumspheres of the Delaunay cells.
- 3: There is one and only one vertex per Voronoi cell. Each vertex will be updated independently in its cell. The new positions of the vertices will be the centroid of their respective Voronoi cell. The centroid c_V of the cell V is computed according to a weighting function which is the saliency information $s(v)$ with the following formula:

$$c_V = \frac{\sum_{v \in V} s(v).v}{\sum_{v \in V} s(v)}$$

- 4: Go back to 1, with the new positions of the vertices.

The advantage of the centroidal Voronoi tessellation is its good convergence quality. Despite rather computationally intensive iteration steps, the vertices quickly converge towards a satisfying solution.

Initially the points are placed randomly throughout the object's domain. The first CVT iterations quickly correct the position of the vertices, whereas in later iteration steps the displacement tends to decrease rapidly, making the actual convergence an asymptotic

[1]The Voronoi diagram of a set of sites is the partition of a space into cells centered on the points (one cell per site). The main property dictating the construction of the cells is that for all the points inside a Voronoi cell, the closest site to these points is the one attached to the cell they belong to. Thus the frontiers of the Voronoi diagram consists of points that are equidistant to two sites (or more).

The Delaunay and Voronoi diagrams are called "duals" because their features are theoretically related. Indeed the connection in a Delaunay mesh occur between sites whose Voronoi cells are adjacent in the Voronoi diagram. Moreover, the intersections between the frontiers of the Voronoi diagram are the centers of the circumspheres of the Delaunay cells. Thus, because of this theoretical relation, conversions in one sense or the other can be easily performed.

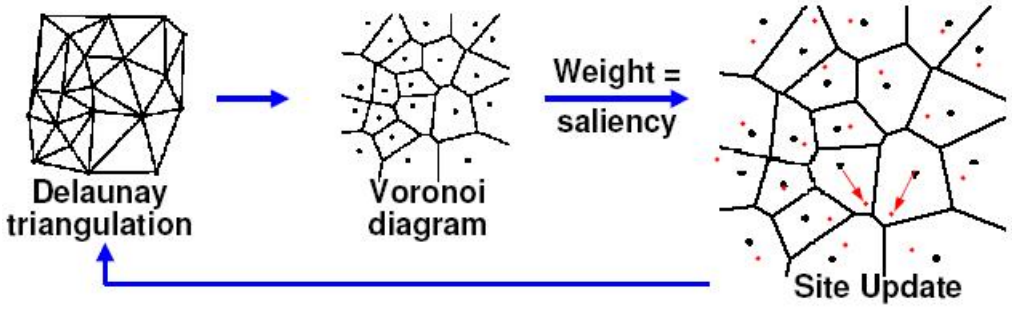


Figure 4.4: The centroidal Voronoi tessellation process. Each operation is described above the illustration

situation. However, after a while, the remaining movements become negligible. Generally, for images and videos with regular dimensions, the mesh has reached a satisfying form after a dozen of iterations.

Another approach that can be substituted to the CVT is what is called the Optimal Delaunay Triangulation. The ODT is introduced in [CX04] and was used in [ACSYD05] from which we partially derived our 3D mesh construction scheme. This relaxation can be seen as a simplification of the the CVT. It is simpler because no computation of the Voronoi diagram is required, the new positions of the sites are computed directly in the Delaunay mesh. For each vertex, we list all the Delaunay cells attached to it and retrieve the center of their circumspheres. The updated position of the vertex is issued as center of mass of these centers. Knowing that the circumsphere centers are actually the corners of the dual Voronoi cells, we understand how this approach can be considered as a simplification of a CVT: instead of integrating the saliency over a whole Voronoi cell to compute the centroid, we approximate it using only the corners of the cell. Clearly the ODT has the execution speed advantage over the CVT since less information is to be extracted per vertex within iteration. Despite a proved theoretical convergence [CX04], we have observed in the practical application that the sites tend to oscillate strongly when being iteratively relaxed, leading to a lack of accuracy compared to a CVT with the same number of iterations.

According to our observations on the geometrization of images and videos, for the same execution time the results obtained with a CVT are generally more reliable. Thus the CVT was chosen, for both the 2D and the 3D case, to be the algorithm used for the placement of the mesh's vertices.

Connections

Once the desired amount of iterations has been executed, the vertices have reached their final position. The connections between them are then established according to a Delaunay scheme, exactly as in the first step of the relaxation sequence mentioned above. The average complexity of a 2D Delaunay triangulation or a 3D tetrahedrization is $n.log(n)$.

4.2 Color fitting

4.2.1 Principle

Once the mesh has been constructed, the image sequence is no pixel grid anymore, but a geometrical object: A tetrahedral or a triangulated mesh. However, the essential part of what makes images and videos what they are is still missing: the colors. As explained in 3.1.2, each cell will be assigned a vector in the chosen function space that best represents the color information of the pixels it contains. This operation is called "color fitting" because it is nothing but the numerical fitting of a set of parameters to minimize a given cost function.

The three color channel R, G and B are processed separately according to the same scheme. What we are about to describe in this section concerns the processing of a one-dimensional intensity function. It will practically be applied to any of the three color components. Moreover, to avoid repeating the same things for both cases, the following section will describe the color fitting of video objects, and thus use the adapted vocabulary (like "tetrahedron" and "3D mesh"). Everything is of course equivalent for 2D image objects.

As mentioned in 3.1.3, The tetrahedrons occupy space in the volume formed by the image sequence, and therefore they "contain" a group of voxels from the grid. Everything has been done so that the intensity variations carried by the voxels inside each tetrahedron remain as simple as possible, so that they could be modeled by a small set of parameters. These parameters are the weighting factors of chosen trivariate functions that define the basis of the chosen function space. The evaluation of the functions at a given (x, y, z) point, weighted by the fitted coefficients for this tetrahedron will give the new intensity value at (x, y, z).

Let us now focus on the method used to determine the α_i coefficients. The quality factor we have to consider here is the visual result, its closeness to reality and the presence of color artifacts. Let α_i, $i = 1..n$ be the fitted coefficients for this tetrahedron, ϕ_i the basis functions and (k_j, l_j, m_j) the appropriate voxel coordinates. The chosen mathematical model is the following:

$$\begin{cases} \alpha_1.\phi_1(k_1,l_1,m_1) + \alpha_2.\phi_2(k_1,l_1,m_1) + ... + \alpha_n.\phi_n(k_1,l_1,m_1) \approx I(k_1,l_1,m_1) \\ \alpha_1.\phi_1(k_2,l_2,m_2) + \alpha_2.\phi_2(k_2,l_2,m_2) + ... + \alpha_n.\phi_n(k_2,l_2,m_2) \approx I(k_2,l_2,m_2) \\ ... \\ \alpha_1.\phi_1(k_K,l_K,m_K) + \alpha_2.\phi_2(k_K,l_K,m_K) + ... + \alpha_n.\phi_n(k_K,l_K,m_K) \approx I(k_K,l_K,m_K) \end{cases}$$

The approximation above becomes an equality only if the original picture intensities obey to a polynomial law. This is never the case in practice. The purpose of the fitting is that the left member of the equation above is as close as possible to the right member. We can isolate the unknowns with the matrix form of the equation:

$$\begin{bmatrix} \phi_1(k_1,l_1,m_1) & \phi_2(k_1,l_1,m_1) & ... & \phi_n(k_1,l_1,m_1) \\ \phi_1(k_2,l_2,m_2) & \phi_2(k_2,l_2,m_2) & ... & \phi_n(k_2,l_2,m_2) \\ ... & & ... & \\ \phi_1(k_K,l_K,m_K) & \phi_2(k_K,l_K,m_K) & ... & \phi_n(k_K,l_K,m_K) \end{bmatrix} . \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ ... \\ \alpha_n \end{bmatrix} \approx \begin{bmatrix} I(k_1,l_1,m_1) \\ I(k_2,l_2,m_2) \\ ... \\ I(k_K,l_K,m_K) \end{bmatrix}$$

This is a $A.\overrightarrow{x} = \overrightarrow{b}$ type equation where $\overrightarrow{x}$ is unknown and symbolizes the coefficients α_i, A is a Kxn matrix composed of the basis function evaluated at the voxel positions, and $\overrightarrow{b}$ being the vector of the original intensities at these positions. This problem is very well known in the theory of optimization problems. We will make only a brief reminder of the possible cases:

- $n > K$, A has more columns than lines. It is an under-determined optimization problem (more unknown than equations). In this case the equation above has an infinite number of solutions. An additional criterion is often added to specify a single one of them, for example the selection of the solution that has the smallest norm ("min norm solution"). In our case however, this is not a satisfying solution. The degrees of freedom left for the choice of the α coefficients might lead to oscillations phenomena outside of the integer voxel position. When this case is encountered, the fitting degree n is decreased and the system is solved with a smaller function basis.
- $n = K$, A is square. There is as many equations than unknowns. If $\det(A) \neq 0$, there exists a unique solution $\overrightarrow{x} = A^{-1}.\overrightarrow{b}$.
- $n < K$, A has more lines than columns, the system has more equations than unknowns (this is the most common case). Generally speaking, there exists no solution that verify all equations. The purpose here is to determine the unknown such that $A.\overrightarrow{x}$ is as close to $\overrightarrow{b}$ as possible. The most famous method to solve this is the Least Squares approximation. This method and its variations were used in this context, as described below.

4.2.2 Least Squares fitting

We aim at solving $A.\overrightarrow{x} = \overrightarrow{b}$, where A describes the space spanned by the function basis (ϕ_i) and $\overrightarrow{b}$ the actual intensities of the original images. As we said in 3.1.2, we want to project the real intensities on the function space, and that is exactly what the Least Squares method does. Indeed a Least Squares solution $\overrightarrow{x_{LS}}$ is the orthogonal projection of $\overrightarrow{b}$ onto the space spanned by the vectors of A ($A.\overrightarrow{x_{LS}} = \overrightarrow{\hat{b}}$, $\overrightarrow{\hat{b}} \in im(A)$, and $\overrightarrow{\hat{b}} \perp (\overrightarrow{b} - \overrightarrow{\hat{b}})$). A nice property deriving from this theoretical point of view is that the square error between the data $\overrightarrow{b}$ and the approximation $A.\overrightarrow{x_{LS}}$ is minimized. The Least squares solution is often describe with the following formula: $\overrightarrow{x_{LS}} = \arg\min_{\overrightarrow{x}}(\left\| \overrightarrow{b} - A.\overrightarrow{x_{LS}} \right\|^2)$.

The Least Squares solution is obtained using he following pseudo-inverse formula:[2]

$$\overrightarrow{x_{LS}} = (A^T A)^{-1} A^T . \overrightarrow{b}$$

In our case, the unknown symbolized by the vector $\overrightarrow{x}$ contain the weighting coefficients α_i. The weights α_i^{LS} obtained from the Least Squares solution are the result of the orthogonal projection of the intensities onto our function basis.

The Least Square method minimizes the square error between the reality and the model, but in its most simple form it does not take the validity of the input data into account. There are cases where the the input data (here the vector $\overrightarrow{b}$) are subject to errors, or more precisely what we call "outliers". We define an outlier here as a voxel that is included in the current color fitting, but whose intensity clearly does not fit with what is to be modeled in the current cell. It can be a dead pixel for instance, a strong localized noise or more commonly a voxel that was not supposed the be included in the current color model. If for some reason the vector $\overrightarrow{b}$ carries this kind of distorted information, the influence on the result of the Least Squares fitting is significant. It is said that the LS method has a worst-case breakdown point of 0%[3]. An example of outliers' influence in a linear fitting operation is presented in figure 4.5.

In our case, most outliers come from the fact that edges in a pixel-based image are not infinitely thin, whereas triangle's and tetrahedrons' edges are. When the color fitting process start, the cells register all the pixel they contain in their $\overrightarrow{b}$ vector. The pixels representing an edge itself are bound to be included in either one or the other cell spreading on each side of the edge. This has consequences in the final visual result,

[2]This is only valid if the matrix A has full rank ($rank(A) = n$). Otherwise the generalized pseudo-inverse (based on the singular value decomposition) has to be used.

[3]The breakdown point is the proportion of outliers needed to influence the final model. A 0% breakdown point means that a single outlier changes the final result.

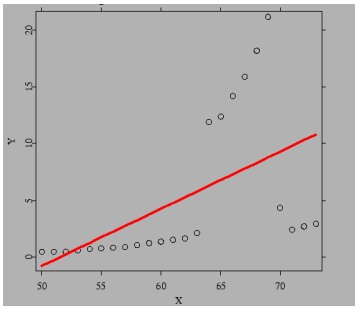


Figure 4.5: Illustration of the lack of robustness of the classic Least Squares fitting method. Example of a linear fitting. By trying to minimize the squared error, the line fit (in red) is strongly affected by the 6 outliers at the top of the figure (source *www.xplore-stat.de*).

similarly to the example displayed in Figure 4.5: if we imagine on one side of a yellow edge an area where the voxels are uniformly blue, the cell supposed to represent this blue area may also includes a few yellow voxels belonging to the edge. A linear fitting process will acknowledge this as a shaded area ranging from yellow to blue, instead of the desired uniform blue. The yellow voxels are in this case outliers for the white cell in the sense that they do not belong to the phenomenon we want to model.

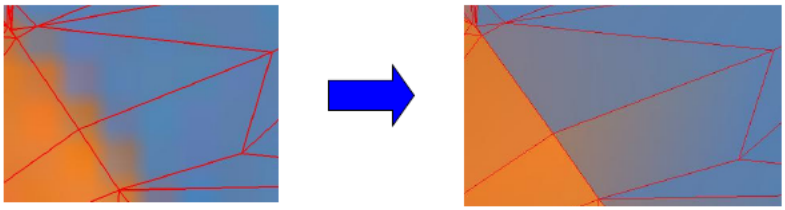


Figure 4.6: Influence of outliers on the fitting quality. The edge pixels force the model of the triangle to a progressive shading whereas a constant color would be visually closer to the reality.

The visual impact on the result is important. To reduce this distortion, we chose to use a more evolved fitting scheme able to minimize the influence of outliers. The scheme is presented in the following section.

4.2.3 Robust fitting

The purpose of the robust fitting scheme is to fit a mathematical model to original data in a more regarding way than a simple Least Squares method, specially towards outliers. A simple and intuitive example of robust estimation is to use the median instead of the mean-value in the case of a degree 0 approximation. If a few data are subject to

strong distortion, this will distort the mean value but only slightly affect the median. Robust schemes for higher order fittings also exists, we will present one of them in this section. Note that this interesting approach comes at the expense of a higher theoretical and computational complexity. Alone the robust constant fitting described above, which requires the use of the median instead of the mean value, uses sorting algorithms instead of a few arithmetical operations. The complexity gap between the Least Squares fitting and the robust one increases with the fitting degree. The scheme that we will use to estimate the coefficient in this section is called "M-estimators". The well-known cost function of a normal Least Squares fitting we look to minimize is:

$$F_{LS}(\overrightarrow{x}) = \left\| \overrightarrow{b} - A.\overrightarrow{x} \right\|^2$$

Similarly, the M-estimators tend to minimize the following cost function (Lp norm):

$$F_M(\overrightarrow{x}) = \rho(b - A.\overrightarrow{x}) = \frac{\left\| \overrightarrow{b} - A.\overrightarrow{x} \right\|^\nu}{\nu}$$

Note that if $\nu = 2$, the M-estimator scheme degenerates to a Least Squares optimization problem. Obviously the higher ν is, the more importance we give to outlying contributions. On the other hand, if ν is too small, the fitting will tend to erase meaningful variations in the cells. The best results were obtained with $\nu = 1.2$. For the Least Squares problem, we are able to derive a direct solution using the pseudo-inverse, but unfortunately, no such method exists for M-estimators. An iterative approach has to be used: the Iteratively Reweighted Least Squares (IRLS) method. This process consists of successive Weighted Least Squares resolutions where the weights are accordingly reevaluated and reassigned at each step. This method is actually a generalization of Newton's method for non-linear optimization problems. It has been proved to converge towards the minimum of the cost function F_M ([Zha96]).

The method's principle is illustrated in figure 4.7.

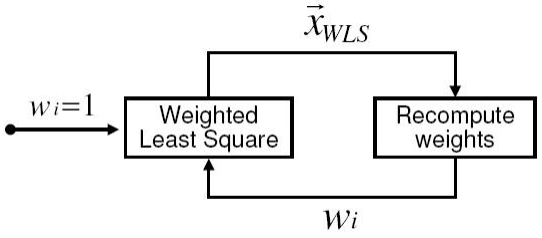


Figure 4.7: Iterative Reweighted Least Squares optimization. $\overrightarrow{x}$ represents the solution of the WLS step, and w_i, $i = 1..n$, are the weights which are computed in the second step.

As we can see, the process consists of subprocesses executed successively. For each cell of the mesh, the following scheme is repeated. The starting point of the loop is a Weighted Least Squares fitting where all weight w_i are initialized to 1, which is nothing but a regular LS fitting. The solution noted $\overrightarrow{x_{WLS}}$ is passed on to the second subprocess on the right side of the graphic. Using the solution $\overrightarrow{x_{WLS}}$, this subprocess evaluates the fitting error r_i at each voxel i between the original pictures and the current model. New weights w_i ($i = 1..n$) are computed in accordance to a particular measure $\omega(r_i)$ of the fitting error, and passed on to the next WLS operation. It has been proved that with the right ω function the process converges towards the minimum of the cost function (ω obviously derives from ρ, i.e. from the cost function).

In the case of the Lp norm, the ω function is given by $\omega(r_i) = |r_i|^{2-\nu}$. The new weight allocated to the corresponding voxel is the inverse of ω,so the weight actualization step of the IRLS process in the case of M-estimators uses the following formula: $w_i = |r_i|^{\nu-2}$.

As we have seen using M-estimators increases the theoretical complexity, but it also increases the computational complexity a great deal, since we now need to perform several LS optimizations instead of one. Five to six iteration are generally necessary to achieve meaningful results, which means that the computation time for the color fitting operation is multiplied by approximately six.

M-estimation is not the only robust fitting method that exists to reduce the influence of outliers. With M-estimators, the contribution of outliers is reduced, but not removed: the theoretical worst-case breakdown point is still 0% like with LS, only the magnitude of outliers' influence is limited. There exists method that truly improve the breakdown point, which means that the contribution of outliers is completely removed. A method known for its robustness is the Least Median of Squares (LMS) method[4]. The cost function to minimize is: $F_M(\overrightarrow{x}) = med(\overrightarrow{b} - A.\overrightarrow{x})$, with $med(\overrightarrow{v})$ is the median value of the coefficient of $\overrightarrow{v}$. The method's breakdown point can go up to of 50%. The practical realization of the principle is however complex and extremely time-consuming. Another robust method with a similar approach based on stochastic processes was implemented and integrated: the RANSAC method[4]. RANSAC estimators are robust fitting schemes commonly used in computer vision tasks[5]. The principle of RANSAC is to extract partial subsets out of the complete data we normally use for a fitting, and perform a fitting on just this subset. The validity of the partial model is then checked on the whole data set and a "grade" is given to the model. The experience is repeated a given number of times and, in the end, the model with the best grade is picked[6]. The choice of the parameters of the scheme was, as it is for plenty of algorithms, a complicated issue. Indeed, parameters like the proportional subset size, the number of rounds and the model grading mode all influence the final result. In our context we have found the ad-

[4][Zha96]

[5]Its main application is the estimation of fundamental matrices in multivision applications.

[6]A more detailed description can be found at *http://en.wikipedia.org/wiki/RANSAC*

justment of these parameters particularly difficult since a lot of different situations are encountered among the cells to be fitted. Adapting the RANSAC parameters to cope with one type of problem generally let the other ones unsolved or even worsened. An intuitive analyze for this is that small subsets are more likely to get rid of important outlier's contribution, but result in a poor stability (two successive fitting operations deliver very different results). Taking bigger subsets surely improve the stability but the system is then unable to solve certain fitting error. All this depends obviously a lot on the image itself and its intensity range. Unfortunately, no way of efficiently using the RANSAC algorithm to cope with outliers has been found. The M-estimators, although not always optimal, handle fitting issues in a more flexible way among the different cells and different picture types.

Chapter 5

The compression modules

5.1 Geometry compression

5.1.1 The compression algorithm

As it has been said multiple times in chapter 3, only the position of the vertices of the object's mesh need to be signaled to a decoder for it to be correctly reconstructed. The vertex set will be compressed with the algorithm described in [DG00]. In this section we give an overview of this method for the 2D case. It can be very easily generalized to any dimension.

Let S be a set of n points all contained in a square bounding box S_0. The spatial dimension of the box are $2^b \times 2^b$ and all the points have integer coordinates. In its raw form, a point's coordinates are then coded on $2b$ bits. At the very beginning of the process, the total number of points n is encoded in the bitstream on a arbitrary fixed number of bits (32 for example), and the entire space S is placed in a FIFO. The main loop of the program then begins. Each iteration of the loop consists of subdividing the current space into two parts, and encoding the number of points contained in one of them on the optimal number of bits[1]. If the two resulting subspaces are not reduced to a singleton, they are placed at the back of the FIFO. The subspace that is in the front of the FIFO is then extracted and the process is repeated with it. Here is a pseudo-code version of the algorithm

$F \leftarrow$`space bounding box` S_0
$n_0 \leftarrow$`total number of points in` S_0
`write` n_0 `in` B `on 32 bits.`
`while` F `is not empty`

[1] If the current space, before the subdivision, contains p points then the number of points in one of the subspaces will be coded on $\lceil log_2(p+1) \rceil$

```
        S ←pop front space in F
        n ←number of points in S
        subdivide S in half
        S1 ←first half of S
        S2 ←second half of S
        n1 ←number of points in S1
        write n1 in B on ⌈log2(n+1)⌉ bits.[2]
        if n1 > 0 and S1 is not a singleton
            push S1 at the back of F
        endif
        if n2 > 0 and S2 is not a singleton
            push S2 at the back of F
        endif
    endwhile
```

For the subdivision part, the space is divide into 2 equal parts along one dimension. the next subdivision of one of the 2 obtained subspaces will be done along the other dimension. Let us say for instance that the original space S_0 is cut in half with respect to the X axis, the resulting subspaces $S_1$and S_2will have to be cut with respect to the Y axis in the following round. For n dimensional cases with n$>$ 2, the principle is the same each part of space will be segmented successively along each dimension.

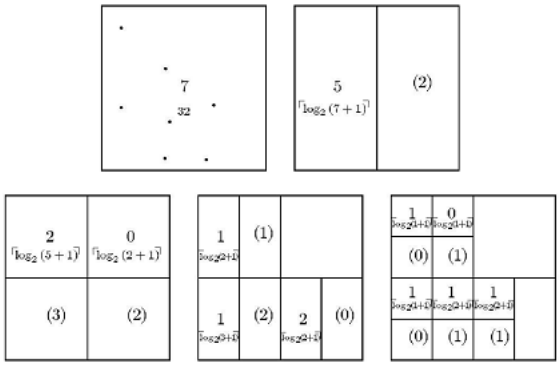


Figure 5.1: Vertex set compression mechanism: the successive space subdivision and transmitted symbols on a two-dimensional example.

Looking at the content of the bitstream, one can note that the only informations it contains are the numbers of points contained in the subspaces. The successive divisions of space form a binary tree structure, and these divisions are executed until the subspaces containing a point are reduced to a singleton. At the decoder side, the same subdivision sequence is run, so that the numbers extracted from the stream are assigned to

[2] $n_2$does not need to be stored because it can be deduced from n and n_1: $n_2 = n - n_1$

their respective subspaces. Obviously when reading a number n_1^d from the bitstream the decoder knows how much bits have been used to encode it since the population of the parent subspace n^d has been decoded earlier, so n_1^d is coded on $\lceil log_2(n^d + 1) \rceil$bits. Like in the encoder, the decoder divides space until the singleton size is reached. The decoder can then trustfully place a vertex at the position of the singleton. That way the positions of the vertices are transmitted without storing a single coordinate.

Compression is achieved because the data are structured in a way that requires no specific signaling for reconstruction and that allow the symbols to be transmitted with an adapted number of bits. The compression performances performances can be further improve if we solve the problem that the numbers are coded on an integer number of bits. Using joint entropy coding makes it possible to encode a symbol on a fractional number of bits. If arithmetic coding is used, even symbols belonging to a binary alphabet can benefit from an adapted fractional number of bits. Arithmetic joint coding tends to reduce the bit length of a codeword from $\lceil log_2(n + 1) \rceil$ to $log_2(n + 1) + \epsilon$, where ϵ is small and decreases as the length of the symbol stream increases).

Arithmetic coding has an additional advantage that makes it the most adapted entropy coder to combine with the present algorithm: the symbols, although encoded jointly, can be progressively transmitted and decoded[3]. The geometry coding algorithm has a top-down approach. The first symbols that have to be transmitted are independent from the symbols that come afterward. Indeed when a subspace S contains n vertices, n has to be transmitted independently from how the n points are scattered in the further subdivisions of S. Equivalently, the vertex set decoding can begin even if the bitstream has not been completely recovered: every bit coming in refines the precision on the position of the vertices, and once the last bit has been decoded, the set has been perfectly reconstructed.

The compression performances of the Devillers and Gandoin codec are presented in table 3.1. The compression ratio is satisfying and the reconstruction contains no error. With the reconstructed vertex set, the mesh is recovered after a Delaunay triangulation of the set. Most of the time, the point set is well-conditioned and the Delaunay triangulation is unique but degenerate cases can happen (see next section). Fortunately, if every precaution is taken, the problem disappears.

5.1.2 A reconstruction issue: cocyclic vertices

In our context (vertex sets that contain more than one point), the Delaunay triangulation is "almost" unique for a given set of points. The Delaunay property ensures that

[3]In Huffman coding, joint symbols are harder to handle. They require large coding tables, and are non-progressive. The Huffman encoder can issue a code word only when all the joint symbols are known, and equivalently the decoder can begin decoding a joint symbol only after its entire code word has been received.

the circumcircle passing through the corner points of a cell contain no other points. A problem arises when more than 3 points in the set (more than 4 in the 3D case) happen to be cocyclic. A subset of points are said to be cocyclic when they all belong to a common circle (a common sphere in the 3D case). When it comes to the Delaunay triangulation, there might be ambiguous case like the one on figure 5.2.

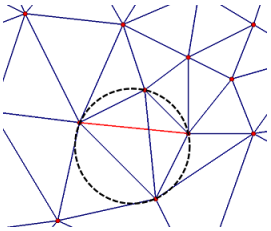


Figure 5.2: Ambiguous case of Delaunay triangulation due to cocyclic points. Both of the possibilities displayed for the edge (blue or red) are valid for a Delaunay triangulation.

It takes an additional vertex ordering procedure to take away the risk of ambiguous Delaunay reconstruction. The procedure has to be executed at both encoder and decoder side to ensure that the exact same deterministic Delaunay construction is performed at both places.

5.2 Color compression

5.2.1 From polynomials to color interpolation points

In our image and video format, the color variations of a cell are described by three multivariate polynomial functions, one for each color channel. Mathematically speaking, the intensity of a color channel C at a given point $p = (x, y, z)$ is given by $I_C(x, y, z) = \alpha_0 + \alpha_1.x + \alpha_2.y + \alpha_3.z + ... + \alpha_{n-1}.x^i.y^j.z^k$, with $i + j + k \leq d$, where d denotes the degree of multivariate polynomials. The color fitting operation find the polynomial that best approximates the original color variations in the cell and delivers the corresponding alpha coefficient set $(\alpha_0, ..., \alpha_{n-1})$. From studies on topic interpolation we know that for particular number n of points p_k and their associated values $I_C(p_k)$, there exists only one d^{th}-degree polynomial $P_C(p)$ for which $P_C(p_k) = I_C(p_k)$. Thus, the polynomial representing a color channel in a given cell can be represented either by its set of alpha coefficients or by a set of pairs $(p_k, I_C(p_k))$. If we impose a particular sampling dictated by the geometry of the cell, the positions p_k are known as soon as the mesh is known: the set $(I_c(p_k))$ is then sufficient to represent the polynomial. For both representations, n values are required. For a given d, and if s is the

dimension of the source data[4], n is given by $n = \begin{pmatrix} s+d \\ d \end{pmatrix} = \frac{(s+d)!}{s!d!}$. A 2D image approximated by second-degree polynomials for instance needs 6 coefficients per channel and per cell. In that case, the chosen sample strategy is displayed on figure 3.13. The chosen interpolation points are obviously the same for the three color channels, so after the representation change, we are left with n interpolation points per cell, each of them carrying three color coefficients (a color vector).

We can note that in the image conversion process, we will use the shift of representation in only one way: from alpha coefficients to interpolation points. In that direction, the conversion is particularly easy since we only need to evaluate the polynomial functions at the positions of the interpolation points. Computationally speaking, the cost of the conversion is negligible. The conversion from interpolation points to alpha coefficients is somewhat more complex since it require the inversion of a $n \times n$ matrix per color channel and per cell.

5.2.2 Selective color merging

Whether relying on alpha coefficients or color sample points, the color shading of a cell is determined independently from the surrounding cells[5]. Thus the color continuity between cells is not ensured by construction. This is profitable when it comes to representing strong discontinuities between cells, but in low varying areas of an image it might lead to unwanted color gaps at the common edge of the cells. If the color fitting is done correctly, the gap is not important, but when color quantization[6] adds up, the gaps are widened causing important visual distortion. The selective color merging is meant to eliminate the gaps in area where the color varies smoothly. To do so, the chosen algorithm first flags the edge of the mesh as continuous or discontinuous, then based on this knowledge, the corner points and the edge points are averaged over adjacent cells or not. To state that an edge is continuous or not, the color distribution of the original image at the edge's location is considered. A polynomial function based on the color information from both sides of the edge is fitted, and compared with the already fitted polynomial functions on both sides of the edge. If the difference between the separate fitting curves and the unified fitting curve is small, the edge is flagged as continuous. Otherwise it is flagged as discontinuous.

The continuous/discontinuous flag is a binary attribute that has to be stored for each edge. We call this information the *discontinuity map*. For the lena picture, 40% to 50% of the edges are flagged continuous. The actual color merging relies entirely on the discontinuity map. The colors of two coincident edge points are merged if the edge

[4] $s = 2$ for images and $s = 3$ for video volumes

[5] see the color fitting process in 4.2

[6] see 3.3.3

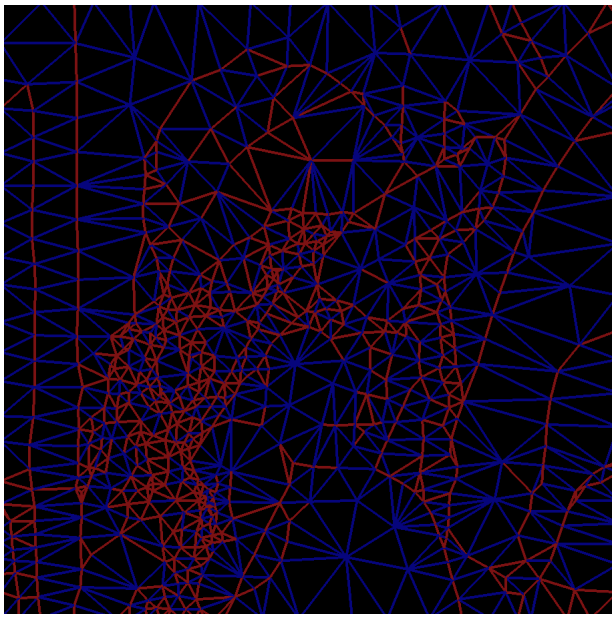


Figure 5.3: The discontinuity map of the lena picture. Blue edges are flagged continuous and red ones are flagged discontinuous.

is flagged continuous. For corner points, the merging mechanism is somewhat finer: The color of the corner points are merged between cells that are not separated by a discontinuous edge. This is illustrated in figure 5.4. When a group k colors is merged, the colors are replaced by the mean color of the group.

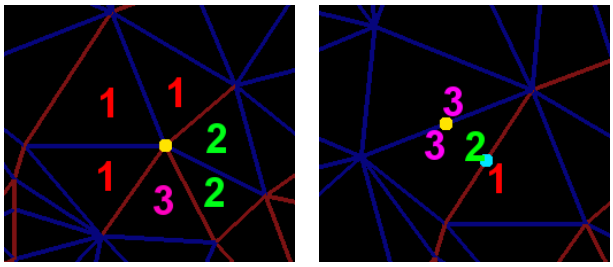


Figure 5.4: Color merging conditions and mechanisms. Left: merging of corner colors. Right: merging of edge colors. The groups are spatially segmented by discontinuous (red) edges. Color groups are associated to numbers. All colors in the same groups are merged.

Beyond the fact that the visual quality is improved, the diversity of the colors in the image is reduced. This means less different color coefficients will have to be sent to the decoder. Moreover, on spots where a group of color coefficients have been merged, only one representative of this group will need to be sent to avoid useless redundancy. The decoder however, needs to be told where these groups are. The easiest way to do this it

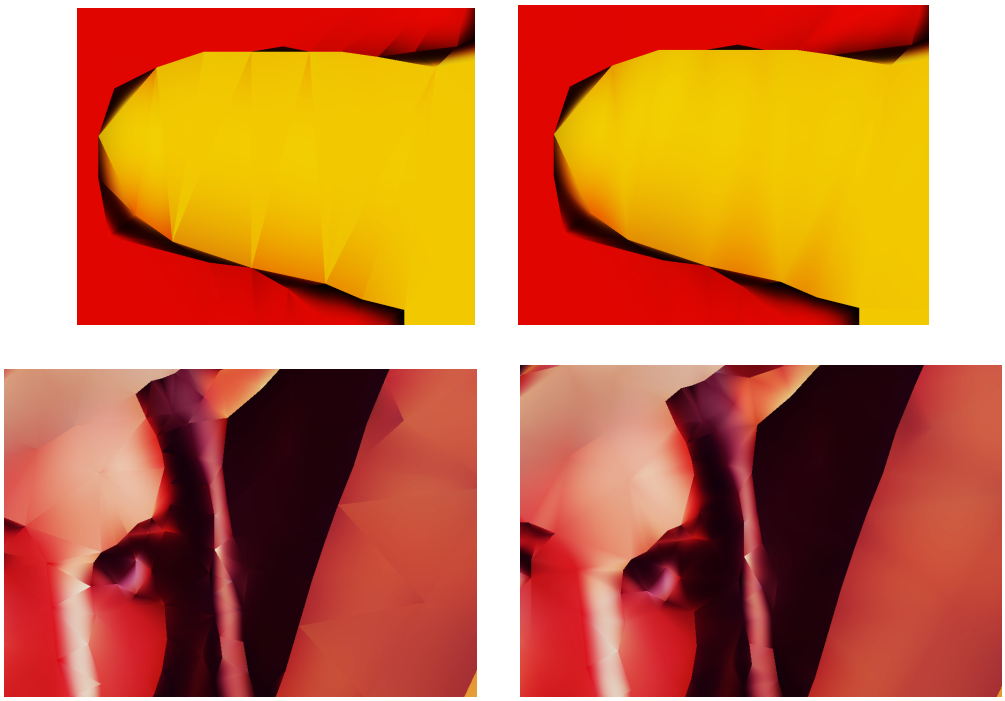


Figure 5.5: Selective color merging examples.

to transmit the discontinuity map in the bitstream.

5.2.3 Color space quantization

3D color quantization

Every color we use to describe the image is originally coded on a 24-bit value, which allows to use more than 16 millions of colors. Not all these colors are used in a single image. Schemes have been developed to reduce the number of colors used to describe an image while introducing as little distortion as possible. Colors from the reduced subset require less bits to be referred to. As it has been explained in 3.3.3, in our context, only a few sample color values are hard-coded in the bitstream while the rest of the image is deduced by interpolating these values. Quantizing the sample color values does not reduce the precision of the gradients, and thus does not distort the image as it does with raster images.

There exists lots of different color quantization methods. Some rely on simple algorithms like the popularity method [Cla95], some on somewhat more elaborate ones like the median-cut algorithm [Kru94] and the octree color quantization [GP90]. The previous schemes benefit from a high execution speed, but are far from being optimal from the rate/distortion point of view. For a more developed quality optimization one must

look at the k-means clustering algorithm [RT99, KYO00], or even the neural network approaches [Dek94, VB95]. These iterative algorithms converge towards a local minimum of the error measure. Thus, they have an influence on the distortion, but since they usually take the number of clusters as an input, they do not control the rate. For an efficient, rate-distortion driven color quantization, a clustering method with a variable number of clusters must be developed. A attractive approach is the pairwise clustering method described in [VGS97], because it ensures we have the absolute minimal distortion for a given number of clusters. This bottom-up method is unfortunately highly complex, computationally speaking, and leads to surrealist execution times. To keep the computation time reasonable, we opted for a top-down approach inspired from [RT99]. The idea is summed in the following lines:

- 1: We start with a very small color palette containing $n = n_0$ representatives
- 2: The palette is optimized with respect to a distortion measure. n is constant in this step.
- 3: The zones of color space that create the maximal image distortion are identified.
- 4: The precision in these zones is refined with the introduction of k new color representatives.
- 5: If the rate/distortion measure is not optimal yet, we go back to 2, with $n \leftarrow n + k$. Otherwise the algorithm exits.

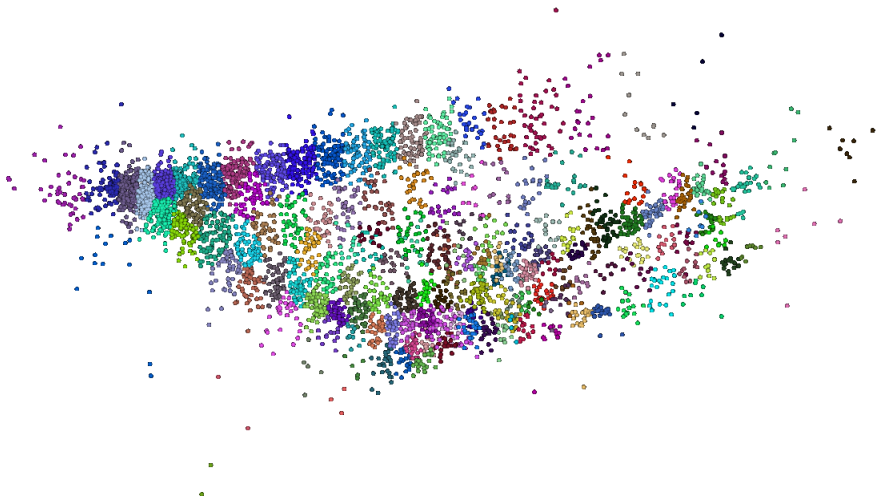


Figure 5.6: Colorspace clustering illustration. All colors appearing in the image (dots) are grouped in color clusters.

Let us clear the important point of this operation sequence:

In step 3, we have to find the color spaces that need to be better subdivided to significantly improve the quality. This is done the best possible way: by comparing the color

quantized image with the non-quantized one. This operation would be long if it was done on a pixel raster, but we can do it directly on mesh-structure, at the cell level[7]. A quadrature method allows us to compute a cell's quantization error with only two matrix-vector products. The new color representative are then introduced in step 4 to explicitly improve the colors from the cells presenting high quantization distortion.

In step 5, we measure the quality of the current image representation with a functional that comprises both rate and distortion. Indeed, adding new color representatives monotonously reduces the distortion, but increases the rate. More entries in the color palette mean that the palette itself will be bigger, and that the alphabet formed by the indices grows, requiring more bits per symbols. We chose the best compromise as the number of representatives that minimizes the following functional: $c = distortion + \lambda.(number\ of\ representatives)$[8]. When new color representatives are added, the first term decreases and obviously the second one increases. Within the first steps, the distortion decrease is very important, so c also decreases. The more representatives we have, the smaller are the decrease of *distortion* between two adjacent steps. At one point, the increase of the second term will compensate this decrease and c will start to grow. We choose the optimal number of representatives as one we had just when c started to grow. Tuning the parameter λ is the way of modifying the nature of the compromise we want between rate and distortion. High values of λ favor low bitrate at the expense of a lower image quality, since the second term of c will tend to compensate the first one earlier. Similarly low values of λ favor quality even if it means higher rates.

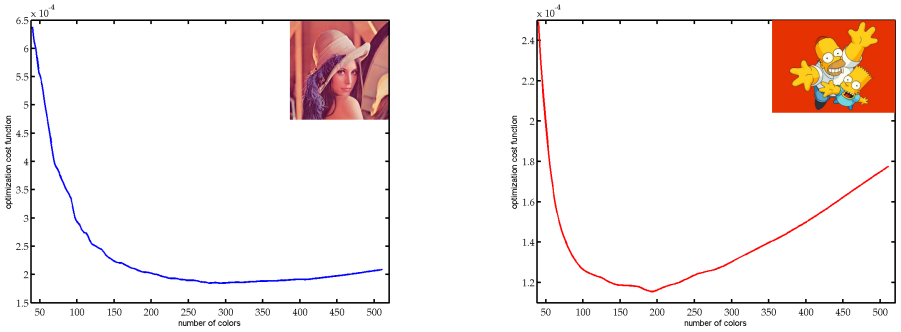


Figure 5.7: Evolution of the cost function c for the color palette generation of two different images with the same trade-off factor λ.

On the figure 5.7, we follow the evolution of c for two images. The second image (right) natively uses less colors than the first one. Consequently, it will be well approximated by a small number of clusters and its distortion measure will tend to plummet earlier than for the first image. For this image less representatives (and thus a lower bitrate)

[7]This is a substantial speedup, since the cells can contain several hundreds of pixels in average at our working bitrate.

[8]the number of representatives directly influences the rate, so this indeed a rate/distortion measure.

will be use for the same image quality. We have what we wanted: a scheme that *adapts the bitrate* to the image content.

Once the palette generation process has come to an end, the color sample values are quantized to their closest representative.

One thing that has not been discussed yet is the nature of the color space in which the color VQ takes place. The scheme that has been described does not technically depend on a particular color space, and can be integrated in any of them we know. The most common one is the RGB color space. Psycho-optic studies have proved that the correlation between the numerical distance in the RGB space and the perceptual one is not very good. The scheme exposed above relies at several points on the squared distance in the color space as a measure of distortion. For this to be coherent, the computed distance should express the real visual discrepancy between two colors. Perceptual color spaces like La*b* or Luv have been introduced, but their topology is uneven and the conversion from and to the RGB space are non linear. color values in the YCbCr space, often use in compression task, are easier to obtain from the RGB value (simple matrix-vector product), and the correlation between numerical and perceptual distance is higher than in RGB. For the actual uses of our compression scheme, we used the YCbCr space as a compromise between space regularity, easy computation and correlation with human perception.

Separate Y/C quantization

A drawback of the scheme described in the previous section is the execution speed. Indeed, 3D VQ is efficient in this case but it is also time-consuming. Moreover, the structure of the color sampling is so that the different color samples representing the same object or an homogeneous region in the picture often share the same chrominance, while their luminance tends to vary a lot more. When looking at the distribution of the color points in 3D, we notice that, in the CbCr space, the color form recognizable cluster whereas the distribution along the Y axis is more dispersed. A profitable way of speeding up the quantization pass is to perform a 2D quantization in the CbCr space while the Y component of the colors is handled separately. For the 2D CbCr quantization, we can use the same method as for the 3D case.

Another interest we have in splitting luminance and chrominance, is that we can perform a chroma subsampling. The human visual system is less sensible to chrominance variations than luminance variations[9]. In our context we can use it too by using lower order chrominance representation when it does not deteriorate the visual quality. For example, if we use 2^{nd} degree polynomials to describe the colors, we can compare the

[9]This characteristic is used in many compression algorithm under the names 4:2:2, 4:1:1 or 4:2:0 chroma subsampling.

2^{nd} degree chrominance representation with a linear (1^{st} degree) chrominance representation in each cell ; if the integrated difference between both representation stays within a tolerance threshold, the cell is flagged to be chroma subsampled.

Unfortunately, chroma subsampling cannot be done equally for every cell. With a native 2^{nd} degree polynomial approximation, some cells capture real quadratic chroma variations, and replacing it causes important distortion. Chroma subsampling can only be done in certain cells (approximately half of them in the lena picture for an imperceptible quality loss). Chroma subsampling obviously reduces the bitrate, since it reduces the number of chroma samples to encode. In return, a map signaling which cells have been chroma subsampled has to be sent to the decoder, but this addtional information remains negligible (about 1% of the color bitstream).

Depending on the image, the Y/C color quantization with chroma subsampling reaches compression factor of 1.5 to 2. Separate Y/C quantization definitely has the speed advantage over full 3D vector quantization, but in terms of rate/distortion performances, the latter remains the most interesting method (3D VQ about 1.3 times more for the same visual quality).

5.2.4 Decoding: progressive reconstruction.

In 5.2.2 we have seen how color values can be merged at vertices and edge points position. As a result of the selective color merging scheme we face the following situation:

- There are color values attached to the vertices of the mesh. Depending on the discontinuity map, the vertex can carry from 1 to n_v colors where n is the number of adjacent cells.
- There are color values attached to edge points. An edge point carry from 1 to n_e colors where n_e is the number of adjacent cells to the edge. For still images, $n_e = 2$.

To allow a progressive reconstruction of the images, the color coefficients stream is organized in successive passes. The passes are of two kinds: the vertex passes and the edge passes. In each pass, one color per entity is transmitted. For example in the first vertex pass, one color per vertex is placed in the bitstream ; in the first edge pass, one color per edge is sent. In the following vertex pass, a additional color per vertex is sent, but only for those vertex which carry more than one color. Similarly, in the second edge pass, the second edge color is sent for the edge whose color have not been merged[10]. Once all the color values attached to an entity has been transmitted, we say the entity is complete, otherwise it is still incomplete, and its remaining colors will be transmitted in future passes. The color coefficients order is summed up in the following list:

[10]For the 2D image case, there can be only two edge point passes since an edge has at most 2 adjacent cells. For videos, more point passes can be necessary.

- 1: 1^{st} vertex pass: one color per vertex.
- 2: 1^{st} edge pass: one color per edge.
- 3: 2^{nd} vertex pass: one additional color per incomplete vertex.
- 4: 2^{nd} edge pass: one additional color per incomplete edge (last edge pass in the case of still images).
- 5: 3^{rd} vertex pass: one additional color per incomplete vertex.
- ...
- i: n^{th} vertex pass...

This goes on until all colors have been placed in the bitstream. At the decoder side, the data reading follows the same order, and between each pass, a preview version of the image can constructed. After a given pass, a part of data is still missing but with simple interpolation strategies, they can be estimated with the coefficients that have already been received. After the very first pass for instance, all we have is one color per vertex. The image reconstruction algorithm will then assume that the colors have been merged at every vertex position, and edge colors will be interpolated from the two adjacent vertices. This obviously gives a very blurry visual result, but the important color variations throughout the image can already be distinguished. After the second pass (the first edge pass), the received edge colors replace the previous interpolation and refine the quality of the pictures. The second vertex pass will add one color to all incomplete vertices, so all the concerned vertex colors will be updated in the preview, as well as the interpolation depending on these vertices.

As soon as the first pass has been entirely decoded, the preview decoded image can be refreshed with every color coefficient that comes in (and not only after each pass). Finally the progressivity of the decoding is ensured at the coefficient level: starting from the very first decoded color coefficient, a visual reconstruction of the image can begin, which is refined with every newly decoded color value.

Note that the synchronization between the encoder at the decoder regarding the coding order of the colors is assured because this order is determined by the discontinuity map[11] only, and the decoder has access to the same discontinuity map.

[11] see 5.2.2.

Figure 5.8: Progressive image reconstruction state after partial decoding of the bitstream. Here follow the decoded bitstream percentage from left to right and top to bottom: 35%, 67%, 76%, 90%, 96% and 100%.

Chapter 6

Results

6.1 Representation of images

In this section, we comment on the quality of the proposed image representation. Our initial goal has been achieved in that we have designed a method to convert bitmap images into a new representation involving higher order content description. The innovation in relation to the state of the art is that we tend to give a faithful representation of the original image, and not a stylized one. Moreover contrary to most existing vectorization schemes, our system is completely automatic, and requires no other manual intervention than providing a quality coefficient. With the rate/distortion driven conversion[1], the most problematic areas are identified and the representation precision is improved there as a priority. This gives the process a good intuition on how to use the representation resources, and leads to meshes that intelligently fit onto the visual data. Low polygon image approximations are thus able to capture the important visual information thanks to the flexibility of the representation organization, which is attractive compared to traditional grid structures. This makes the scheme interesting for compression purposes (see 6.3).

When it comes to high-quality image, the number of cells required to reach 35dB and higher grows exponentially. Although it impairs the compression performances, the quality does improve. When a sufficient number of triangles are used, a *visual lossless quality* can be achieved, which means that we have changed the image representation structure with no visible quality loss. This opens up the way of other tasks than compression for this image representation scheme. Outlooks on possible uses of the mesh-based image representation are mentioned in 7.2.

[1] see 3.2.3.

Figure 6.1: Low bitrate image representation. Left: mesh-based codec (0.197 bpp). Right: JPEG (0.199 bpp).

6.2 Representation of videos

The results concerning the video representation are unfortunately a little bit disappointing compared to what has been achieved for images. Obviously, since the method is the same in both cases, we observe the same tendency as for image, namely that the scheme manages to capture the essential information of the original video. Like in 6.1, we notice the tendency of the algorithm to automatically place the resource where they are really needed. Our special video object structure is innovative and has the advantage to easily handle problems like objects apparitions or overlapping compared to the 2D deformable mesh approach. Moreover since it is based on a generic Delaunay mesh, powerful generic algorithms and method can be used for the construction and for additional post-processing.

The flaw in this method is that we try to handle the time dimension as one of the spatial dimension whereas its perceptual role is very different. The main problem arises when an object from the video moves *quickly* throughout the scene. When the object goes through a given spatial location, its "life expectancy" at this location is very short. Yet, in the video object structure, the object's presence at this location at the right time is represented by dedicated tetrahedrons that carry the object's color. The issue comes from the fact that the tetrahedrons cannot have an arbitrarily irregular form[2]. In such a situation, when cells are introduced that depict the moving object, they tend to have a longer time extension than they should and thus are responsible for a certain visual

[2]Tetrahedrons that are too flattened, for example, are the source of bad conditioning problems at the color fitting step.

inertia. This is a very annoying artifact we refer to as *motion ghosts*.

Figure 6.2: Motion ghosts in a video sequence.

The visual impact of motion ghosts is reduced by the use of the *artificial motion blur*[3], but it still is a strong handicap for the video quality.

As mentioned earlier, the problem comes from the tetrahedrization scheme. We have seen in 4.1.2 that we first supervise the distribution of the vertices in the spatio-temporal video volume according to an optimization strategy, and then finally perform a Delaunay tetrahedrization on the set of vertices. The position of the vertices obtained after the relaxation is generally very satisfying, but the tetrahedrization step that follows is unsupervised. Indeed we have chosen to rely on unconstrained Delaunay tetrahedrization for compression purposes[4]. This is the source of the scheme's weakness when rapid motion occurs: the motion tends to attract vertices to refine its geometrical representation, but the tetrahedrization of this zone will be done considering only the available set of vertices, with no special regards to the structure of the motion. To solve this problem at its source, the way we perform the Delaunay tetrahedrization has to be reconsidered. Constraints have to be added to the process to force the cells to conform with the motion flow. This would obviously result in an increase of the computational complexity, in particular for the detection of the motion flow and its translation into usable constraints. Moreover, the obtained mesh would not be a straightforward Delaunay tetrahedrization anymore: additional informations would need to be sent to the decoder in order to ensure a one to one reconstruction, which would cause harm to the bitrate. A roundabout way to improve the representation of motion it to refine the volume sampling at the right places. This is what the rate/distortion driven mesh construction, and other more generic works like [Sch97] try to do. In our particular context, introducing more tetrahedrons reduces the visual impact of motion ghosts but, up to a reasonable maximum of cells, never make their influence completely disappear.

The presence of the motion ghosts draws a limit for both the subjective and numerical video quality. This video representation scheme, as it is now, cannot be visually lossless

[3] see 3.3.4

[4] The reasons for this are exposed in 3.1.2

like our image representation can be, nor does it represent a viable compression method. A more evolved method of constructing a mesh that intelligently conforms to motion is necessary for this approach to reach a better quality range and become a conceivable video description format.

Another drawback of the 3D mesh approach for videos as we have described it throughout 3 is the computational time. Processing a whole space/time volume in an iterative process like our rate/distortion driven mesh construction is very costly for the nowadays available computational power. Nevertheless, we can remark as compensation that almost the entire computational effort is located at the encoder side, whereas the decoder's role is limited to the reorganization of the incoming data. From the decoding angle, mesh-based video reconstruction require a lot less processing performance than the present video codec like MPEG2 or MPEG4. As an acceleration outlook, we can also notice that the processes involved in the conversion (the color fitting step in particular) can be easily distributed on multiple processing units, which goes in the way of the upcoming computer architectures.

6.3 Compression performances

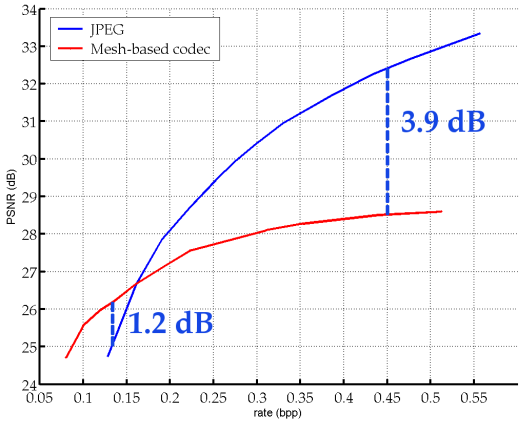


Figure 6.3: Rate/distortion coding performance of the mesh-based image codec compared to the traditional JPEG codec on the Lena picture.

The results on the presented curves present the rate and distortion results measured on image whose bitstream has been hard-written and decoded[5]. Thus every needed piece of information is accounted for in the displayed bitrates.

As we can see, the image coding scheme performs better than JPEG only in the very low bitrate zone (approximately up to 27dB). Above a certain quality threshold, JPEG

[5]all JPEG measurements have been made with the software GIMP 2.2

surpasses the mesh-based coding scheme and the gap between them then continues to grow. The reason for both phenomena comes from a the structural difference between the two representation. At low rates, the JPEG scheme has a regular block structure with a very low precision at its disposal. Only basic intensity variations along the grid's direction can be rendered. The mesh-based coding also has only a limited power of representation, since it is limited to second degree variations by construction. Yet the cells are free to be placed at the most strategic places in the domain, which results in a better use of the available resources. The gain resulting from this adaptation even over-compensates the overhead needed to describe the mesh. At high rate on the other hand, JPEG coefficient's precision gets finer, and the description power of the DCT block improves drastically. JPEG is less handicapped by it regular grid structure. As for the mesh-based scheme, the color variations in the cells is still limited to quadratic functions, so the only way to improve the quality is to increase the number of cells. Complicated texture-like region in the picture will require great amount of cells to be correctly rendered. Additional cells in the mesh-based scheme are a more expensive resource than coefficients precision in a DCT quantization process, which explains the growing performance gap as rates rise.

This is illustrated in figure 6.4, displaying the final triangular mesh generated for the Lena picture at two different quality levels. We see that Lena's feather-hat attracts a lot a cells and coding resources and occupies an important part of the bitstream as the quality improves.

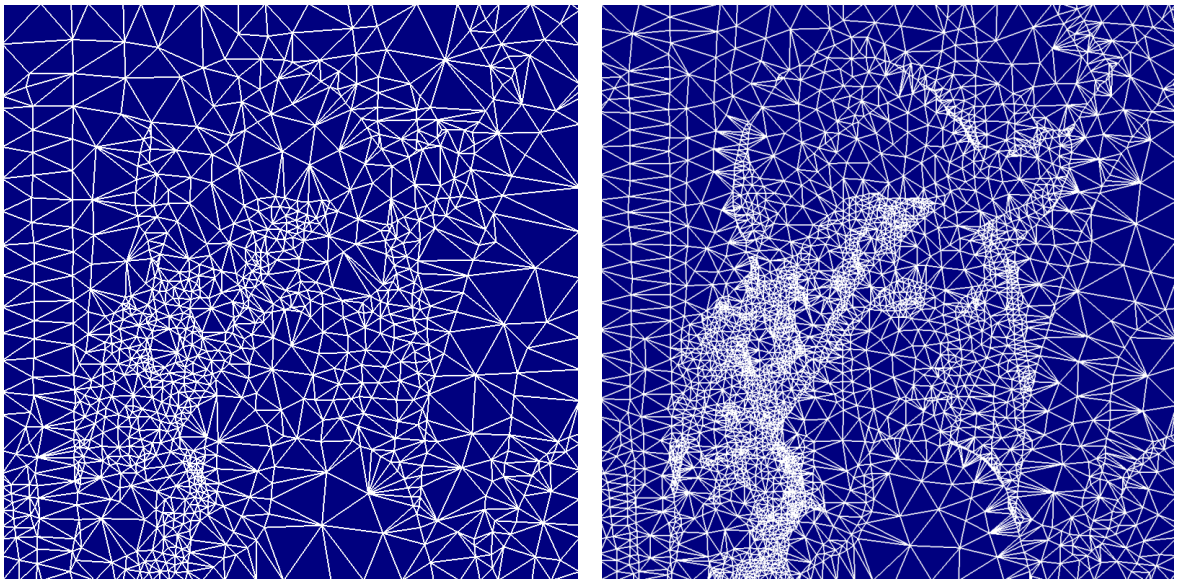


Figure 6.4: Mesh-based segmentation of the Lena picture at two different quality levels.

The mesh-based image rate/distortion curve eventually continues going up to reach the high-quality zone but with a very low slope.

As it has been said, the scheme inability to compete with JPEG at high rates comes from the difficulty to describe irregular intensity variations with second degree polynomials. In smooth areas however, the mesh-based image compression does a good job since the cells adapt the nature and orientation of the smooth shading. Only a few cells are needed to reach a high-quality result. This is confirmed by the rate/distortion coding performances on smooth, untextured images as displayed on figure 6.5.

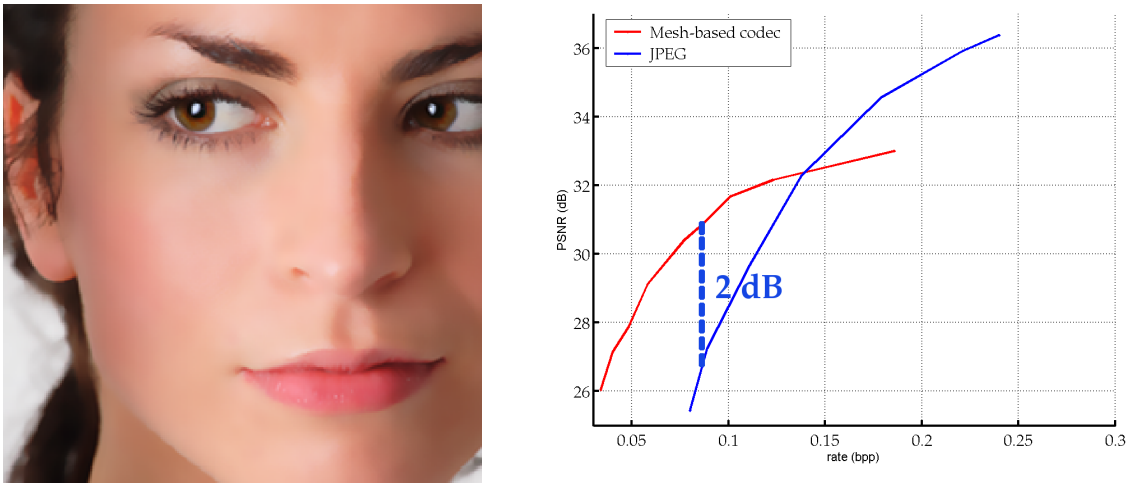


Figure 6.5: Left: original raster graphics picture. Right: rate/distortion performance comparison between the mesh-based codec and JPEG (resolution 512x512).

Another interesting feature peculiar to this mesh-based image representation is that, for smooth areas, the image representation with our type of mesh is resolution-independent. Indeed when a picture area composed of n pixels is perfectly approximated by a cell with a ploynomial model, The same area composed of $2n$ pixel would still require only one cell. The visual quality would be the same but the bitrate would be divided by two. Consequently, the mesh-based approach becomes even more interesting when dealing with higher resolution smooth images. If we consider the same image in figure 6.5 at a higher resolution, the superiority of the mesh-based representation in that case become obvious (figure 6.6).

It is important to note that this is only true for smooth areas, for which the polynomial approximation is adapted. When it comes to textured zone, a higher raster resolution introduces more complex visual information that will require more cells to be depicted with the same precision level.

Moreover, if we analyze the reconstruction error distribution of the mesh-based coding scheme, we see that an important part is located on the edges of the image. This is due to that fact that objects boundaries and contours are approximated by line segments (the borders of the cells). Also the optimization of the vertices positions sometimes results in

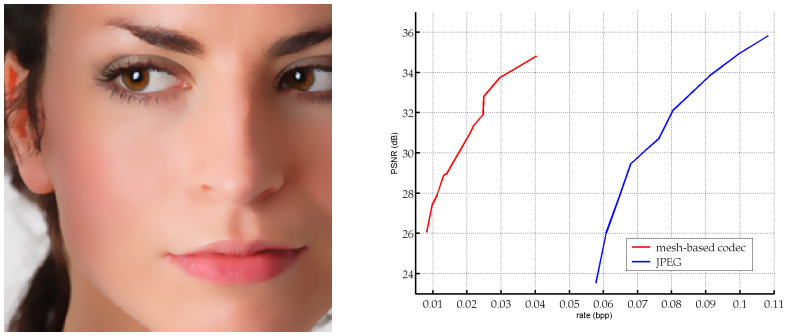


Figure 6.6: Rate/distortion performance comparison between the mesh-based codec and JPEG (resolution 1024x1024)

cells frontier not being exactly superimposed with the pixel-based discontinuities, but are translated by a few pixels as we see on figure 6.7.

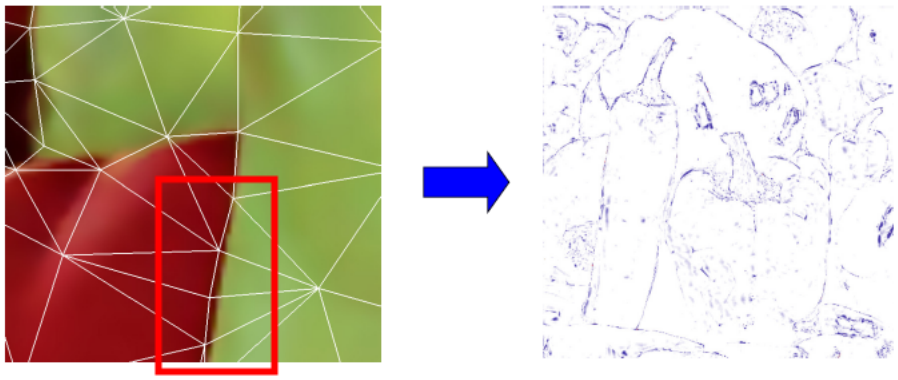


Figure 6.7: Left: illustration of the discontinuity mismatch. Right: pixel to pixel squared difference between the original image and the mesh-based coded image.

While practically insignificant to the human eye, this artifact introduces numerical error and impair the rate/distortion performances of our scheme. The PSNR quality measures of the image encoded with the mesh-based codec suffer from this problem, and the does not always reflect the perceived quality as one can see on figure 6.8.

A more perception-oriented metric than the PSNR might acknowledge a better representation quality to our method at middle rates. The discrepancy between perceived and measured quality is even more noticeable on figure 6.9.

On this image, a purely numerical rate/distortion comparison is does not make a lot of sense. Here follow a visual comparison of the two coding scheme at comparable bitrates:

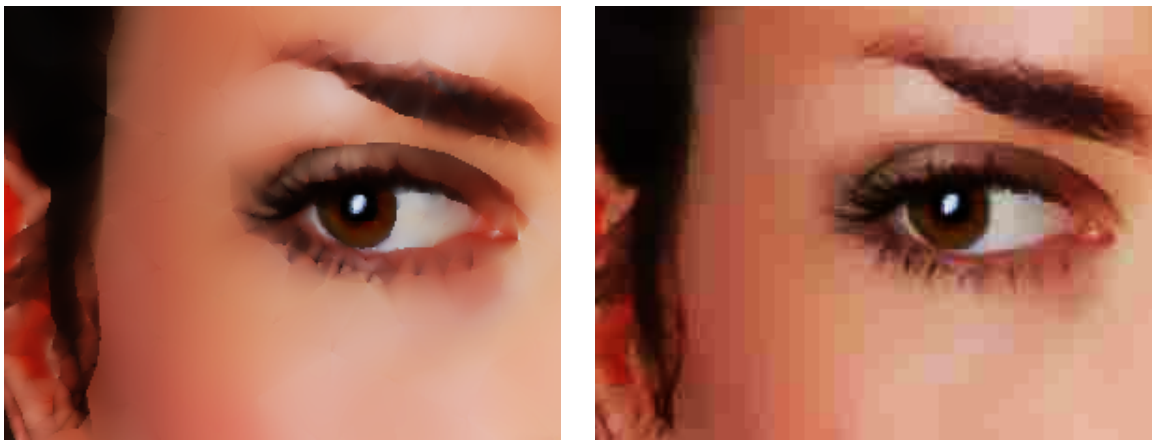


Figure 6.8: Left: Mesh-based codec output image at 34.31dB. Right: JPEG image at 34.57dB.

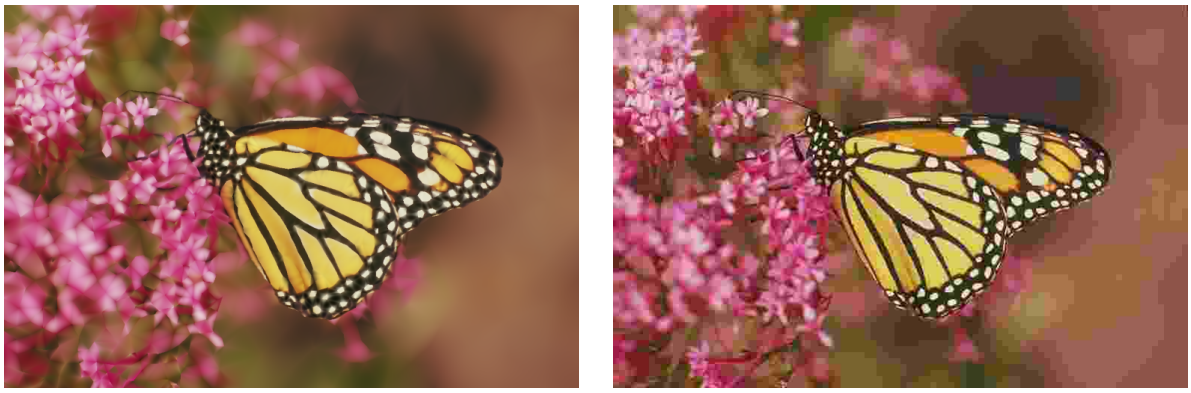
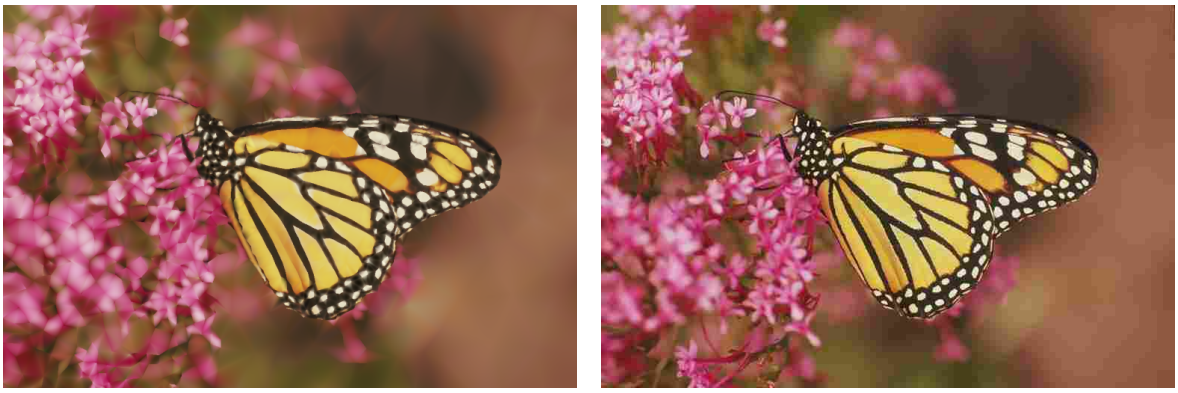
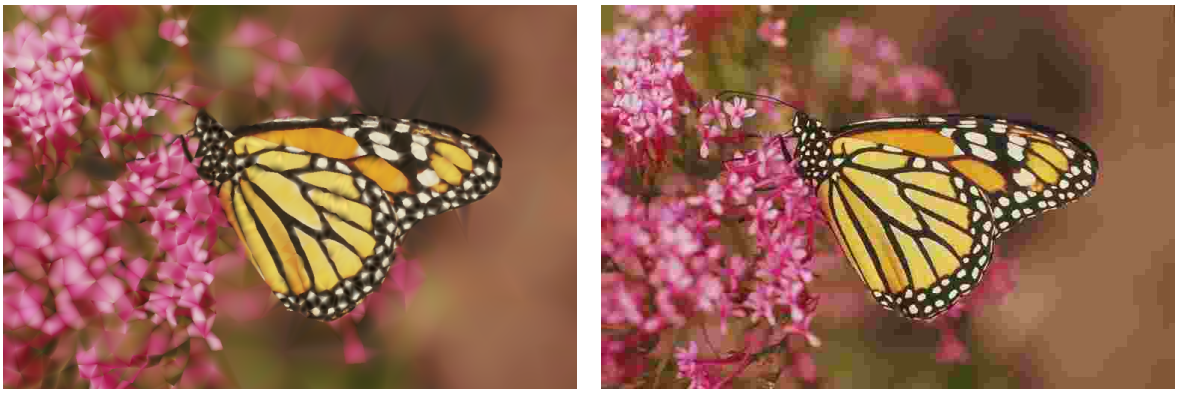


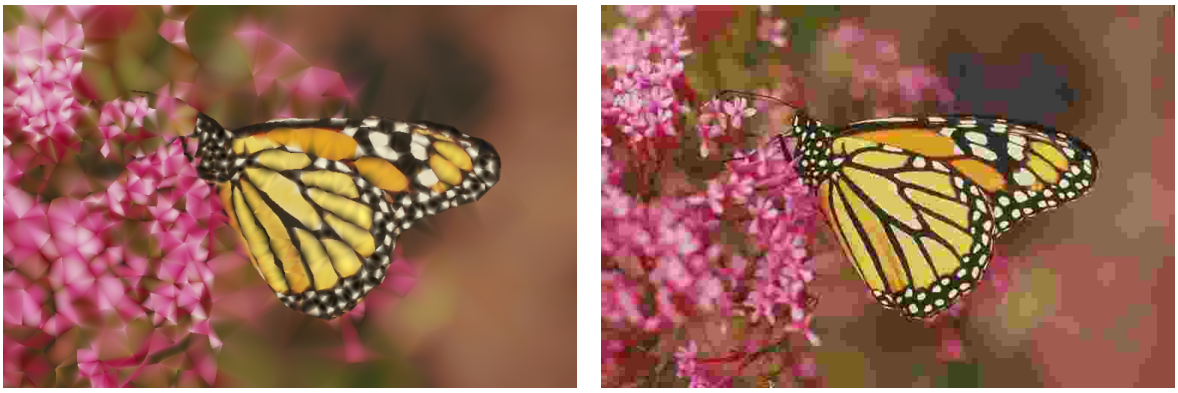
Figure 6.9: Left: Mesh-based codec output image at a measured PSNR of 25.38 dB. Right: JPEG image at 25.43 dB.



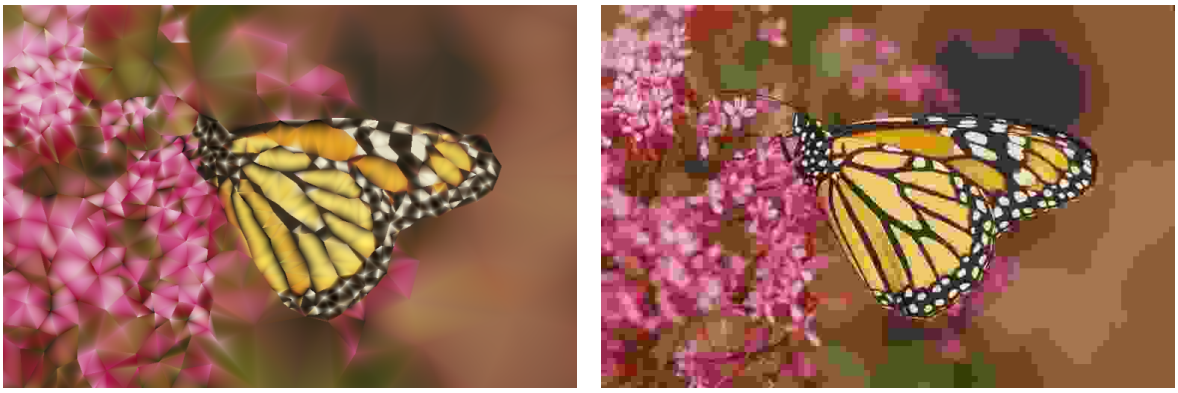
Left: mesh-based codec at 0.339 bpp. Right: JPEG at 0.341 bpp.



Left: mesh-based codec at 0.238 bpp. Right: JPEG at 0.241 bpp.

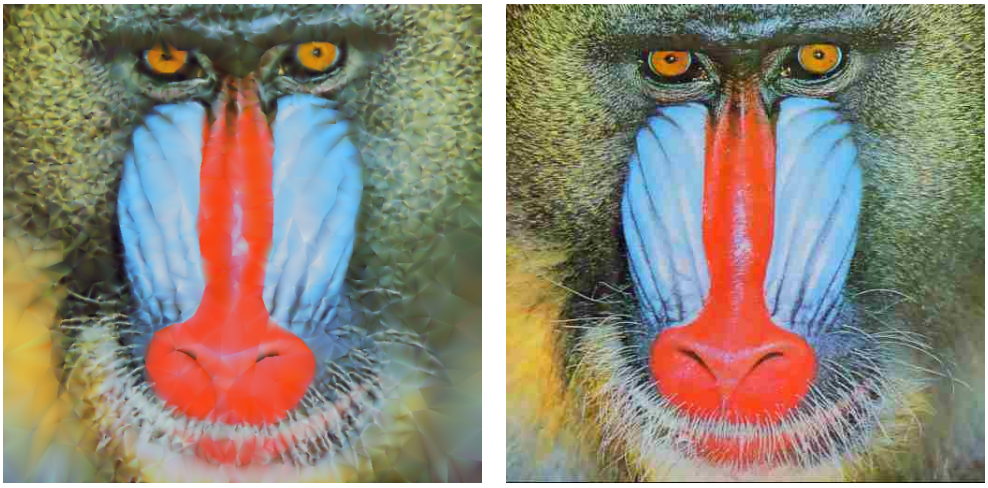


Left: mesh-based codec at 0.164 bpp. Right: JPEG at 0.177 bpp.



Left: mesh-based codec at 0.121 bpp. Right: JPEG at 0.133 bpp.

On this particular image, the mesh-based codec delivers a more pleasant image at low rate, whereas the blocking artefact of JPEG highly distorts the rendering of smooth areas. Yet in the case of images containing large textured area, JPEG clearly has the advantage as we see on the following figure.



Left: mesh-based codec at 0.689 bpp. Right: JPEG at 0.676 bpp.

As mentioned in the "previous works" section in 2.2, it would be interesting to compare our rate/distortion performance of the mesh-based image coding presented in [MPL00a]. According to their results, both suffer from the same problems: they have interesting performances at low rates but to achieve higher-quality they need to multiply the number of cells, which has a strong impact on the rate, and they have difficulties reaching high-quality images with a decent bitrate. Our scheme however tends to have the advantage at low rates, presumably for the following reasons. First, the disposition of the cells is not free in the hierarchical mesh structure used in [MPL00a], the vertices of the mesh cannot be moved and the mesh adaptation to an image can come only from successive cell subdivisions. Thus an intensity discontinuity with the wrong orientation will require many cell subdivisions to be correctly approximated, while our flexi-

ble Delaunay mesh can adapt to any discontinuity orientation. Secondly, The intensity representation in [MPL00a] is global and impose continuous transitions between cells, whereas we work on a cell basis, allowing abrupt discontinuities between adjacent cells. This certainly improves the representation quality at objects boundaries. [MPL00a] obtains rate/distortion results lower than JPEG on the lena picture, whereas we have seen that our mesh-based approach could compete with it at low very low rates.

From a decoding point of view however, the use of a hierarchical mesh is more attractive because is allows the progressive reconstruction of the image. Indeed their hierarchical mesh is organized in layers, so the image transmission can be organized layer by layer without adding overhead. At the receiver side, the image can be progressively reconstructed starting by a coarse image formed by the higher-level cells, and being refined by lower level subdivision cell contributions. Our mesh does not have this structure, but a single-layered one so a hierarchical progression is not possible. Yet, a progressive transmission and reconstruction based on geometrical importance is implemented in our scheme and allows progressive previewing of the decoded image[6], whereas JPEG cannot.

[6]The progressive transmission and decoding is presented in 5.2.4.

Chapter 7

Conclusions and Outlook

7.1 Conclusions

During this Diplomarbeit, a particular representation of images and videos based on meshes has been studied. Important aspects like the representation characteristics, the construction process, and the compression performance have been analyzed in order to judge the qualities that such a representation can offer. Of particular interest to us were the visual performances of a video description format based on tetrahedral meshes, and the compression results a mesh-based description could reach on both images and videos.

As seen in section 6.1, the mesh-based image representation is interesting. It has a flexibility that enables a strategic distribution of the rendering resources at low-polygon approximations, and it is capable of high-fidelity at upper quality ranges. Moreover, the structure opens the doors to other interesting tasks that will be detailled in section 7.2. Unfortunately the same conclusion cannot be drawn for videos. The extension of the concept used for 2D informations to 3D spatio-temporal video volumes is not trivial due to very peculiar nature of motion in digital videos. Despite all the precautions that have been taken and the efficient tools that have been developed (3D filtering, tetrahedral Delaunay mesh structure and Voronoi diagrams, 3D generic rasterizer[1]), efficiently capturing motion with tetrahedral cells is still a problem that requires some more theoretical and computational advances. The phenomenon referred to as "motion ghosts" in 6.2 prevents relevant representation and compression measurements from being done.

For the previously exposed reasons, interesting compression conclusions could, at this point, only be drawn for images. During the time of this Diplomarbeit, the whole image compression codec was implemented, tested and integrated in the *graphite* platform. The results have been presented in section 6.3. We have said that the flexibility and

[1]The generic rasterizer is presented in Appendix A

adaption abilities of a mesh lead to interesting image description results. Interestingly it is also the reason why the compression performances are not up to state-of-the-art JPEG and obviously to JPEG2000. Efficient compression algorithms come from stable *structure*. The regular grids used in JPEG and JPEG2000 are very basic compared to an adaptive triangulated mesh, but the compression mechanisms that will be built upon it will be all the more efficient since the structure is known, and they can take advantage of it[2]. Adaptive meshes on the other hand have a very free geometric structure that makes it difficult to derive generic and efficient information compression methods, as our attempts including the ones presented in 3.3.3 tend to show. They also prevent hierarchical description and transmission, but we have seen that our scheme nevertheless proposes progressivity features in transmission and decoding.

Another noticeable feature is the large complexity difference between the encoder and the decoder. The computational burden at the encoder is rather important, mainly due to the iterative mesh generation process that pursuit the maximal rate/distortion trade-off. The decoder is kept very simple, and except from entropy decoding, it is limited to a geometrical reconstruction of the incoming information. Such a discrepancy is interesting in *encode once, access many times* type scenarios, like access to visual information in the internet or transmission of images over communications networks to small devices with limited computational power. Moreover, we know that the effective display of the images relies on a quadratic interpolation technique which can be entirely taken over by a fragment shader program on a GPU (see 3.3.4). This not only guarantees a high reconstruction and display speed, but also decreases the information transmission burden to the graphic board. It tends to become an important point now that the power of graphics device become limited by this transmission rate, and we know that most of this rate is mobilized for the transfer of images and textures. Last but not least, it is important to recall that the converted images and videos are described by equations, and thus are completely resolution-independent. Linear, affine and even projective transformation can be performed very easily contrary to the heavy pixel grid manipulation required in the raster graphics case. At a time where visual content is meant to travel on various platform, all having different resolutions and display characteristics, this feature is appealing.

Finally, if we leave the compression performance aside, the system that has been developed is a fully automatic image and video conversion tool, that distributes the resources according to a quality/compactness trade-off. In that sense it enters the category of vectorization methods for a photorealistic representation of images and videos that has been evoked in 2.2. Vectorization and photorealistic vectorization in particular are areas of research of growing importance. More and more effort is being put in the design of efficient and elaborated methods to handle vector graphics and its relation to raster graphics as its presence on our computers grow (Microsoft's new operating system, in-

[2]For instance : fixed quantization tables for DCT coefficients, Zig-Zag scanning and Run Length Coding, Embedded Zero-tree Wavelet coding, *etc*.

ternet visual content, ...). I hope the results of this study will bring some experience on how vectorization can be done, and its limitations will give hints on how to do it better.

7.2 Outlook to further development

Compression

In the results section 6.3, we have seen that the mesh-based image compression has an interest since it is capable of decent rate/distortion performances at low rates. However we have noticed that the color representation in a cell being limited to polynomial functions is an handicap that encourages the cells proliferation at higher rates as displayed on figure 6.4. Textured areas in particular are concerned. In these area, the spectral representation shows a more attractive coding ability due to DCT's tendency to decorrelate and concentrate the energy on a few spectral coefficients. For smooth, non-textured area on the other hand, an adaptive polynomial scheme seems to be more appropriate as we saw on figure 6.5. The spatial adaptation of tiangular cells to smooth-type areas allow their signalization on a low amount of bits without creating JPEG's blocking artifact. To take advantage of both behaviour, an interesting approach would certainly be to design a mesh-based hybrid scheme. The hybrid scheme would still rely on a mesh representation, but the color representation inside each cell would either be done by a set of polynomial functions, or by an adapted spectral representation, a DCT on a triangular domain for instance. The principle is attractive: Textured areas in an image would require less triangular cells to be correctly represented and the slope we observe on both figures 6.3 and 6.5 would hopefully increase, however this outlook deserve a closer look. Interesting questions arise when practical problems come to be considered, like: which frequential function basis would we use ? How would the iterative mesh construction process be operated ? On what criteria do we base the choice of one or the other representation of a cell ? These are crucial aspects one has to considered when attempting an hybrid mesh-based image codec, yet this might be rewarded by rate/distortion performance that could compete with JPEG on a wider quality range.

A certain degree of progressivity in the decoding of a mesh-based image has been illustrated in figure 5.8. This result requires a clever ordering scheme of the color indices in the bitstream, but no additional information. Yet, the partial reconstruction can only begin after a first wave of coefficients (the first *vertex pass*) has been successfully decoded. By adding some additional information in the bistream, it might be possible to start the visual reconstruction even before all vertices have received their first color. At the decoder side, it would imply that we could interpolate the missing vertex colors from the ones we already have. The interpolation must be done in accordance with the semantics of regions in the image: indeed a vertex color must be interpolated from vertices that represent the same object or zone of the picture. This semantics is not known *a priori*

at the decoder side, which is why additional signaling is necessary. At the expense of a larger bitrate, the decoding progressivity of the image compression scheme could be improved.

As far as videos are concerned, the same observation that has been made for still image can apply to video: the scheme does a good job when it comes to representing smoothly-varying areas of the video volume but requires a lot more resources when dealing with textures. The biggest problem however remains the handling of motion. Future developments involving the tetrahedrization of a space/time video volume will need to focus on a tetrahedrization method that better conforms to the nature of frame-based motion, with regards to rate/distortion contraints.

Image processing

From a pure image manipulation point of view, the meshing of a still image has interesting properties to exploit. Its ability to isolate smooth zone and conforming to image discontinuities make it an interesting image segmentation method.

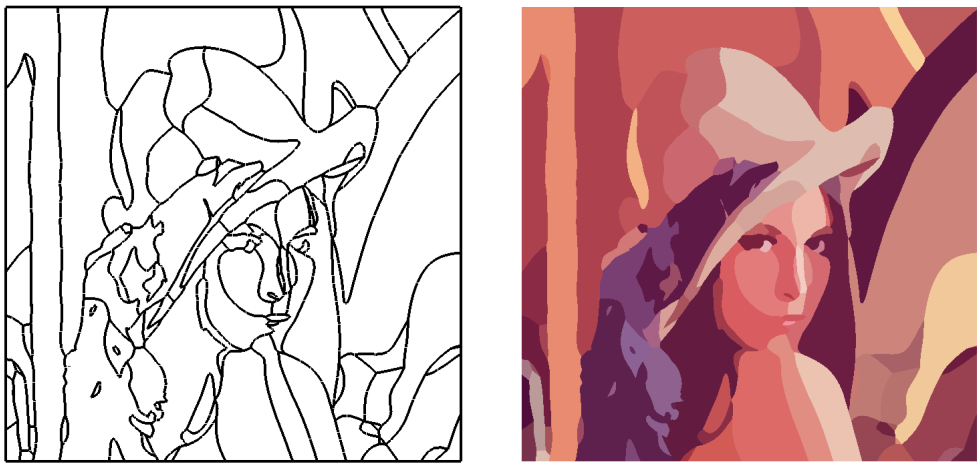


Figure 7.1: Example of a segmentation based on the mesh structure (boundaries have been smoothed). Left: detected segments. Right: regions filled with their mean color.

The conversion scheme can also be considered as an image vectorizer. By adjusting the conversion rate/distortion threshold one can get the method to focus mainly on the quality of the final result. As a vectorization scheme, our method cannot be competitive against the latest ones that have been developed for this very purpose. The use of triangles and tetrahedrons as primitive was motivated by the fact that the method had to be efficient from the coding point of view, but it is obviously not optimal to approximate objects and curves. A refinement is however possible if we consider higher-order

triangles. Instead of traditional triangles with linear edges, we have developed the possibility to bend the edges of the triangles in order to better conform with the original image geometry. The additional bitrate required to store this information overcompensates the quality gain, which makes this feature unfit for image compression, but when only quality is concerned it helps rectifying the flaws of a linear geometry.

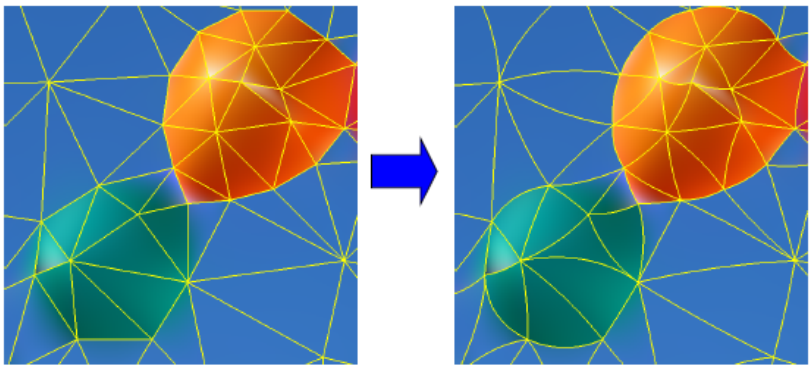


Figure 7.2: Introduction of a higher-order geometry. Top: segmentation of the bitmap image into cells. Bottom: result of the color fitting.

Finally, the last image and video processing application of the new image and video representation we will mention concerns non-photorealistic rendering. By store higher-level visual information (geometry, color gradients), the method facilitates the semantic altering of the image content. As a matter of fact, NPR-oriented uses of the formalism introduced in this report is already being considered.

Appendix A

The generic rasterizer

A.1 Motivation

In this section, we describe an important system module: the rasterizer. In 3.2, we have been describing a system that transforms images geometrical objects. It is inevitable at one point to have an interaction between the real-valued world of the geometrical object and the pixel-based image world. More specifically, we need a tool that is capable of determining and registering the pixels (or voxels) that are contained inside the cells of the mesh. Even if the image and video format involve natively only triangular and tetrahedral cells, a more general rasterizer has been developed to be able to cope with more elaborate processes[1].

For a given convex geometrical primitive (triangles and tetrahedrons in our case), the rasterizer determines and registers the pixels (or voxels) that are contained in it.

At several places in the conversion process, the services of the rasterizer are required. For instance:

- In the relaxation step[2], the points are updated according to the position of the centroid of their Voronoi cell. In this case, the centroid is the analog to a center of mass but using the saliency information as a weighting function. The saliency values are only available at integer positions since they are calculated on a voxel grid[3]. The centroid is calculated according to the following formula:
  for all the pixels v contained in a cell V, $c_V = \frac{\sum_{v \in V} s(v).v}{\sum_{v \in V} s(v)}$.

[1] Instead of processing the cells independently, we might consider processing groups of cells. In that case, a more generic rasterizer, not only meant for simplices, becomes necessary.

[2] see 4.1

[3] see 4.1.1

- In the color fitting step, the color information of all the voxels contained in a tetrahedron have to be compiled into a linear system whose solution gives the fitting coefficients[4]. The linear system is composed of the values of the functions (ϕ_n) and the original intensity I at the integer voxel positions inside a cell.

Both processes require the capacity to retrieve the pixels belonging to the volume (or surface) spanned by a Delaunay cell. Let us see how this was done.

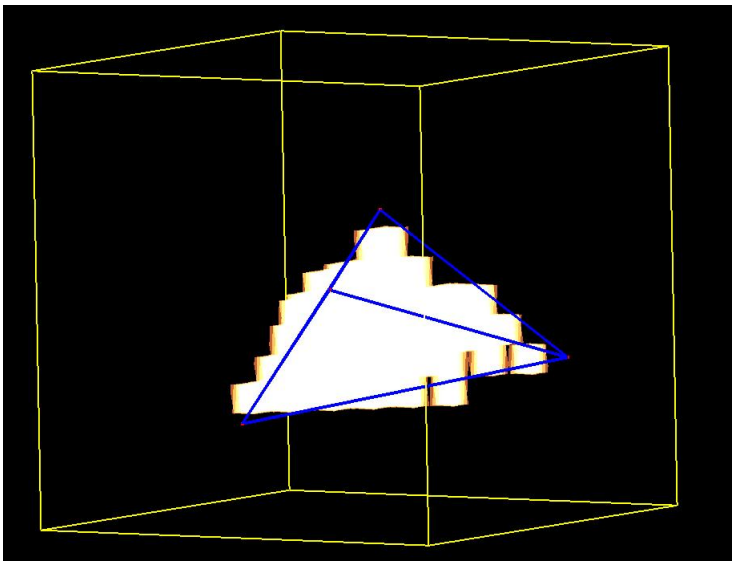


Figure A.1: Rasterization of a tetrahedral cell in a voxel grid

The generic 2D rasterizer and the 3D version use the same mechanism. For the description of the common features that follows, we will use the 3D notations since the 3D case is the most interesting and challenging one.

A.2 The algorithm

The following generic rasterizer originates from an idea from Bruno Levy. We call it *generic* because it works for every kind of cells, provided they are convex, and the operation that is executed for each rasterized voxel can be parametrized. All that the generic rasterizer actually does is listing the voxels contained in the input volume, and pass their reference to a user-chosen program for further processing.

The kernel of the algorithm is a modified version of the famous Bresenham algorithm [Bre65], also known as middle-point algorithm. The Bresenham method has emerged in the early days of computer graphics and was one the first to perform a rasterization

[4] see 4.2

operation, namely the actual drawing of a geometrical line segment on a pixel grid. We will begin the rasterizer description with a brief overview of the Bresenham method.

A.2.1 Bresenham rasterization of a segment

The Bresenham middle-point algorithm performs the rasterization of a segment on a pixel grid. This means that its takes the geometrical parameters describing a line segment as input and draws this segment on the pixel grid by lighting up the pixels that best represent the line's shape. There follows the explicit equation of a line: $y = \alpha.x + \beta$, and the implicit equation of the same line: $F(x, y) = a.x + b.y + c = 0$. α, β, a, b and c are of course linked with each other.

With the implicit formulation, we can test whether a point (x, y)belongs to the line or not. If $F(x, y) = 0$, the point is on the line, if $F(x, y) \neq 0$ it is either over the line or under it. By looking at the sign of $F(x, y)$ it is easy to determine whether (x, y) is over or under the line:

If $F(x, y) > 0$, (x, y) is under the line.

If $F(x, y) < 0$, (x, y) is over the line.

The Bresenham rasterization method is also called "middle-point algorithm" because at each step, the position of the *middle* of the segment separating two adjacent pixel centers is compared to the position of the theoritical line. The outcome of the comparison indirectly determines which one of the pixel centers is of the closest to the line, and the winning pixel will be marked as part of the rasterized line. The algorithm carries on doing this for each pixel couple along the line direction (linewise or columnwise). The result is illustrated on figure A.2.

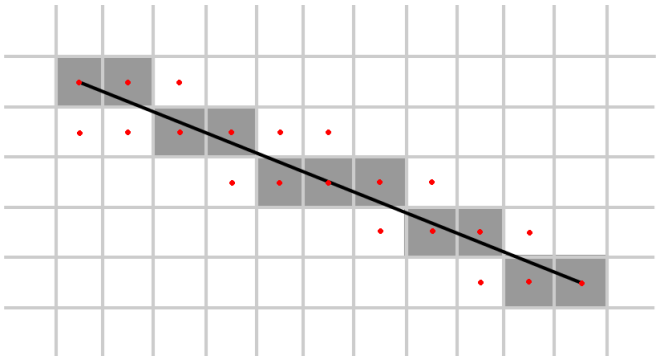


Figure A.2: The outcome of the basic Bresenham algorithm for lines. The thin black line is the theoretical line, the red dots are the pixel centers. The obtained rasterized line is formed by the dark pixels.

A.2.2 3D Rasterizer

The 3D generic Rasterizer uses a slightly modified version of the Bresenham algorithm. The latter will not be used to mark the pixels belonging to a line, but to signal the borders of a polygon. Let us assume we have a convex 3D polyhedron to rasterize. This polyhedron is composed of vertices and faces. In a three-dimensional Cartesian coordinate system (X, Y, Z) the faces can be split into two groups. If Z is the characterized as the "height" coordinate, we have the group of the faces looking up and the group of faces looking down with respect to Z. To determine whether a face is looking up or down we simply test the sign of the dot product $\overrightarrow{N}.\overrightarrow{N_z}$, $\overrightarrow{N}$ being the face normal vector and $\overrightarrow{N_z}$ the unitary basis vector in the Z direction. Each group of faces in now orthogonally projected onto two planes. The faces looking up are projected on the plane $Z = Z_{top}$, the ones looking down on $Z = Z_{bottom}$.

Figure A.3: The faces of the polyhedron are split into two groups and projected on the corresponding plane.

From this point on, the two planes are processed separately, but the process is the same for both planes. Our purpose is to determine the extension of the polyhedron in all 3 dimensions. To do so, we process couples of vertices. Let us run the algorithm on the vertices v_1 and v_2 in A.3. v_1 and v_2 form a border of the projected polyhedron on Z_{bottom} and/or Z_{top}. Using the Bresenham algorithm we can determine for each Y between Y_{v_1} and Y_{v_2} the X coordinate border of the projected polyhedron. In our example, the detected X coordinate represents the left border, its coordinate will be stored in a table called X_{left} that records the X positions of the left border for each Y. The equivalent processing is done for the right border with the table X_{right} if the vertices v_1 and v_2 correspond to the right border of the projected polygon. This process is executed for every relevant pair of vertices and after that, at a given Y, $[X_{left}(Y), X_{right}(Y)]$ is meant to represents the X-extension of the polyhedron on the projection plane.

Between $X_{left}(Y)$ and $X_{right}(Y)$ on a line of constant Y , the Z-extension of the 3D volume is to be determined using the Bresenham method once again. The position of the border in the Z direction can first be determined between two vertices v_1 and v_2 exactly like we did for X. Whether the found Z coordinate represent the lower or upper border of the polyhedron depends on which plane (Z_{bottom} and Z_{top}) we currently operate. After this step the lower and upper limits of the polyhedron are known only on the edges of the polygon. A last pass of the Bresenham algorithm is thus needed to interpolate the coordinates of the lower and upper border of the volume over the whole projected polygon.

At that point, for every point (X, Y) within the $X-$and Y-extension of the volume, we have the coordinates Z_{down} and Z_{up} of the lower and upper borders of the volume in the

Z direction. We know only need to list the voxels placed within the detected borders. This can be easily described by a pseudo-code program:

```
for Y between Y_min and Y_max
    for X between X_min[Y] and X_max[Y]
        Z_top ← Z_topplane[X, Y]
        Z_bottom ← Z_bottomplane[X, Y]
        for Z between Z_bottom and Z_top
            process voxel (X, Y, Z)
        endfor
    endfor
endfor
```

The "process voxel" routine is generic and can be configured to do whatever the user wants to do with the rasterized geometrical information. As it has been said in A.1, the rasterizer's contribution is the corner stone of several processes presented in this report, including the mesh relaxation scheme, the color fitting algorithm as well as the SSD and PSNR measurements.

Appendix B

Notation and Abbreviations

bpp: bit per pixel.

CVT: Centroidal Voronoi Tesselation.

dB: decibel.

DCT: Discrete Cosine Transform.

DWT: Discrete Wavelet Transform.

GIF: Graphics Interchange Format.

GPU: Graphical Processing Unit.

IRLS: Iteratively Reweighted Least Squares.

JBIG: Joint Bi-level Image experts Group.

JPEG: Joint Photographic Experts Group.

LMS: Least Median of Squares.

LS: Least Squares.

MPEG: Motion Picture Experts Group.

NPR: Non-Photorealistic Rendering.

ODT: Optimal Delaunay Triangulation.

PSNR: Peak Signal to Noise Ratio.

RANSAC: RANdom SAmple Consensus.

SSD: Sum of Squared Differences.

VQ: Vector Quantization.

WLS: Weighted Least Squares.

List of Figures

Bibliography

[ACSYD05] P. Alliez, D. Cohen-Steiner, M. Yvinec, and M. Desbrun. Variational tetrahedral meshing. *Proceedings of ACM SIGGRAPH 2005*, 24:pp 617–625, 2005.

[Bre65] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4, 1965. useful information can also be found here : `http://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm`.

[Cla95] D. Clark. The popularity algorithm. *Dr. Dobb's journalh*, 1995.

[COLR99] D. Cohen-Or, D. Levin, and O. Remez. Progressive compression of arbitrary triangular meshes. *IEEE Visualization Proceedings*, 1999.

[CPM05] N. Cammas, S. Pateux, and L. Morin. Video coding using non-manifold mesh. *Proceedings of the 13th European Signal Processing Conference*, 2005.

[CRH05] J. P. Collomosse, D. Rowntree, and P. M. Hall. Stroke surfaces: Temporally coherent artistic animations from video. *IEEE Transactions on visualization and computer graphics*, 2005.

[CSAD04] D. Cohen-Steiner, P. Alliez, and M. Desbrun. Variational shape approximation. *International conference on computer graphics ans interactive techniques. ACM SIGGRAPH papers*, 2004.

[Cse] D. Csetverikov. Basic algorithms for digital image analysis. *Institute of Informatics, Eötvös Loràand University, Budapest Hungary*. http://visual.ipan.sztaki.hu.

[CX04] L. Chen and J. Xu. Optimal delaunay triangulations. *Journal of computational mathematics 22*, 2:pp 299 – 308, 2004.

[Dek94] A. H. Dekker. Kohonen neural networks for optimal colour quantization. *Network: Communication in neural systems*, 5:351–367, 1994.

[DG00] O. Devillers and P.M. Gandoin. Geometric compression for interactive transmission. *Proceedings of the conference on Visualization*, 2000.

[DS02] D. DeCarlo and A. Santella. Stylization and abstraction of photographs. *SIGGRAPH proceedings*, 2002.

[GCS02] B. Gooch, G. Coombe, and P. Shirley. Artistic vision: Painterly rendering using computer vision techniques. *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, 2002.

[GP90] M. Gervautz and W. Purgathofer. *Graphics Gems I*, chapter A simple method for color quantization : Octree quantization. Acad. Press, 1990.

[Hae90] P. Haeberli. Paint by numbers: abstract image representations. *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages pp 207–214, 1990.

[Her01] A. Hertzmann. Paint by relaxation. *Computer Graphics International Proceedings*, pages pp 47–54, 2001.

[ILGS06] M. Isenburg, P. Lindstrom, S. Gumhold, and J. Schewchuk. Streaming compression of tetrahedral volume meshes. *Graphics Interface Proceedings*, 2006.

[Krä95] J. Krämer. Delaunay-triangulierung in zwei und drei dimensionen. Master's thesis, Ebehard-Karls-Universität Tübingen, Germany, 1995.

[Kru94] A. Kruger. Median-cut color quantization. *Dr. Dobb's journal*, 1994.

[KYO00] H. Kasuga, H. Yamamoto, and M. Okamoto. Color quantization using the fast k-means algorith. *Systems and Computers in Japan*, 2000.

[Lit97] P. Litwinowicz. Processing images and video for an impressionist effect. *International Conference on Computer Graphics and Interactive Techniques*, pages pp 407–414, 1997.

[LL06] G. Lecot and B. Levy. Ardeco : Automatic region detection and conversion. *Eurographics Symposium on Rendering*, 2006.

[Llo82] S. P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28:pp 129–138, 1982.

[LLS99] P. Lechat, N. Laurent, and H. Sanson. Scalable image coding with fine granularity based on hierarchical mesh. *Visual communications and image processing (San Jose, USA)*, 3653 (2):pp 1130–1142, 1999.

[MPL00a] G. Marquant, S. Pateux, and C. Labit. Mesh-based scalable image coding with rate-distortion optimization. *Image and video communications and processing (San Jose, USA)*, 2000.

[MPL00b] G. Marquant, S. Pateux, and C. Labit. Mesh-based scalable video coding with rate-distortion optimization. *Visual Communications and Image Processing (San Jose, USA)*, 2000.

[PM05] E. Le Pennec and S. Mallat. Sparse geometric image representations with bandelets. *IEEE Transactions on Image Processing*, 14:pp 423– 438, 2005.

[RT99] S. Ray and R. H. Turi. Determination of number of clusters in k-means clustering and application in colour image segmentation. *Proceedings of the 4th International Conference on Advances in Pattern Recognition and Digital Techniques*, 1999.

[Sch97] J. R. Schewchuk. *Delaunay Refinement Mesh Generation*. PhD thesis, School of Computer Science, Computer Science Department, Carnegie Mellon University, Pittsburgh, 1997.

[SLWS07] J. Sun, L. Liang, F. Wen, and H.-Y. Shum. Image vectorization using optimized gradient meshes. *SIGGRAPH proceedings*, 2007.

[SP06] S. Swaminarayan and L. Prasad. Rapid automated polygnal image decomposition. *35th applied imagery and pattern recognition workshop*, pages pp 28–33, 2006.

[TG98] C. Touma and C. Gotsman. Triangle mesh compression. *Graphics Interface Proceedings*, 1998.

[TR96] G. Taubin and J. Rossignac. Geometric compression through topological surgery. *ACM Transactions on Graphics*, 17, 1996.

[VB95] O. Verevka and J. W. Buchanan. Local k-means algorithm for colour image quantization. *Proceedings of Graphics Interface*, 1995.

[VGS97] L. Velho, J. Gomes, and M.V.R. Sobreiro. Color image quantization by pairwise clustering. *10th Brazilian symposium on computer graphics and image processing proc.*, 1997.

[WXSC04] J. Wang, Y. Xu, H.-Y. Shum, and M. F. Cohen. Video tooning. *SIGGRAPH proceedings*, 2004.

[Zha96] Z. Zhang. Tutorial on parameter estimation techniques. 1996. `www-sop.inria.fr/robotvis/personnel/zzhang/Publis/Tutorial-Estim/Main.html`.