Modeles de representation et d'acquisition pour le rendu expressif

Pascal Barla

▶ **To cite this version:**

Pascal Barla. Modeles de representation et d'acquisition pour le rendu expressif. Interface homme-machine [cs.HC]. Institut National Polytechnique de Grenoble - INPG, 2006. Français. NNT : . tel-00116849v2

HAL Id: tel-00116849
https://theses.hal.science/tel-00116849v2

Submitted on 22 Oct 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Institut National Polytechnique de Grenoble

Representation and acquisition models for expressive rendering

Pascal BARLA

Thesis presented for the obtention of the title of Doctor of INPG, in computer science.
Prepared in the ARTIS-GRAVIR/IMAG-INRIA laboratory, UMR CNRS C5527.
Defended on November 10th, 2006.

Jury members:

Marie-Paule CANI	–	president
John F. HUGHES	–	reviewer
Pascal GUITTON	–	reviewer
Lee MARKOSIAN	–	examinator
Joëlle THOLLOT	–	co-advisor
François X. SILLION	–	advisor

Acknowledgements

Many people contributed in a direct or indirect way to this thesis. I would first like to thank my advisers Joëlle Thollot and François Sillion for their availability, advice, and sympathy. Through the three years of my PhD, they always gave a lot of interest to my work and have been of a crucial help. I would also like to thank people from the ARTIS and EVASION teams (INRIA Rhone-Alpes, France) for their opinions, suggestions and theoretical as well as practical help. Other people at the INRIA Rhone-Alpes were of good advice: Peter Sturm from the PERCEPTION team for discussions on computer vision and Pierre Bessière from the EMOTION team for discussions on bayesian theory.

This thesis is also the result of collaborations within the French computer graphics community. I would thus like to thank George Drettakis and the REVES team (INRIA Sophia Antipolis, France) for the collaboration in the ARCHEOS project (involving expressive rendering and archaeology); Gwenola Thomas and the IPARLA team (University of Bordeaux I, France) for the collaboration in the MIRO project (involving expressive rendering and legible representations); And Bernard Peroche (Universite Lyon I, France) for managing the creation of a French book on computer graphics (currently in the process of publication) including a chapter on expressive rendering, authored by Joëlle Thollot, Gwenola Thomas and I.

An important time in my PhD have been my EURODOC internship at the University of Michigan (Ann Arbor, USA). I would like to thank Lee Markosian, who was my adviser there, for the work we've done together during these 7 months; and also Nathan Lezotte and Simon Breslav for their active collaboration and technical contributions; Igor Guskov for his suggestions; John Hughes for presenting me to Lee Markosian in the first place; and to both of them, for accepting to come to France and be part of my thesis jury. Other collaborations have been of a great help during the course of the PhD. In this respect, I would like to thank Fredo Durand for hosting me for a month in MIT (Boston, USA), and for the many discussions and bibliographic help (mostly by email and with a quasi-instantaneous feedback). Sylvain Paris, also at MIT at this time, have also been of good advice and gave me clever suggestions both on practical aspects of the PhD and scientific discussions. Both Sylvain and Fredo introduced me to Edward Adelson, with whom I had recent discussions on visual perception, and who inspired me consequently for the discussion on the role of human vision in expressive rendering. Victor Ostromoukhov, from the University of Montreal (Quebec, Canada), has been an inspiration for topics on synthesis. Valerie Bonnardel, from the University of Sunderland (England), helped me better understand the links between color categorization and linguistics.

Finally, I'll be eternally thankful to my family and friends, for their help, support, care, comprehensiveness, and ability to release my stress in the sometimes tense episodes of the PhD.

Introduction

Expressive rendering is a field that emerged around the beginning of the 90s as an alternative to the common trend in computer graphics to produce ever more realistic renderings. By producing "non-photorealistic" imagery, researchers tried to mimic the way artists create paintings and drawings, in order to capture many of these qualities that differentiate them from photographs. The field grew consequently and nowadays it is an inherent part of computer graphics, with expressive rendering papers published in the most important computer graphics conferences and journals, and even a dedicated biennial symposium since 2000, the International Symposium on Non-Photorealistic Animation and Rendering (NPAR).

Besides this sudden growth of interest, expressive rendering is still a young research field and naturally, most of the previous work proposed specific solutions to specific problems, nearly always imitating traditional techniques. This thesis proposes to investigate expressive rendering from a more principled point of view, through two main issues which will be presented below: *representation*, or how to balance user interaction and computer automation to give an intended style to a representation; and *acquisition*, or how to analyse stylistic information input by a user in order to apply it elsewhere in a representation.

Before going into greater detail about expressive rendering, we should first direct our attention to the notion of "realism", from which it tries to depart. To this end, we briefly consider, in the next section, how the quest of realism evolved in art history, before relating expressive rendering to this evolution. We also give here a first glimpse at the connections that tie human vision to any expressive representation, which will be an underlying theme. Finally, we give an overview of the different parts that constitute this thesis. A deeper description of traditional techniques and related work in expressive rendering can be found in each part.

Realism, photorealism and "non"-photorealism

Realism, through the imitation of nature, has been a goal for many artists in paintings and drawings from the Renaissance to the 19th century. With the apparition of photography, though, traditional image creation shifted towards more abstracted or expressive representations. The advantages of these alternative depictions compared to photography are many: by intentionally omitting or abstracting some elements of a depicted scene, and by placing more emphasis on others, artists are able to efficiently control the visual communication of information. This is the reason why paintings and drawings are still used in place of photographs in fields such as medical, technical or archaeological illustration; but also in other media such as traditional animation or comics, for their evocative power.

To give an extremely rough outline of the evolution of art through history, one can say that the quest of realism (at least in the western culture) has been of main interest through the Renaissance, and culminated with Impressionism, the ultimate study of light and color. Interestingly, Impressionism is also often considered as the first modern movement. Soon after came the “explosion”: abstraction took a more and more important place in art. As McCloud [McCloud 94] points out, “The main trust was a return to meaning in art, away from resemblance, back to the realm of ideas.” There is indeed a very broad distinction between pictorial representations, in which shape and space can be seen in the picture; and pure pictorial abstractions. In this thesis we will only consider representations of a possible scene, real or imaginary.

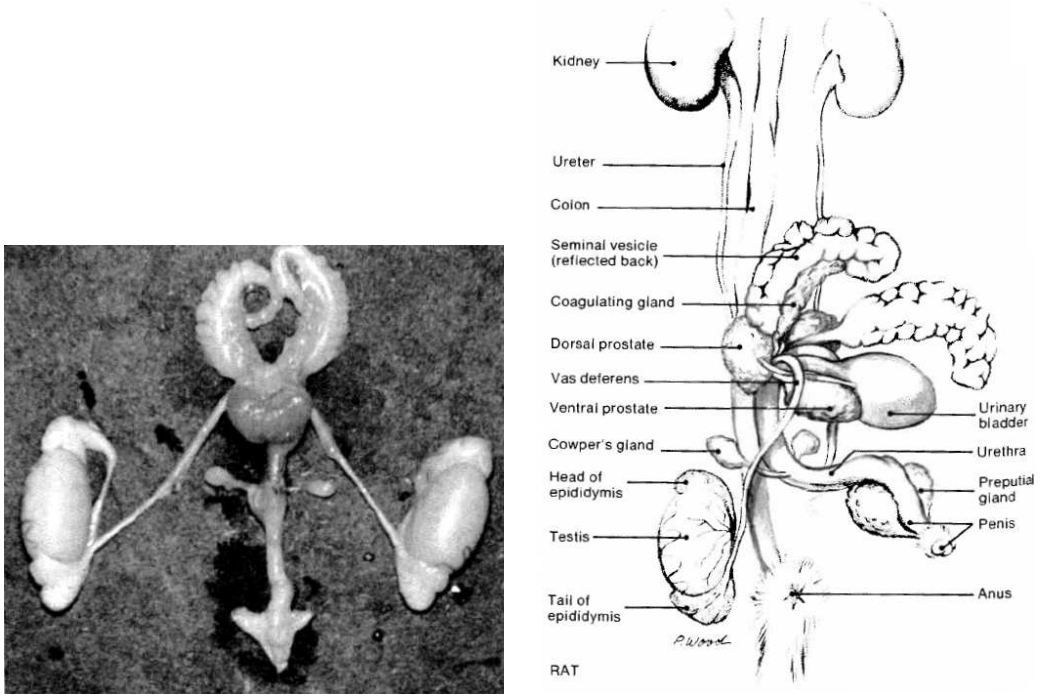
There is no need for a pure abstraction, however, to produce expressive pictures. Even Photorealism is prone to expressivity. As noted Durand [Durand 02], the photographer is never passive towards light. The choice of the viewpoint and perspective plays an important role in photo production. The photographer may use additional artificial light sources to make the scene "look real", or the lighting from different viewpoints may be changed to "look unchanged". So the recording is realistic, but the light configuration may be artificial. Moreover, reflectance of surfaces may be modified in an analogous way, for example by using makeup for faces. Finally, the printing process is never a passive reproduction of the negative, and many “post-processing” operations can be applied at this stage to yield a dramatically different result. If one considers that chemical photography is only a new photorealistic technique, then Photorealism itself is only one pictorial style from many.

However, this photorealistic style, like any other, has limitations. A convincing example is given by scientific illustration [Wood 94]. Photography records a subject accurately while the illustrator draws only what is necessary (see Figure 1). The illustrator can omit extraneous detail - clarifying, emphasizing, and selecting the important parts. The artist can simplify or summarize the essence of a subject and can reveal what is underneath or inside. He can idealize, ignoring specimen variations. In other words, while the camera establishes and documents the existence of a subject, the illustrator illuminates its essence: an illustrator must see not just an individual specimen but one specimen in relation to many species in the broader context of an entire field.

Definition of expressive rendering

Computer graphics evolved in a way quite similar to traditional techniques: the quest for realism has been a main challenge with the simulation of light transport and realistic materials; but now that these goals seem within reach, new rendering techniques are introduced that take inspiration from traditional drawing and painting techniques [Strothotte 02, Gooch 01]. This domain has received a lot of interest in the last 15 years under different names such as non-photo-realistic rendering (NPR), expressive rendering, or more generally computer depiction [Durand 02].

The term “NPR” has been widely adopted by the community. However, as noted by Schofield in his Thesis [Schofield 93], NPR is any computer graphics rendering system which gets away from Photorealism. It is defined by an absence of something. At the same time, Photorealism in Computer Graphics is often departing from reality: the addition of depth of field, lens flares, motion blur and grain are all wanted



(a) (b)

Figure 1: (a) A photograph records a subject without emphasis or omission. (b) A drawing can selectively interpret, emphasizing and creating perfect background and lighting [Wood 94].

artifacts that do not belong to reality.

Thus we need a better definition for NPR. As Photorealism is a trace or print of reality, NPR should be a personal, artistic interpretation of reality. Schofield [Schofield 93] proposes a criterion of success for a NPR algorithm: the degree of significance achieved between the thing represented and the way it is represented. We build on this definition, and in order to reflect the importance of the role played by the user in the rendering process, we use the term "expressive rendering" in place of "NPR". We then give the following definition: *expressive rendering is any rendering technique that allows the user to be part of the way scene features are represented; its criterion of success is the degree of significance achieved between the depicted scene and how close its representation comes to the user's initial intention.*

Expressive rendering has of course many common points with traditional techniques. It shares the same objectives, independent of any medium: what are the important scene features to emphasize, and what are the features that we will prefer to abstract, maybe even omit? Every new medium must go through a phase of imitating a predecessor [Schofield 93, McCloud 94, Durand 02]. Therefore expressive rendering should inspire from paintings and drawings, even photography and cinematography, to create images more functional, more aesthetic and more suitable to our needs.

However, simulation of paintings and drawings in expressive rendering is unnecessary, because this is the simulation of something absent (the canvas, the chemicals, etc...) [Schofield 93]. Similarities between expressive rendering and paintings and drawings should be incidental. So expressive rendering should convey qualities which transcend any particular medium, or transfer these qualities from paintings

and drawings. Schofield takes the example of displaying a sentence with an editing software: the linguistic content is maintained intact, but no simulation of the writing process is actually performed.

On the other hand, using a computer offers many new possibilities: expressive rendering systems can assist an artist in producing more easily and more quickly a picture or an animation via automatic or semi-automatic tools; it allows to better control the evolution of a representation throughout an animation (we speak of temporal coherence); it facilitates the reproduction of a work by storing the various steps of a composition on a digital device; it allows interactive animations, almost absent from traditional media; etc. These many advantages make expressive rendering a very useful tool for many domains such as animation, scientific illustration, archaeology, architecture, technical drawing, video games, advertisement, etc.

Even more important is the freedom allowed by expressive rendering: any traditional medium has intrinsic limitations due to its physical implementation (the diffusion of paint, the density of the canvas and the tools used by the artist); but there is no such a limitation when using a computer. While it virtually offers an infinite amount of expressive ways to create a representation, it is the responsibility of the programmer who designs the software to give intuitive, easy-to-use interfaces, unless the advantages of using a computer become compomised. Thus, in our definition, we must add as a quality requirement that *expressive rendering should offer the user intuitive and ergonomic interfaces to specify the style of a representation.*

Art and illusion

To summarize, in order to produce meaningful expressive renderings, we need to find a way to describe what is the essence of a representation, be it a painting, a drawing or a photograph. We previously used the term *style* to this end, as it is often done in art history for instance. Gombrich, a famous art historian, has been one of the first authors to study this abstract notion. The following ideas come from his book, "Art and Illusion" [Gombrich 61].

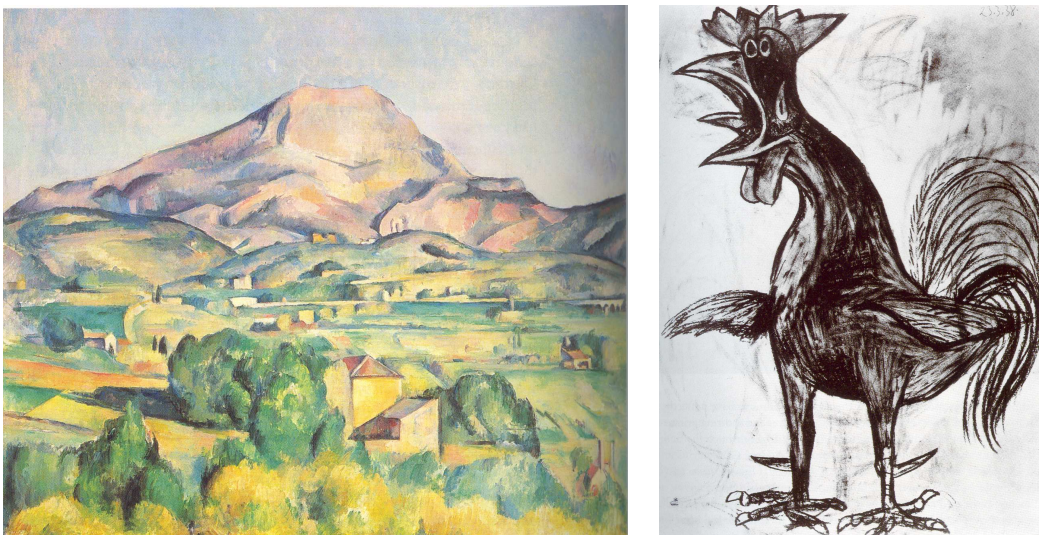
An artist looks for some aspects of his environment that he will be able to reproduce. However, he does not paint from a pure visual impression, but more from an idea or concept (as Gombrich says, "the artist is more prone to see what he paints, rather than painting what he sees"). Thus, he needs a vocabulary before trying to represent his environment. The particular choice he makes of this vocabulary and the way he combines various elements constitute the style of a representation.

The human visual system is particularly well suited to build such a vocabulary: we naturally distinguish peculiarities, we perceive organic ensembles where there was before only a meaningless ensemble (think for instance of the familiar images one can see when looking at clouds in the sky). By manipulating this natural tendency, the artist is able to play with the observer, with his ability to see in a representation some objects or images that have been captured by his memory. Therefore, a painting, while it is nothing more than a set of paint layers on a canvas, will inevitably produce in an observer's mind the perception of a scene. It is thus a mere *illusion*.

The body of techniques invented and learnt to produce such illusions are called *schema* by Gombrich. Schema evolved greatly through art history, and constitute the basic building blocks of a style vocabulary. They can thus be considered as low-level

techniques that, when properly combined, lead to a particular style. This is why the evolution of styles is coherent with time: the addition of new schema and their various combinations led to various ways to express an idea or concept through the representation of a scene.

The conditions of illusion thus reside in the knowledge of the language used by the artist and in the conditions in which it has been used. In order to make his idea or concept clearer, the artist can avoid any useless detail, because he knows his public will be able to understand his suggestions. Indeed, we are so efficient in recognizing familiar objects that we rarely fail at performing this task; And at the same time, most of us are unaware of the conditions of that illusion. For instance, Cezanne as well as Picasso made use of very geometric, simple forms to sharpen their representations and evoke more of the essence of their subject, as illustrated in Figure 2; However, besides this high degree of abstraction, any observer perceives with no difficulty a landscape with a mountain and a rooster, respectively.



(a) (b)

Figure 2: (a) “La montagne Sainte Victoire” by P. Cezanne. (b) “Cockerel” by Picasso.

In order to really understand what is the essence of traditional art, we need to unravel the mechanisms of that illusion, at least at the low levels of its functioning. This means to understand schema and the reason why they trigger so efficiently the human visual system in traditional paintings and drawings. It will not only allow us to analyse existing pieces of artwork in order to *acquire* some of their important stylistic properties, but also to create expressive *representations* possessing similar qualities.

Overview

This thesis is structured along two research axes: First, in Part I, we investigate the representation issues raised by the production of expressive images and animations; Second, in Parts II and III, we study the acquisition of low-level notion of styles from vector-based and bitmap color images respectively.

In addition, I have been personally motivated by the fascinating field of human vision and its many links with expressive rendering. Thus, throughout this thesis, I

tried to build as many connections as possible between the two fields. However, many more connections are waiting to be discovered, and I hope this work will motivate others to investigate them.

More precisely, Part I of this thesis focuses on expressive animation metaphors. Although it mainly treats animations produced from 3D scenes, a theoretical foundation for images as well as animations, greatly inspired from the work of Willats and Durand [Willats 97, Durand 00, Durand 02, Willats 05], is presented in its introduction. This helps to situate our contributions to expressive rendering. The part is closed by a discussion on the the importance of visual perception as a significance measure for expressive animations.

Then Part II addresses an inverse, yet complementary problem: the analysis and synthesis of line drawings. As opposed to the previous part, the input data here is composed of 2D vector drawings. We propose two different tools to process line drawings (simplification and synthesis). We finally open the problem to future work by discussing the perceptual cues potentially revealed by contour lines and line patterns, and their perceptual segregation in a drawing.

Part III has a rather similar goal to Part II: the analysis of an input representation in order to acquire some elements of style. However, this time, the focus is on picture color composition. The input data is thus a bitmap color image, and we propose a colorimetric analysis that gives a categorical description of color composition. This part is closed by a discussion on the connections between categories and language, and the importance of the spatial perception of colors.

Finally, the representation and acquisition issues we presented, and to which we gave some solutions, are put together in Part IV, where we insist on the role of vision in expressive rendering techniques. This helps not only to position previous approaches, but also to motivate new ones, and to design evaluation methods for expressive rendering.

Contents

Introduction v

I Expressive animation metaphors 1

1 How to create expressive image sequences ? 3

2 An extended toon shader for attribute mapping 21

3 A painterly rendering metaphor 35

4 An alternative metaphor: dynamic 2D patterns 49

5 Remarks on low-level perceptual criteria 59

II Line drawings analysis and synthesis 65

6 What do line drawings represent ? 67

7 Line drawing simplification 77

8 Stroke pattern synthesis 93

9 Remarks on drawing segregation 107

III Picture color composition 111

10 How are colors distributed in a picture ? 113

11 Color palette extraction 119

12 Remarks on color categorisation 137

IV Closing remarks 141

Discussion on the role of vision in expressive rendering 143

Conclusions 153

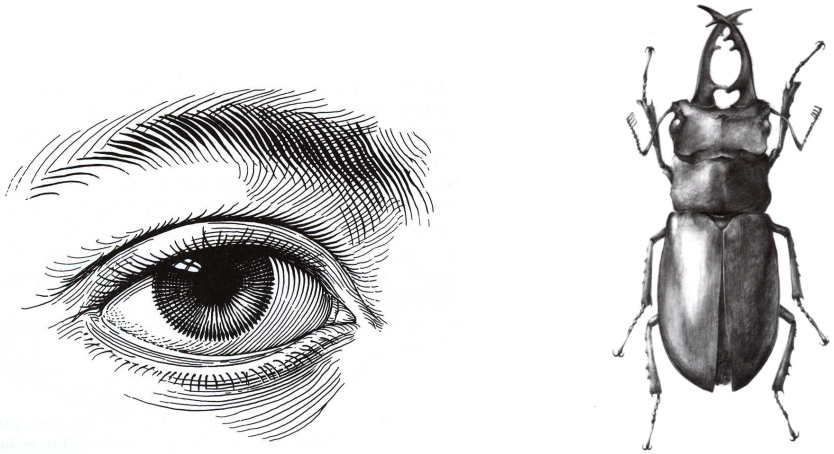
Part I

Expressive animation metaphors

Chapter 1

How to create expressive image sequences ?

As pointed out in the introduction, there are many potential advantages to using a computer to produce expressive images, and the first that comes to mind is animation. The production of image sequences is a tedious and delicate process when done by hand, as explained in the next section. However, inspiring lessons can be drawn from existing traditional techniques.



(a) (b)

Figure 1.1: (a) A drawing made with the use of ink on scratch board, where independent lines are perceived. (b) Smooth continuous tones obtained with a carbon dust technique [Wood 94].

1.1 Examples from hand-made representations

Various representations There exists a vast number of painting and drawing techniques and an exhaustive presentation is, of course, out of the scope of this thesis. However, some general observations can be made. Depending on the medium used, and the way the artist applies it on a canvas, the final representation can be perceived as an *accumulation of independent brush strokes* or a *continuous variation of tone* (see Figure 1.1). This is only a vast simplification, but at least it helps in the

(a) (b) (c) (d)

Figure 1.2: Various oil-painting techniques: (a) blending, (b) rubbing, (c) glaze and (d) impasto.

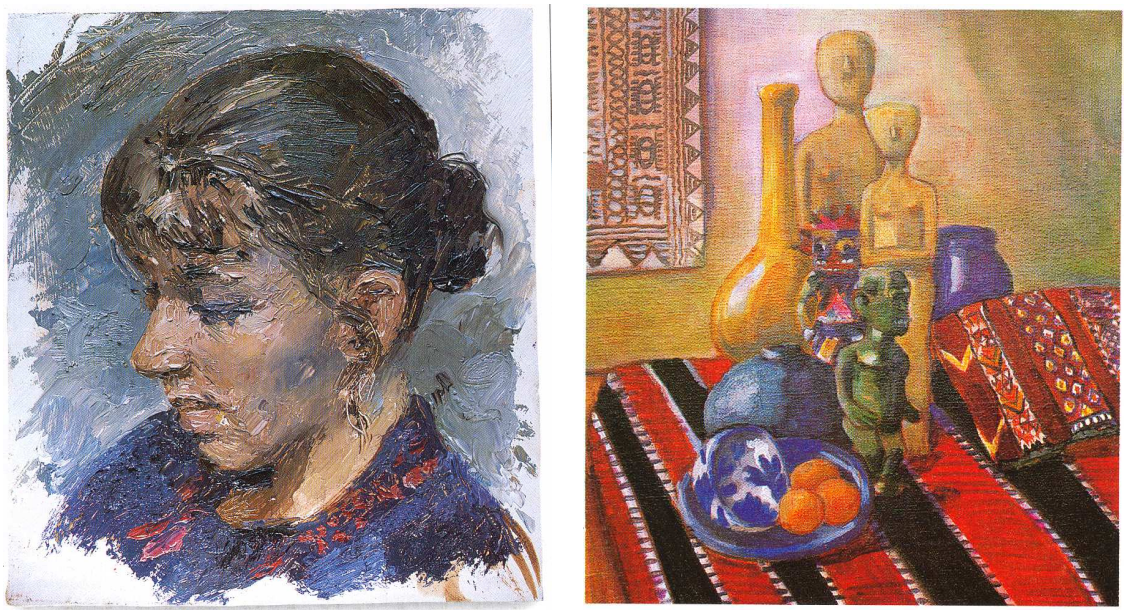
analysis of traditional paintings and drawings. For instance, as prescribed by Wood in her Scientific Illustration book [Wood 94],

> "While the pencil is at its best in rendering detail, the airbrush really shines in producing smoothed and graded areas of tone. These areas are stroke-less and brush-less in quality and may be rendered very quickly".

There are many ways to combine these two extreme strategies, and the choice of a representation will depend on the intention of the artist. Just to give a rough idea, Figure 1.2 shows various methods used to apply oil paint on a canvas. The blending operation (1.2(a)) consists in applying various colored layers of paint and mixing them on the canvas, giving a continuous variation of colors. The rubbing operation (1.2(b)) consists in making a thick amount of paint penetrate the canvas with a hard brush. This technique lets a relief appear on the canvas and spreads colors that fix to ridges of the canvas, but leaves valleys intact. The glaze operation (1.2(c)) allows, by applying a thin layer of transparent paint, to create various effects, like giving a better color harmony, add a subtle relief, etc, without masking underlying colors. The impasto operation (1.2(d)) allows to reflect the gesture of the artist by using a great amount of paint.

Some artists even use mixed representations, e.g. to focus on an important region of the canvas using continuous tone and only suggest unimportant regions with sparse brush strokes. Again, the ways various painting techniques may be mixed are numerous. Figure 1.3 shows two very different approaches. On the left, a knife is used to add relief texture to important regions of the painting, while unimportant ones are roughly painted with sparse brush strokes. While on the right, many layers using different techniques are used and then superimposed, with distinct brush strokes added to represent highlights and fabrics patterns, in order to focus on the material properties of the depicted objects.

Moreover, strokes can be organised into groups, like hatching and stippling groups found in some illustrations or comics (see Figure 1.4). Then, the strokes are not only perceived as independent elements, but also as part of a higher-level distribution.



(a) (b)

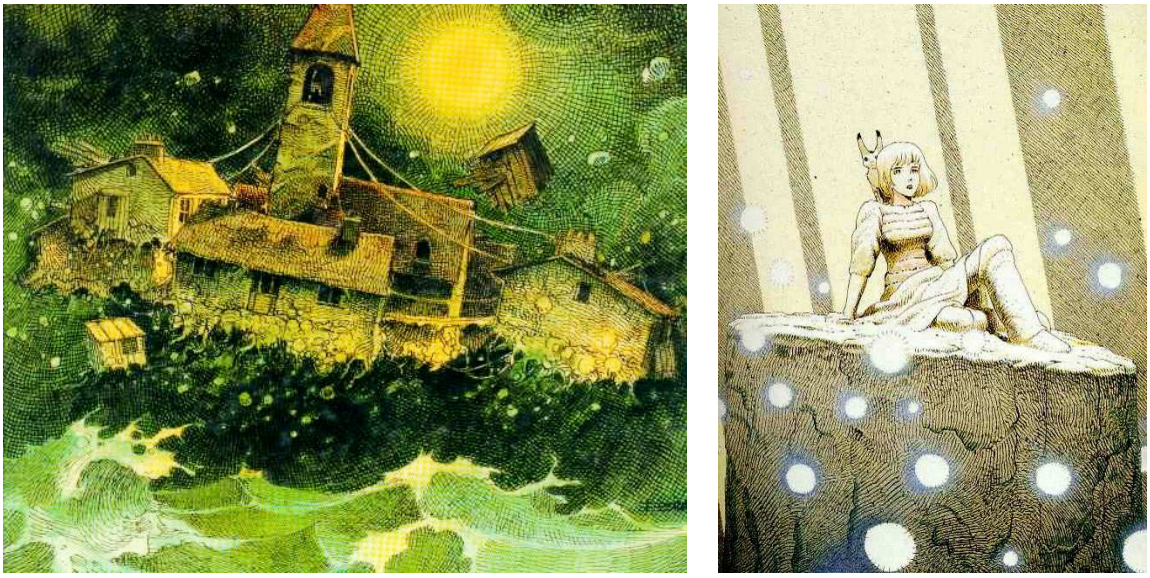
Figure 1.3: Different focus techniques: (a) knife and (b) superimposition of layers.

How the medium evolves in an animation When an animation is made by drawing or painting successive frames, the question of how the medium moves from frame to frame arises.

In the case of continuous tone, if the chosen medium exhibits a small-scale texture (like with the pigments of watercolor), then successive frames will produce incontrollable small-scale *flickering*: the paint being applied for each new frame, the texture will inevitably be present at different locations in two successive frames. To avoid this problem, some artists use more homogeneous media and fine grained canvases (see Figure 1.5(a)).

In the case of brush strokes perceived more or less individually, then one might want to make strokes of the current frame correspond to strokes of the previous frame. This is however a tedious process, and a very small deviation will appear as a *jittering* in the final animation. Moreover, strokes will appear and disappear from frame to frame due to occlusions in the depicted scene, introducing *poppings* in the animation. For stroke groups, the motion is even more problematic since not only the strokes have to be coherent over time, but their distribution also need to be controlled.

Flickering, jitterings and poppings are often referred as *temporal coherence* issues in the expressive rendering literature. They are sometimes so hard to avoid that most of the artists in traditional animation exploit the temporal coherence problems in artistic ways: they add independent jittering motion to the strokes so that any observer will not even be tempted to look for coherence, and by the way they add vitality to the animation even on a static sequence (i.e. the objects in the depicted scene are static, but the paint strokes used to represent them move arbitrarily). This is an example of how the limitations of the medium (in this case, the difficulty of ensuring temporal coherence by hand) leads to creative solutions, as in the work of Georges Schwizgebel (see Figure 1.5(b)).



(a) (b)

Figure 1.4: Two examples of stroke patterns found in comics illustrations. (a) “La croisière des oubliés”©by Enki Bilal (b) “Nausicaa”©by Moebius.

What is depicted in a representation The effectiveness of a hand-made representation not only consists in the mastery of a medium; It first resides in the way specific elements of a depicted scene are represented, i.e. some may be abstracted or ignored, while others will be emphasized.

There are multiple ways of applying brush strokes. For instance, in order to clearly emphasize the constituting elements of a scene, their spatial extent as well as their color should follow the shape and color of an element. Figure 1.6(a) illustrates the use of brush strokes of multiple size, shape and color to represent adequately a flower bouquet in its vase: Leaves are painted using one or two large brush strokes in various shades of green in order to represent the light reflecting from them; Small paint stipples are painted to suggest flower pistil; Curved strokes roughly follow the texture on the vase; A soft wash is applied to add a soft shadow; etc. So the “paint” on the canvas can represent shape, texture, patterns of small elements or illumination.

Note however that the choice of a medium is quite independent from the choice of depicted scene elements: The previous example, made using watercolor, could have been performed using oil paint or pastels. The choice of the medium constraints the artist though, leading to new creative techniques. For instance, the use of a dry medium like pastels seems to make it inappropriate to depict translucent objects that exhibit very smooth variations of tone. However, as shown in Figure 1.6(b), the transparency of objects like the bottle or the glasses can be very efficiently represented with pastels by taking a particular care in drawing their shadows: a ray of light in the bottle’s shadow for example suggests its material properties. This is an example of a *schema* (see the introduction): a technique learnt by artists in order to better depict a scene, that gives at the same time some style in the composition.

Other schema aim at representing more global effects in a scene, for example showing the undulations of a water surface, or increasing the perception of depth in an outdoor scene via aerial perspective. Figure 1.7(a) shows how stripe-like brush strokes are used to render the fragmented reflections of an undulating water surface.

(a) (b)

Figure 1.5: (a) “Feeling from Mountain and Water”©watercolor animation by Te Wei. (b) “Man without a shadow”©painterly animation by Georges Schwizgebel.

Figure 1.7(b) illustrates the effects of aerial perspective on a monochrome painting: Depth is suggested by variations in tone and contrast. In general, it can also be suggested by color “temperature” variations or a loss of detail.

Finally, in the context of animations, the motion of the medium on the canvas, especially for individually perceived brush strokes, reveals some motion in the scene, and here again the effectiveness of a representation resides in emphasis and/or abstraction. There are various possibilities that have been employed by different artists. In the oil-painted animations of Alexander Petrov, individual brush strokes seem to approximately follow the motion of objects in the scene. While for the watercolor animations of Te Wei, washes of paint seem to follow self-shadowing effects and thus slide on the characters. More complex medium motion can be performed though, like in the work of Georges Schwizgebel, where it is used to morph between different scenes and contributes to the illusion of a continuous motion, which is an example of a motion schema often used in traditional animation.

Technical difficulties of hand-made representations Traditional techniques impose a lot of technical issues: Mainly, the coherence issue is merely ensured in hand-painted animations. Moreover, it is an extremely time consuming process to paint successive frames of an animation. Just to give an example, Alexander Petrov took two and a half years to complete “The old man and the sea”, a 20 minutes painterly animation. Another issue of painterly animations is that there is little or no previewing possibilities. Petrov uses a paint on glass technique that allows him to visualize coarse paintings of a sequence, then he adds fine details on each frame independently (see Figures 1.8). This is another example where a very constrained medium forces the artist to find tricks, here to get a preview of the final animation. Even for a still image, it is hard to master painting with various mediums and to manage the constraints they impose.

However, these constraints are not present anymore when using a computer to create expressive rendered images or animations. So there is no need to simulate them, as advocated by Schofield in his Thesis [Schofield 93]. Rather, expressive rendering should take inspiration from traditional techniques, and extract their essence in order to provide tools that hold similar expressive qualities. Expressive rendering

(a) (b)

Figure 1.6: (a) Brush strokes of various sizes are used to depict different elements. (b) Translucency of objects is suggested by the representation of their shadows.

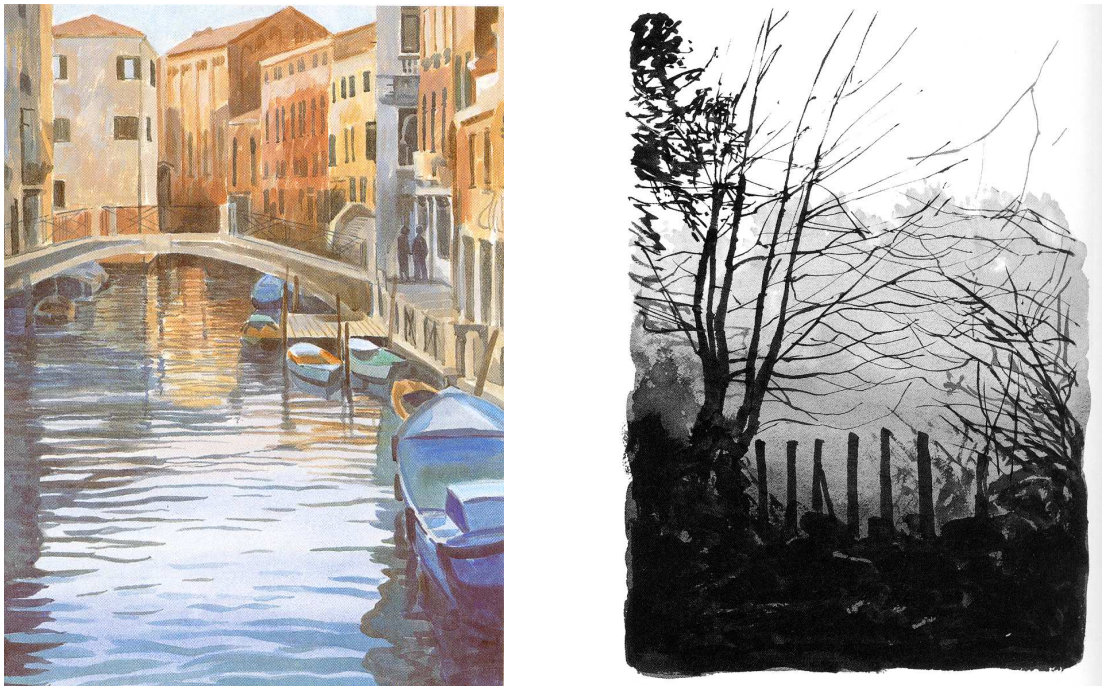
techniques should thus be able to give the artist what he can only rarely obtain with traditional techniques: the control of temporal coherence, automated tasks to speed up the creation process, instant previews of the final rendering, and easy manipulations that do not require a tremendous amount of training.

1.2 A model of representation

In order to "extract the essence" of traditional techniques in creating expressive renderings, we need a way to relate an image created by hand with another created with a computer. Actually, there exists a theoretical model of pictorial representation, that aims at describing the systems involved in the creation of a representation **regardless of the medium employed**. The first version of the model has been proposed in a book by Willats [Willats 97], then refined by Durand [Durand 00, Durand 02] and Willats & Durand [Willats 05]. It will serve us as a basis for relating traditional and expressive rendering techniques throughout the remainder of this thesis.

The initial representation model In his book [Willats 97], Willats presents two systems for describing pictures: the *drawing* and *denotation* systems. Together, they provide a way of classifying a very wide range of pictures and therefore form a basis for describing at least part of what we mean by pictorial style.

Drawing systems are mainly concerned with projections or topological views: they map spatial relations in the scene (primary geometry) into spatial relations in the picture (secondary geometry). Denotation systems, on the other hand, relate the scene and its representation by mapping scene primitives (volumes, surfaces, etc.) into picture primitives (points, lines and regions). A primitive is the most elementary unit of shape information.



(a) (b)

Figure 1.7: (a) Water reflections are depicted by horizontal brush strokes of alternating color. (b) Aerial perspective uses contrast and tone variations to emphasize depth.

Picture primitives and scene primitives are abstract concepts. In practice, picture primitives are represented by physical marks. This distinction is analogous to that made in language between the smallest units of meaning — phonemes and morphemes — and their realization in physical sounds or letters.

The choice of a given drawing or denotation system depends on the function it will serve. Different applications lead to different functions: for example, in engineering the true scale of objects is important, so orthogonal projections are chosen for the drawing system; whereas in cubist paintings, one of the goals is to flatten the picture surface, so a combination of object-centered orthogonal projections, together with a view-centered distorted perspective (naive, inverted or oblique) are employed. In the same idea, anomalous denotation systems, like anomalous line junctions or reversal of tone on objects can be employed for expressive purposes.

This first version of the representation model is rather vague and general, but it has the merits of giving a comprehensible common model to any medium, without focusing on the final implementation of picture primitives by marks. For instance, a photo-realistic representation is one that combines a perspective drawing system with a denotation system that maps continuous visible points in the scene to continuous points in the picture. Whether picture points are pixels on a screen or ink blobs on photographic paper or paint stipples on a canvas does not change the very nature of the representation.

The extended representation model Durand [Durand 02] further extended Willats model, by refining it into four systems: the drawing system is renamed the *spatial* system, the denotation system is split into a *primitive* system and an *attribute* sys-

(a) (b)

Figure 1.8: (a) Alexander Petrov and his paint-on-glass technique. (b) A frame from "The old man and the sea"©.

tem, and the *mark* system is explicitly added to the model.

To summarize:

1. The spatial system maps 3D spatial properties onto 2D spatial properties;
2. The primitive system maps primitives in object space (points, lines, surfaces, volumes) to primitives in image space (discrete and continuous points, lines, surfaces);
3. The attribute system assigns visual properties to picture primitives (color, thickness, orientation, etc);
4. The mark system implements the primitives placed at their spatial location with their corresponding attributes.

Therefore, in the extended model, a picture is made up of marks which represent picture primitives with their associated picture attributes; picture primitives and attributes denote corresponding scene primitives and attributes which represent objects. An artist creates pictures through this flow, even if it is done unconsciously. To the contrary, expressive renderings perform the *inverse process*, starting with a description of a 3D scene and seeking for a specific representation.

Establishing a relation between expressive rendering and traditional techniques is then possible using this model of pictorial representation. By analyzing paintings and drawings with respect to each of the four systems, one can factor out essential properties; Then, by constructing expressive rendering pipelines that explicitly possess the same properties in each system, we will have conveyed qualities that transcend any particular medium.

Recall the question we asked in the previous section: "What is depicted in a representation ?". In Willats & Durand model, this is equivalent to characterize what kind of picture and scene primitives a mark implements: Is it a region corresponding to a shadow in the scene? Is it a curve corresponding to a silhouette? Is it a point

corresponding to a small volume from the current view? It also demands to link the color and other attributes of, say, a brush stroke to some attributes in the scene: does color take illumination into account or is it only a crude approximation using a constant color? Finally, what is the spatial layout of the representation: Perspective ? Curvilinear ? Etc.

Motion and temporal coherence Note that the representation model has only focused on the creation of still images. The extension to animation holds, but it raises the important question of temporal coherence that we talked about in the previous section. More technically, can we trace back a temporal coherence problem from the mark system to earlier systems? If we can do so, then we are able to identify the source of temporal incoherence, and hence propose methods to avoid it, or at least attenuate it.

Let first rule out two naive approaches that do ensure temporal coherence but cannot be considered as valid solutions. The first one is "texture mapping": One can directly apply marks onto surfaces so that they behave as textures and are thus perfectly coherent across time. It is straightforward that this method cannot be valid since a crucial property of marks is that they lie in the picture plane, not on surfaces. Indeed, applying this texture mapping leads to a realistic rendering of an object painted in 3D.

The other naive approach consists in placing marks at fixed locations in the picture plane and draw them using attributes coming from the scene. The drawback of this approach is that the corresponding picture primitives are not related to any scene primitive; Thus marks slide on scene objects with motion, leading to a "shower-door effect" that is visually very distracting.

An important observation is that the temporal coherence of marks is directly related to the temporal coherence of corresponding picture primitives. On the other hand, under general conditions, the motion of scene primitives is coherent across time. Therefore, if there is an incoherence to occur in motion, it might be due to the mapping between scene and picture primitives, i.e. the primitive system. The attribute system is usually not prone to temporal incoherence. Indeed, we will see in the next section that the majority of previous approaches is concerned with establishing a temporally coherent primitive mapping.

1.3 Previous work

Expressive rendering has received a strong interest from the Computer Graphics community in the past 15 years, with various metaphors that usually try to mimic traditional techniques. Producing an exhaustive presentation of the domain would result in a huge list of expressive rendering methods, making it hard for the reader to have a coherent and global view of the field. We rather present here what we consider as milestone techniques in each of the four representation systems. However, in the following chapters, we review additional related approaches and discuss limitations in the context of a specific problem statement, in order to compare them to our proposed solutions.

Moreover, since the main subject of this part is animation, we will only focus on systems that render animations. Note that they are in general subject to the same

constraints than with single image renderers, with the addition of temporal coherence. On the other hand, methods that operate on still images cannot be used directly for animations precisely because they will suffer from temporal coherence problems.

Previous work can further be classified based on input data; Whether a full animated 3D scene or simply a video is provided will produce a severe impact on the possibilities offered by an expressive system. Note however that image-space techniques can be considered as special cases of the representation model where projection (spatial system) and shading (attribute system) have already been performed. We will thus in general consider techniques working on video or 3D scenes in a similar way.

(a) (b)

Figure 1.9: (a) An example of multi-perspective projection by [Coleman 04] (b) Automatically generated step-by-step instructions by [Agrawala 03].

Non-linear projections The first step of any renderer is to define how to map from the 3D space of the scene onto the 2D space of the picture. Most of the time, perspective projection is used, but other kinds of projection exist and may be used for artistic or pedagogical reasons in paintings and drawings. The interested reader is referred to Levene's Master Thesis on the subject [Levene 98], where he presents a framework that completely defines a projection by five criteria: The shape of the projection surface; The degree of convergence/divergence of orthogonals to/from a vanishing point; The behavior of orthogonals; A deformation of the "unit cube" (for oblique projections); The subset of objects processed (for multiple projections). Multi-projections have also received a lot of attention in the expressive rendering community. See for instance [Coleman 04, Yu 04, Agrawala 00] (see Figure 1.9(a)).

Another class of non-linear projections concern topological transformations, that essentially map relations between objects or parts in 3D to similar relations in 2D. Agrawala presented interesting applications to the automatic design of route maps [Agrawala 01] and step-by-step assembly instructions [Agrawala 03] (see Figure 1.9(b)).

Expressive shading As we have seen above, the attribute system is responsible for the choice of attributes that will be assigned to the marks of the final representation. Among all the possible attributes, *color* is the single most important one, obviously. In

photo-realistic rendering, the color at a pixel is computed based on the environmental illumination, and on material properties of the depicted object, possibly involving very complex and physically accurate simulations. With expressive rendering, though, such a complexity is usually not necessary, and the need is more towards intuitive tools to control illumination and material properties.

(a) (b)

Figure 1.10: (a) A toon shading example and its 1D toon texture. (b) Tweakable toon highlights by [Anjyo 06].

Early shading models, like the empirical Phong shader, are indeed very intuitive, and this might be why they have been used during so many years. However, the Phong model is rather "realistic". A very famous modification of the Phong model is the toon shader [Lake 00]. The idea is simple: Only consider the diffuse component of the Phong model and use it to index a 1D texture that describes how the color evolves from dark to light tones. By introducing discontinuities in the 1D toon texture (using for instance three stripes of dark, medium and light colors), one can reproduce an effect similar to those found in comics and cartoons, see Figure 1.10(a). Similar approaches have been adopted by Gooch et al. [Gooch 99] and Sloan et al. [Sloan 01]. In the former, the diffuse component is modified in order to create a cool-to-warm gradient, with results similar to those found in technical illustrations; while in the later, the gradient is replaced with a painted environment map.

On the other hand, very few methods considered the specular component of the Phong model. Anjyo et al. [Anjyo 03] proposed a model that manipulates the half vector (used to approximate a reflected light vector) to create plastic or metal materials with a cartoon look. They recently extended their method [Anjyo 06] to make manipulations more intuitive and powerful: the system now allows to manipulate and animate highlights (via key-framing) very easily, see Figure 1.10(b).

Apart from color, other attributes (e.g. orientation, thickness, opacity, etc.) are either set in an empirical way, or follow a picture primitive property. However, there are many avenues for abstracting attributes that will be later used in the extraction or mapping of primitives. For instance, *depth* could be segmented so that subsequent processes using depth will exhibit depth planes.

Image-space primitives extraction In order to go from scene primitives to picture primitives, two alternate methods can be employed. A first category of techniques works purely in the picture plane by extracting 2D picture primitives from the information available at each pixel. The other metaphor, object-space primitive mapping, will be discussed in the next paragraph. The simplest image-space approach is to directly use the *pixel* as a primitive as done by Hertzmann et al. [Haeberli 90, Hertzmann 98] where they distribute strokes that take their color from an underlying video and use randomness for other attributes. Temporal coherence problems are reduced by using strokes of a previous frame in the current frame. But as noted by Schofield [Schofield 93], this approach does not rely on any arbitrariness; And in order to articulate meaning, marks must be different from others. Indeed, the obtained results are hard to interpret and produce noisy animations that might disturb observers.

(a) (b) (c) (d)

Figure 1.11: Painterly rendering using a vector field [Hays 04]. (a) Original image (b) Vector field computed via a local color gradient. For videos, Hays et al. use optic flow. (c) The vector field after radial basis function interpolation, and corresponding strokes attributes (e) The final painterly rendering.

Another approach is to use a *motion flow* [Litwinowicz 97, Hays 04] as the basic picture primitive. The motion flow itself can either be extracted from a video (it is then usually called optic flow), or more accurately computed from an animated 3D scene, but as mentioned above, this will make a small difference in the final rendering (e.g. some deformations may appear at image borders when the flow is computed from a video). In these techniques, paint strokes that follow the flow are distributed in the picture plane. Some of their attributes may be mapped from the flow itself, like length and orientation; while some others come from the underlying scene, like color. An inconvenience of this approach is that strokes sometimes seem to "slide" on objects: Indeed, there is no notion of object boundaries, strokes thus follow the global motion field, usually changing color when crossing an object or texture boundary. As long as the motion field is smoothly varying, the resulting strokes are temporally coherent. Therefore, in order to impose this smoothness constraint, Hays et al. [Hays 04] use radial basis functions to interpolate between values of the motion field that have a high "confidence" (see Figure 1.11).

Color regions may also be extracted using various segmentation methods. The most straightforward method consists of using an existing segmentation. Luft et al.

(a) (b)

Figure 1.12: (a) Regions are directly mapped from the output of a toon shader in [Bousseau 06], while in (b) regions are extracted using a Normalized Cuts segmentation [Kolliopoulos 06].

[Luft 05, Luft 06] directly use the ID images coming from the 3D scene to create a perfectly coherent segmentation of the picture plane. Bousseau et al. [Bousseau 06] use a toon shading instead, which allows them to create fuzzy borders between some of the color regions (see Figure 1.12(a)). In both approaches, segmented regions are then filled with a watercolor style (a good example of continuous tone). Another approach consists in extracting regions from pixel colors like in the work of Wang et al. [Wang 04c], possibly taking into account other attributes like depth or IDs as in the work by Kolliopoulos et al. [Kolliopoulos 06] (see Figure 1.12(b)). Both methods use sophisticated segmentation procedures (Mean-shift segmentation for the former and Normalized Cuts for the later) and are rather costly in computation times. Moreover, segmentation has to be performed on a sequence of images (sometimes called a video cube) to ensure temporal coherence. In [Wang 04c], a user interface is proposed to correct and merge regions, and then to add strokes that move with the regions via key-framing, while in [Kolliopoulos 06], strokes are attached to objects in the scene as explained in object-space primitive techniques below. The resulting animated filled regions emphasize the picture plane, but this approach could be pushed further by working on an efficient and temporally coherent parametrization of regions.

Finally, *lines* can be extracted in image space to convey a notion of shape. Interestingly, there has not been much work done on this problematic. Collomosse [Collomosse 04], as well as Wang et al. [Wang 04c], use extracted region borders to define lines that evolve more or less coherently from frame to frame (represented as surfaces in the video cube representation). Winnemoller et al. [Winnemöller 06] recently proposed a contour detection mechanism that evolves coherently over time based on differences of gaussians. Those methods are limited in the sense that they do not provide any parametrization of lines; there is thus no trivial method to stylize the extracted primitives.

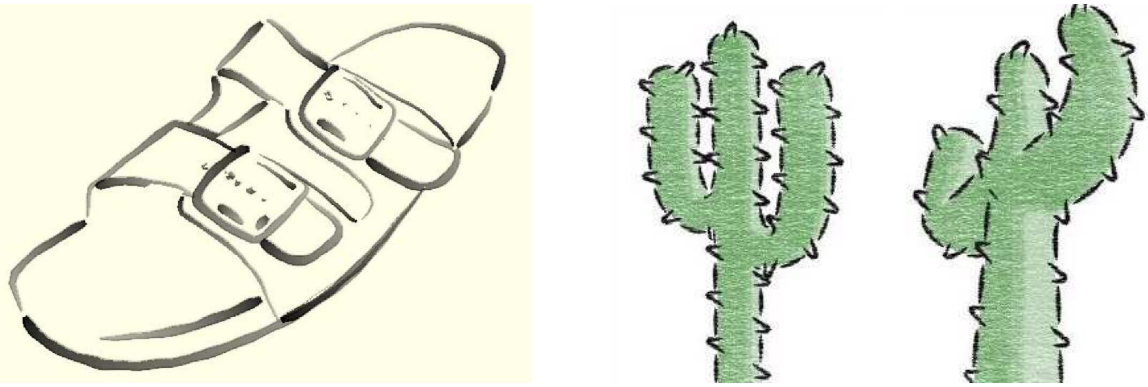
Renderings performed from image-space extracted primitives thus give a strong impression of a canvas painted frame by frame that loosely represents some information coming from the scene (at least colors). Therefore, they give little cues about

shape, hence abstracting the depicted scene and focusing on the picture plane: For instance, they allow to group objects under a single primitive (think of a forest in the background, represented with a single color region), resulting in more **view-centered** representations. Some methods are also highly dependent on the extraction of the 2D primitives they follow, which results in the need for manual editing and creates problems at image borders.

Figure 1.13: Two decal strokes examples of the WYSIWYG NPR system from Kalnins et al [Kalnins 02].

Object-space primitive mapping Object-based techniques offer another metaphor that gives more importance to the representation of scene properties. Their main idea is to distribute scene primitives onto surfaces that will later be projected onto the picture plane and used to draw paint strokes. One possibility is to lay down *paths* onto a surface and use their visible projection from a given viewpoint to draw strokes. This approach has been investigated in systems such as WYSIWYG NPR [Kalnins 02] or Deep Canvas [Daniels 99]. The resulting "decal strokes" stick to the depicted shape and tightly follow its motion, ensuring good temporal coherence (see Figure 1.13). Their style is usually set by the user in a pre-process, and to deal with density issues, levels-of-detail can also be provided manually. This approach is very efficient for representing some surface pattern or texture. However, for the dense distributions of strokes usually found in painterly renderings for instance, it imposes many constraints on the user side: The specification of paint strokes is a tedious process if one wants to deal with density issues and style variations, and, due to perspective projection, there is no way for the user to predict the 2D shape of the paint strokes in advance.

A specific category of paths are the ones corresponding to *contours* of an object. Their main use is in depicting shape with an economy of primitives. There exists a vast literature dealing with the subject, and various subcategories of contours as well. For a deep treatment, the interested reader is referred to the survey by Isenberg et al. [Isenberg 03]. To summarize, three main classes of contours are frequently used in expressive rendering: silhouettes, suggestive contours and ridges & valleys. Silhouettes (see for instance [Northrup 00], Figure 1.14(a)) broadly correspond to the loci of points separating the visible portions of a surface from its invisible portions (some formulations extend this definition to hidden silhouettes); Thus silhouettes



(a) (b)

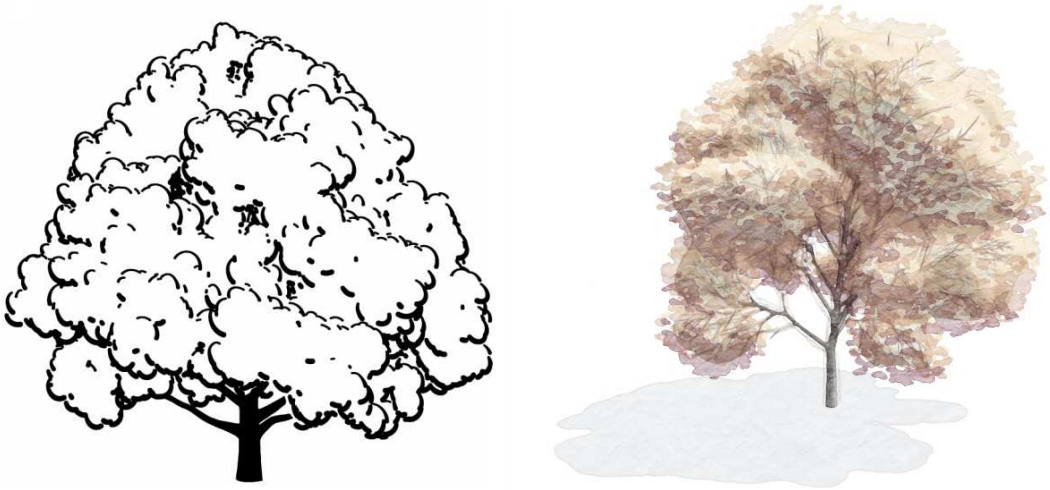
Figure 1.14: Silhouettes extracted and stylized with (a) the technique of Northrup et al. [Northrup 00] and (b) the technique of Kalnins et al. [Kalnins 03]

are inherently view-dependent and have to be updated for each frame. Suggestive contours [DeCarlo 03] extend the notion of silhouettes by considering the points that would be part of the silhouette in a nearby view, and are thus also view-dependent. Ridges & Valleys (see for instance [Yoshizawa 05]), on the other hand, correspond to loci where principal curvatures are local extrema, and are thus independent of the viewpoint; Note that creases can be considered as extreme cases of ridges & valleys. The choice of a class of lines to render is not an easy one: they are not mutually exclusive, and there is no conclusive evidence that one class is better than another. Another problem is that for view-dependent primitives, there is no trivial way to assign a temporally coherent parametrization, and thus stylization is not an easy task either. Kalnins et al. [Kalnins 03] proposed an efficient method to deal with this problem via an hybrid technique that works both in object and image space (see Figure 1.14(b)).

Figure 1.15: Two frames from Meier's painterly animation system.

Another possibility is to use *particles* on surfaces instead of paths. This time, the projected particle in the picture plane becomes a point to which a paint stroke, for example, is anchored. This method, first introduced by the pioneering work of Meier [Meier 96], has the advantage of giving many more degrees of freedom during stroke drawing (since only the position of the strokes is constrained), while ensuring temporal coherence because of the coherent motion of the 3D particles (see Figure 1.15). However, the selection of a viewpoint is limited in Meier's initial system: no density management is performed, limiting viewing to a small range of zooms and a

few rotations. View-dependent methods that deal with stroke density have been investigated for stippling [Cornish 01, Pastor 03], paper effect [Kaplan 05], watercolor [Bousseau 06] and painterly rendering [Chi 06]. They use a pre-computed hierarchy of particles that is pruned depending on the current viewpoint. Between two consecutive frames, particles may be added or removed from the hierarchy, thus producing poppings when the corresponding stroke is introduced or removed from the rendering. To reduce these temporal artifacts, the authors make use of transitions that are usually specific to the type of strokes they consider: In [Cornish 01, Pastor 03], only black point strokes are considered and they appear and disappear by varying their thickness; In [Kaplan 05, Bousseau 06], the strokes represent paper fibers or pigments, have fixed attributes independent of the scene and are simply blended; and in [Chi 06], strokes are represented by tonal art maps [Praun 01].



(a) (b)

Figure 1.16: (a) Pen and ink rendering of trees [Deussen 00b] exhibit implicit regions in the foliage (b) Watercolor rendering [Luft 06] where blobby particles are merged to create regions.

Such particles can also be used to create region primitives. An early method of Deussen et al. [Deussen 00b] used this technique to create abstracted foliages of trees in a pen and ink style, by agglomerating particles (leaves) both close in the picture and in depth (see Figure 1.16(a)). Another method is the one of Luft et al. [Luft 06] that creates blobby regions at each projected particle; these regions are agglomerated to create a bigger region that will be later filled with a watercolor rendering technique (see Figure 1.16(b)). Bousseau et al [Bousseau 06] take a complementary approach, where each particle holds a spot of low- or high-frequency noise, that mimics the inhomogeneous diffusion and pigmentation of watercolor respectively.

Renderings performed from object-space mapped primitives tend to emphasize more properties of the scene than properties of the picture plane. The resulting picture primitives motion is tightly attached to that of objects in the scene, and depending on the chosen technique (paths in particular), they can reveal important shape cues. Hence they produce more **object-centered** representations.

Mark implementation Once picture primitives have been chosen, there are many different ways to implement them using what we called marks in the previous section.

(a) (b)

Figure 1.17: (a) Simulation of thick paint (impasto) by Baxter et al. [Baxter 04]. (b) Mimicking of an impasto effect by Hertzmann [Hertzmann 02a]

Most of the previous work concentrated on simulating or mimicking traditional media like oil paint, acrylic, pen and ink, watercolor, etc. However, the choice of a particular model is rather independent of the previous stages: Maybe the only requirement is that it does not introduce any temporal incoherence. Here, we will only scratch the surface of possible techniques. For a deeper treatment of the subject, the interested reader might consult the survey by Hertzmann [Hertzmann 03].

Concerning medium simulation, three important papers are the watercolor dispersion model of Curtis [Curtis 97], the ink dispersion model on absorbent paper by Chu et al. [Chu 05], and the impasto techniques of Baxter et al. [Baxter 04] (see Figure 1.17(a)). A problem with approaches based on simulation is that, if they are recomputed for every frame of an animation, there is no guarantee that the result will be temporally coherent. Moreover, the degree of control is more oriented towards interaction than animation: The final look is governed by the dispersion simulation, making it hard to control it procedurally; But at the same time, the medium "reacts" to user interactions in real time for most approaches, making them very efficient as interactive tools, with all the advantages of computer graphics (e.g. undo/redo).

Recall that simulation is not necessary for producing expressive renderings, and this is especially true for animation. A more important aspect is the degree to which the style of a representation can be finely designed, and reproduced in an animation with temporal coherence. With this idea in mind, previous work concentrated on creating simple yet efficient models of media. The most widely used is the skeletal stroke model of Hsu et al. [Hsu 93]. To represent a brush stroke, they simply use a triangle strip with a texture and a bunch of parameters like thickness, length, and curvature. This model has an interesting extension described by Hertzmann [Hertzmann 02a], that re-creates the impasto effect found in oil paintings for instance, via the use of height fields(see Figure 1.17(b)). It would be interesting to depart from traditional techniques and invent new "media" with new properties that would not be feasible with traditional techniques.

1.4 Contributions

The use of a model of representation in Section 1.2 both helps to better understand how to transfer essential qualities of traditional paintings and drawings to expressive rendering techniques, and to organize previous work in this field based on how an approach fit in each of the four representation systems. It also allows us to define new research directions, by pointing out schema present in traditional techniques that have not yet found their counterpart in expressive rendering. The next three chapters present such contributions.

In Chapter 2, a new attribute mapping method is presented in the form of an extended toon shader. Its goal is to give an intuitive control over the color attribute (expressive shading) based on other attributes such as surface depth, surface orientation, distance to a focus point, material specularity, or even motion flow. This way, it allows common color effects found in traditional techniques (or schema) to be easily incorporated in any expressive rendering system by the means of a simple shader. Examples of schema include aerial perspective to reveal depth, abstraction of colors found near silhouettes to soften or harden shape, decrease of contrast or opacity away from a focus point in order to emphasis some surface region, and stylization of highlights to suggest material properties.

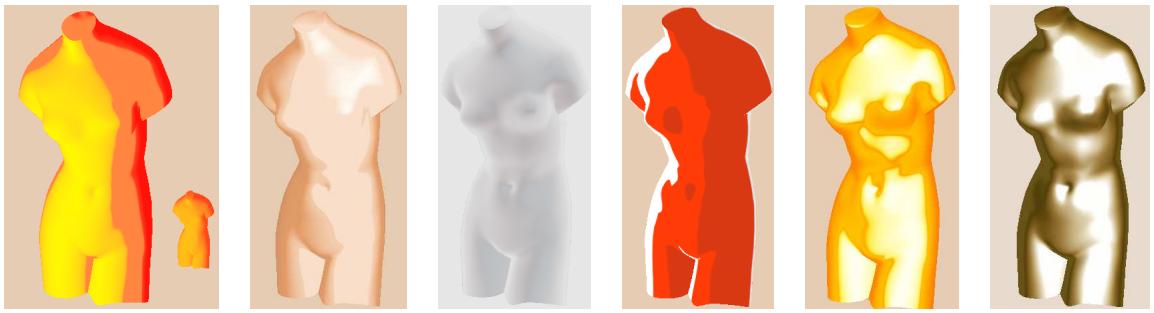
In Chapter 3, we focus on painterly rendering with individually perceived paint strokes and address some of the limitations found in previous approaches: Meier's work is extended to dynamic lighting and viewing environments, without imposing any particular constraint on the style of marks to draw. To do that, we perform a density management coupled with a stroke marching technique: Paint strokes are drawn at locations defined by the density operator, with the style provided by the user, as long as the important features of the scene are conserved. This performs a trade-off between a low-level style to represent and the scene elements to depict. Temporal coherence is ensured both during density management and stroke drawing, with a particular care on stroke insertions or deletions. We also designed our system to be an explicit instance of the representation model.

Finally, in Chapter 4, we present an alternative metaphor that has seldom been addressed in previous work: the depiction of tone via stroke patterns like hatching, half-toning and screening. This schema is extremely common in comics and in traditional illustration, as discussed in the beginning of this chapter. However, its extension to animation is not straightforward since a stroke pattern is a (large) 2D primitive that is supposed to follow the complex motion of a 3D object. The main difficulty is thus in the primitive mapping and its temporal coherence during animation. We show that an optimal pattern mapping can be created and illustrate it with various pattern styles that depict tone in different ways.

We close this Part on animation by discussing how available techniques might be enriched in order to allow more and more intuitive control on the user side, which leads us to consider computer depiction as an optimisation process. In particular, we consider the role of low-level visual perception in such an optimisation as an objective to seamlessly fuse user intention and scene features in motion.

Chapter 2

An extended toon shader for attribute mapping



(a) (b) (c) (d) (e) (f)

Figure 2.1: Some example effects achieved by our extended toon shader: Continuous levels of abstraction (a); Abstraction of near-silhouette regions: Smoothing (b), and opacity (c); Back-lighting (d); And highlighting: Plastic (e), and metal (f).

Our first contribution focuses on the single most important attribute of the expressive rendering pipeline, color, and to this end we extend an existing technique: cartoon shading.[1] Note that the presented approach is able to improve any expressive rendering system, regardless of the style represented. Indeed, any renderer, even a photo-realistic one, has to define the color of its final marks. More precisely, we target an intuitive design tool to control color based on other attributes such as depth or orientation.

Over the past decade, "toon" shading has proved popular in a variety of 3D renderers, video games, and animations. The idea is simple but effective: extend the lambertian shading model by using the computed illumination (a dot product between a light vector and the surface normal) to index into a 1D texture that describes how the final shading varies from "dark" to "light" regions. The designer controls the behavior of the shader by creating a 1D texture, typically composed of two or three regions of constant color, to mimic the flat regions of constant colors found in comics and

[1]This work was done during an EURODOC internship at the University of Michigan, in collaboration with Professor Lee Markosian, and my co-advisor Joëlle Thollot. It has been published in the International Symposium on Non-Photo-realistic Animation and Rendering (NPAR) in 2006, see [Barla 06c].

traditional animation. Toon shading can be implemented efficiently via vertex and fragment programs on modern GPUs.

Toon shading however suffers from several limitations. First, it does not reflect the desired level of abstraction (LOA) of a surface. Such LOA behavior plays an important role in traditional media, however. Often, some objects are considered less important (*e.g.*, characters in the background) and thus are depicted with greater abstraction [McCloud 94]. In paintings and drawings, an effect known as aerial perspective makes objects in the background appear desaturated and less detailed than those in the foreground. And in scientific illustration, a technique similar to depth-of-field is used to focus on a region of a shape by decreasing contrast or opacity in less-important or "out-of-focus" parts of the surface [Wood 94].

Second, conventional toon shading is view-independent, and so cannot represent plastic or metallic materials, for which view-dependent highlights are of primary importance. Similarly, it cannot support the view-dependent back-lighting effects, often used in traditional comics and animation, in which a virtual "back light" illuminates the surface near the silhouette.

Finally, in conventional toon shading, every surface location is rendered with full accuracy, so that even small shape features are depicted by the shading (at least for some light directions). This can be desirable, but often designers working traditionally apply a degree of abstraction so that small shape features are omitted. A similar ability to depict an abstracted version of the shape is thus desirable in an automatic toon shader.

To meet these goals, we present X-Toon, a toon shader that supports view-dependent effects through two independent extensions to conventional toon shading. The first incorporates a notion of **tone detail**, so that tone varies with depth or orientation relative to the camera. For this, we replace the conventional 1D texture used in toon shading with a 2D texture, where the second dimension corresponds to the desired "detail" (see below). We describe several ways to define the additional texture coordinate.

Our second extension, presented in Section 2.3, lets us vary the perceived **shape detail** of the shading. We achieve this by using a modified normal field defined by interpolating between normals of the original shape and normals of a highly abstracted shape. This approach has the advantage of abstracting the shading from a shape, while preserving silhouettes.

Note that "detail" here corresponds to visual abstraction: less detail means greater abstraction. We use the term LOA instead of LOD to emphasize that our goal is visual abstraction, not increased rendering speed (the usual motivation for LODs in computer graphics).

The main contribution of this work is a simple, unified approach that lets a designer intuitively achieve a variety of effects, including those that rely on *correlating* desired tone detail with the underlying tone value, as explained in Section 2.2.

2.1 Previous Work limitations

Mip-maps provide one means of taking depth into account as an attribute. Klein et al. [Klein 00] used specially constructed mip-maps (called "art maps") to achieve constant-sized strokes in textures applied to 3D scenes. The "tonal art maps" of Praun

et al. [Praun 01] extended this approach to take lighting into account in hatching patterns.

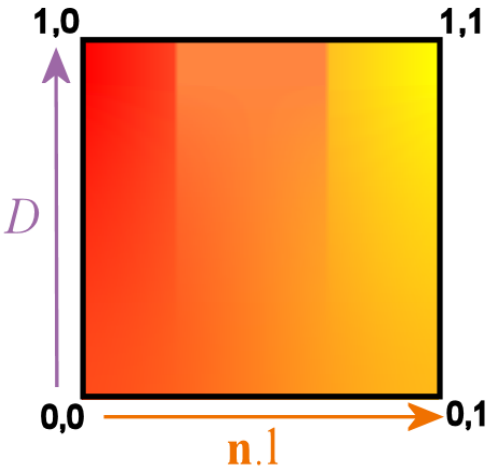
We could similarly define specialized art maps for the management of 1D toon textures with LOAs. Instead, we prefer a more general approach where a 2D texture represents an ordinary toon texture at a continuous range of abstraction represented by the added dimension. This way, the LOA selection mechanism can be more general than the fixed depth-based computations used in mip maps. It also lets designers create LOA toon textures directly as 2D images, *e.g.* by painting them in a paint program. The only requirement is to understand that the 2 dimensions of the texture correspond to tone and LOA, respectively. This approach retains the simplicity of the classic toon shader and does not constrain the designer with a set of predefined behaviors and discrete LOAs.

Anjyo el al. [Anjyo 03, Anjyo 06] described techniques to add stylized highlights to a classic toon shader, thus taking viewing orientation into account. Their technique, which acts on top of a toon shader, supports abstraction of highlights through the use of translation, rotation, splitting and squaring operations. Other methods could be devised to control the highlights independently of the tone computation (*e.g.*, using a 1D highlight texture). However, once the properties of a highlight (*e.g.*, size, softness, color) are chosen, they remain the same under any viewing condition. Thus, the method of Anjyo el al. only considers highlights with hard contours, a single color and a fixed size. Our approach offers the ability to correlate properties of the highlights with the tone value, in a way precomputing the response of the highlight to changing viewing conditions: highlights can change in softness, color and size dynamically.

We now describe our first extension to the classic toon shader: tone detail.

2.2 Tone detail

We extend the classic 1D toon texture by adding a vertical "detail" axis (corresponding to LOA) to build a 2D toon texture. The horizontal axis corresponds to tone as in a classic toon texture – the texture coordinate along that axis is derived from a standard lambertian shading computation: $\mathbf{n} \cdot \ell$, where $\mathbf{n}$ is the unit surface normal and ℓ is the unit light direction. The vertical axis corresponds to tone detail: each value along this axis (labelled D for detail) corresponds to its own 1D toon texture. The whole 2D texture can thus be regarded as a stack of 1D toon textures with increasing "detail." The designer is free to create this 2D texture using any image processing or painting tools available. We generally found it convenient to start with a standard 1D toon texture at $D = 1$ and apply image processing transformations down the D axis to account for detail loss. We provide many example textures that have been created with this approach, and we demonstrate the resulting behavior of the extended toon shader.



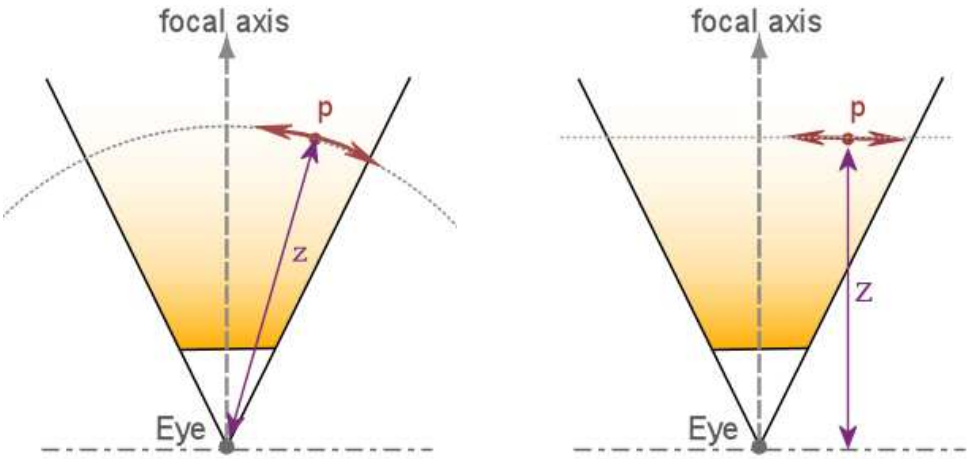
Once the 2D texture is defined, the designer must choose an attribute (*e.g.*, orientation or depth) that will control the tone detail, and provide functions called **attribute**

maps that map the attribute to a detail value D at each location on a surface. This formulation is general in that any attribute can be chosen, depending on the application goals.

In this work, we consider view-dependent attributes. In the next two sections we describe how to compute depth-based and orientation-based attribute maps to achieve LOAs, aerial perspective, depth-of-field, back-lighting, and specular highlights.

2.2.1 Depth-based attribute mapping

We consider two ways to define the "depth" of a point in 3D: we can use its euclidean distance to the viewpoint, or its distance along the focal axis (see Figure 2.2). The latter assigns the same depth to all the points that lie in a plane parallel to the image plane. Depending on the intended effect, one computation or the other may be preferred. For LOAs and aerial perspective effects, we use distance along the focal axis; for depth-of-field effects, we prefer distance to the eye.



(a) (b)

Figure 2.2: The depth of a point p relative to the eye can be calculated in two ways: (a) The distance between the eye and p or (b) the distance along the focal axis.

We derive a detail value $D \in [0..1]$ from the computed depth z via two user-specified parameters: z_{min}, the distance at which the detail starts decreasing, and $r > 1$, the scale factor that defines the coarsest detail (greatest abstraction) at distance $z_{max} = rz_{min}$. To account for perspective projection, we use the following formula for detail mapping [2]:

$$D = 1 - \log(z/z_{\min})/\log(z_{\max}/z_{\min})$$

$$D = 1 - \log_r(z/z_{\min})$$

For depth-of-field, we use a different formula for the detail computation: Given a focus point c in 3D, we compute D as follows [3]:

$$D = \begin{cases} 1 - \log(z/z^-_{\min})/\log(z^-_{\max}/z^-_{\min}), & z < z_c \\ \log(z/z^+_{\max})/\log(z^+_{\min}/z^+_{\max}), & z > z_c \end{cases}$$

[2] We assume $0 < z_{min} \leq z \leq z_{max}$.

[3] We assume $0 < z^-_{max} < z^-_{min} < c < z^+_{min} < z^+_{max}$.

with $z^{\pm}_{\min} = z_c \pm z_{\min}$ and $z^{\pm}_{\max} = z_c \pm r z_{\min}$.

In our implementation, we compute D per vertex in a vertex shader. A pixel shader performs the 2D toon texture lookup using the value of D interpolated at each pixel. In an interactive session, the designer can thus experiment with the z_{min} and r parameters and set the point of focus, and then observe the resulting behavior of the X-Toon shader in real time.

Figure 2.5 shows depth-based tone LOA effects using 2D textures created in three different ways: in (b) we progressively blurred an input 1D toon texture, in (c) we interpolated between three 1D toon textures with smooth transitions, and in (d) we shifted and lightened a 1D texture to create a "receding shadows" effect. The LOA attribute map is shown in (a).

Figure 2.6 shows two examples of a focus-based detail map. The depth-of-field attribute map is shown in (a), along with the focus point depicted by a small blue sphere. In (b), we used a smooth two-tone texture that converges to a constant color with detail loss. In (c), we use a texture that decreases opacity and contrast with detail loss, and only shadows and highlights are fully opaque. We used these effects to focus attention on a specific depth range of a mechanical model.

Finally, Figure 2.7 shows a landscape rendered with four different aerial perspective textures. We created these textures using filters such as decreasing contrast (b), converging to a specific hue (c), and decreasing opacity (d). We did this independently for the main tone values (dark, intermediate, and highlights) so that we can correlate the change made by aerial perspective with a specific tone value (*e.g.*, we make the intermediate colors more transparent with distance). The aerial perspective attribute map is shown in (a).

2.2.2 Orientation-based attribute mapping

We next consider an attribute map based on the orientation of the surface with respect to the observer. With the appropriate choice of 2D toon texture and policy for computing D, we can achieve effects such as fading of near-silhouette regions, or brightening and coloring of near-silhouette regions to suggest a virtual "back-light." We define the near-silhouette attribute mapping as follows: $D = |\mathbf{n} \cdot \mathbf{v}|^r$, where $\mathbf{n}$ and $\mathbf{v}$ are the unit normal and view vector, respectively, and $r \geq 0$ is a user-defined parameter that controls the magnitude of the effect.

Specular highlights are another view-dependent effect: they depend on the angle between the viewing direction and the light reflection vector, and the specular properties of the material. Depending on the chosen 2D texture, we can model various material highlights intuitively (*e.g.*, plastics or metals). This is better than just compositing a specular layer over the ordinary 1D toon shader, because the 2D texture lets us control the profile of the highlight (smoothness, width, alpha) so that it is correlated with the underlying tone. In our implementation, we use the Phong highlight model to map the highlight attribute values to detail: $D = |\mathbf{v} \cdot \mathbf{r}|^s$, where $\mathbf{r}$ is the light reflection vector at the current surface location and $s \geq 1$ is the "shininess" coefficient set by the designer to control the magnitude of the effect.

In practice, it is straightforward to compute these values in a vertex shader. For accuracy reasons, however, we pass the relevant vectors to a fragment shader where they are interpolated before being used in the detail computation.

Figure 2.8 illustrates how we can use the orientation detail map (shown in (a)) for

different purposes. In (b) we use it with a texture similar to the LOA examples in Figure 2.5 to abstract tonal detail in near-silhouette regions. In (c), we make the surface fade out in near-silhouette regions to yield a "fuzzy" appearance. A different 2D texture is used in (d) to produce an effect we call "backlighting". With the appropriate choice of 2D texture, the thickness and color of backlit regions can be made to depend on the tone value of the underlying 1D toon texture.

Figure 2.9 shows three types of highlights (with the detail map presented in (a)). The first depicts a plastic-like shader, where we control the thickness and opacity of the highlight directly in the texture. The second shows a metallic-like shader, where the highlight has a quality of "glowing" suddenly when the camera moves to a favorable orientation. While the first supports dynamic variation in the size of the highlights, the second controls the smoothness of tone transitions view-dependently. A third example modifies the colors that appear near transitions to highlights.

2.3 Shape detail

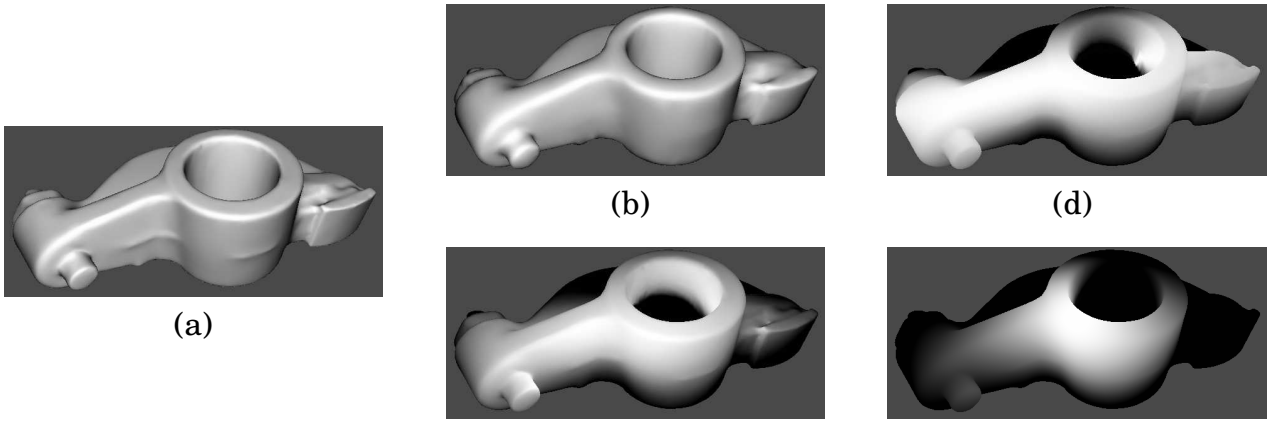


Figure 2.3: For each figure, darkest tones correspond to normals that are oriented away from the camera. (a) The normals of the input mesh. (b) Smoothed normals (note the shape details that disappeared in the front of the object). (c) Elliptic normals whose distribution follows the principal direction of elongation of the object. (d) Cylindric normals, oriented in the direction of principal elongation. (e) Spheric normals, highly abstracted from the original model. Note that for every geometric map, the silhouettes are preserved.

We now describe how to further modify the shading to depict an abstracted version of the shape. Note that this extension is independent of the tone detail extension and is done view-independently. We achieve it by modifying the surface normal field used in shading computations with a **geometric map**: we map the input mesh to an abstracted shape that acts as a lower bound for shape detail. We ensure this abstracted shape has a direct correspondence with the input surface. The designer controls the "abstraction" of shape detail by setting a blending parameter that controls how the shader interpolates between normals of the input mesh and the abstracted shape. Note that while normal vectors used in shading calculations are modified, ver-

tex locations are not. This has the important property of ensuring consistency when combined with other rendering passes.

We describe four types of abstracted shape: a smoothed version of the original mesh, and an enclosing ellipse, cylinder, and sphere. We show examples of these abstracted shapes in Figure 2.3, rendered using a simple lambertian shader with the light at the camera to better visualize the abstraction. The dimensions of the ellipse come from a bounding box of the input mesh. To map normals from the input mesh to the sphere, cylinder, or ellipse, we use straightforward analytic mapping techniques. We create the smoothed mesh using a simple normal smoothing technique, and the 1-1 correspondence between mesh vertices defines the mapping. These four types of abstracted shapes provide a great deal of expressiveness: they let us "flatten" the shading in a way that resembles some drawings, paintings and comics. Note that this manipulation of normals to abstract the shading need not be restricted to toon shading.

In our implementation, we precompute and store abstracted normals at each mesh vertex. A vertex shader is used to linearly blend between the input and abstracted normals using a single (global) weight provided by the designer. The resulting vector is renormalized in the pixel shader where the toon texture lookup is performed. While simple, this method works reasonably well, is easy to implement, requires just one extra normal per vertex, and provides interactive feedback when implemented on the GPU. Of course, for the shading to be coherent, the normals of the highly abstracted shape should not exhibit any degeneracies.

2.4 Discussion and future work

We have presented two independent extensions to the original toon shader, aimed at retaining its simplicity while allowing more general behaviors. An important property of the original toon shader is that it is fast. To compare our approach in terms of efficiency, we have made a set of measurements summarized in Table 2.1. The rendering time of X-Toon for typical scenes is only about 20% slower than the original toon shader, and can thus be used in real-time applications as well as for off-line rendering.

Model	*Size*	*Resolution*	*Toon*	*X-Toon*	*Ratio*
Mech part	$10,000$ ○,	$640x480$	287	241	1.19
	$20,000$ △	$1280x950$	268	222	1.21
David	$26,051$ ○,	$640x480$	94	80	1.18
	$49,998$ △	$1280x950$	90	76	1.18
Terrain	$105,152$ ○,	$640x480$	33	28	1.18
	$208,962$ △	$1280x950$	32	27	1.19

Table 2.1: Run-time performance (in frames per second) and ratio of toon to X-Toon frame rates for three different models (○ = vertices, △ = triangles) at two different resolutions. These measurements were made on a Pentium 4 with ATI Radeon 9800 GPU.

Although we have focused in our approach on view-dependent attributes, our system can easily be extended to handle other attributes. For example, we can use optical

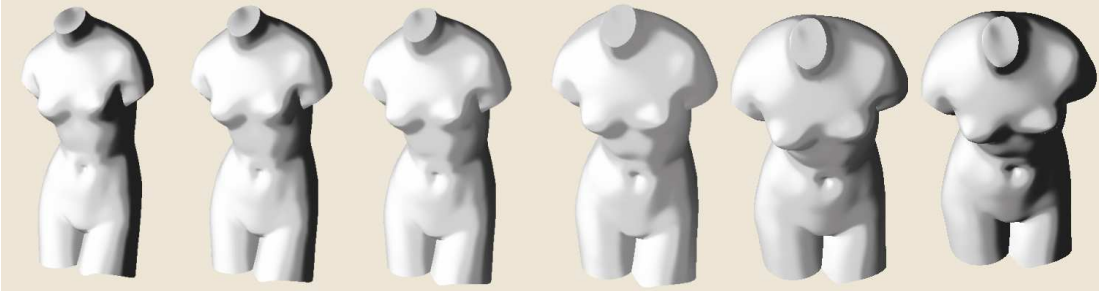


Figure 2.4: We show six frames of a small animation where coarser detail is automatically selected where the magnitude of optical flow is higher. We used the receding shadow texture from Figure 2.5.

flow to assign less detail to objects as they move faster in image space. To implement this method, we store the world-to-pixel matrix from the previous frame and use it to compute a per vertex displacement vector, measured in pixels. We use the length of this vector to assign a value to D (See Figure 2.4). In future work, we plan to investigate additional attribute detail maps, *e.g.* based on haloing, labeled importance, points of interest, cast shadows or reflections, and find a way to combine them efficiently.

Our geometric mapping is similar to the one developed by Ni el al. [Ni 06], except that they interpolate normals *and* positions using a *local* detail map, and they support the use of multiple shapes (not just two) representing distinct levels of detail. Applying similar ideas in an X-Toon shader is one possibility for future work. Another possibility is to take inspiration from Anjyo el al. [Anjyo 03, Anjyo 06], and use light vector fields to control the shape of highlights. We could also consider ways to control detail parameters over time (via key-framing, *e.g.*) as was done in the same paper.

We have only considered a single global weight to interpolate between the original and abstracted normals. A more flexible approach would be to control shape detail using a locally varying weight. This varying weight could be provided by the designer via a texture painted on the surface, or through an attribute map similar to those we have described for controlling tone detail. Controlling shape and tone detail via the same attribute maps may make sense in terms of providing more consistency in the resulting shading.

As future work, we plan to implement a layered version of X-Toon, with each layer controlling a single effect via tone and shape detail. Layers can then be composited using classic blending operations like in image processing softwares; they can be kept separate and run through the primitive system, up to the mark system, so that not only attributes are layered, but also the whole final representation. This last approach has a strong similarity with traditional painting and drawing layering methods discussed in Chapter 1, and has also perceptual justifications as discussed in Chapter 5.

Finally, many similar tools are awaiting to be designed to intuitively control other attributes. For properties related to illumination (like shadows, highlights, translucency, etc), it seems that a layered version of X-Toon would be appropriate. However, one might want to design alternative tools more tailored to other attributes such as depth, where X-Toon might not be adapted. It would then be interesting to combine various tools into a versatile attribute system.

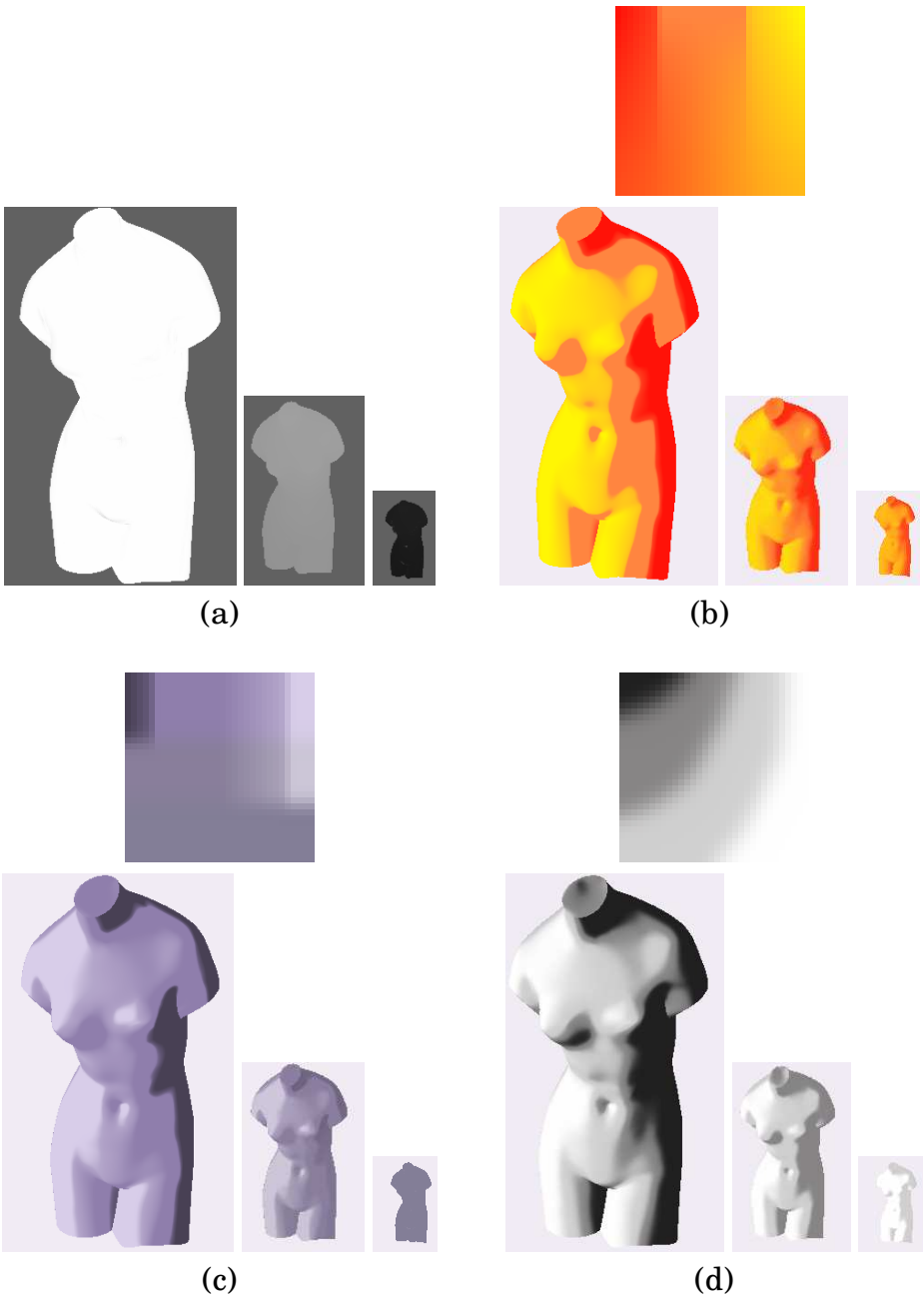
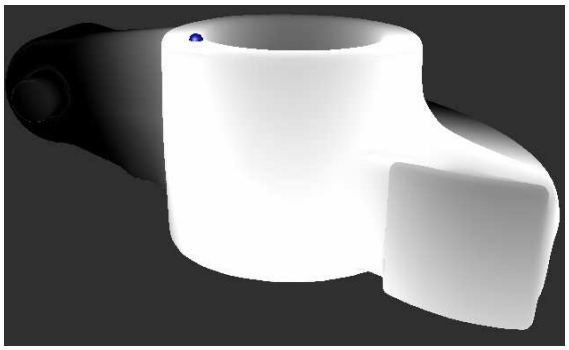
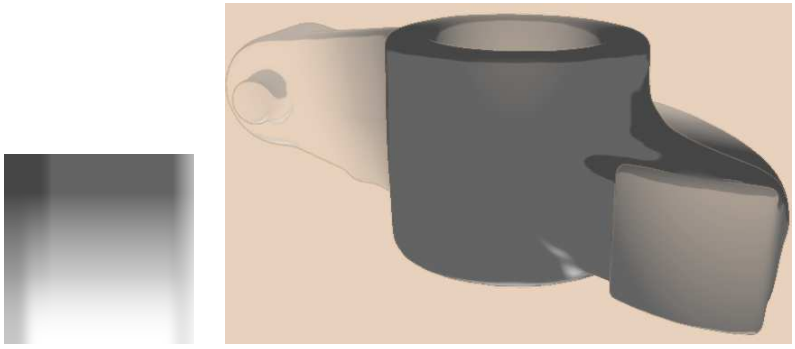


Figure 2.5: Some examples of the venus model rendered at various LOAs, with the corresponding toon textures on top. (a) shows the LOA detail map. In (b), we use a blurred texture to smooth out the shading with distance; in (c), we mimic a discrete LOA behavior using a texture with smooth steps along the detail axis, hence controlling the interpolation behavior; in (d), we make a receding shadow effect by sliding the darker tones to the left of the toon texture along the detail axis.



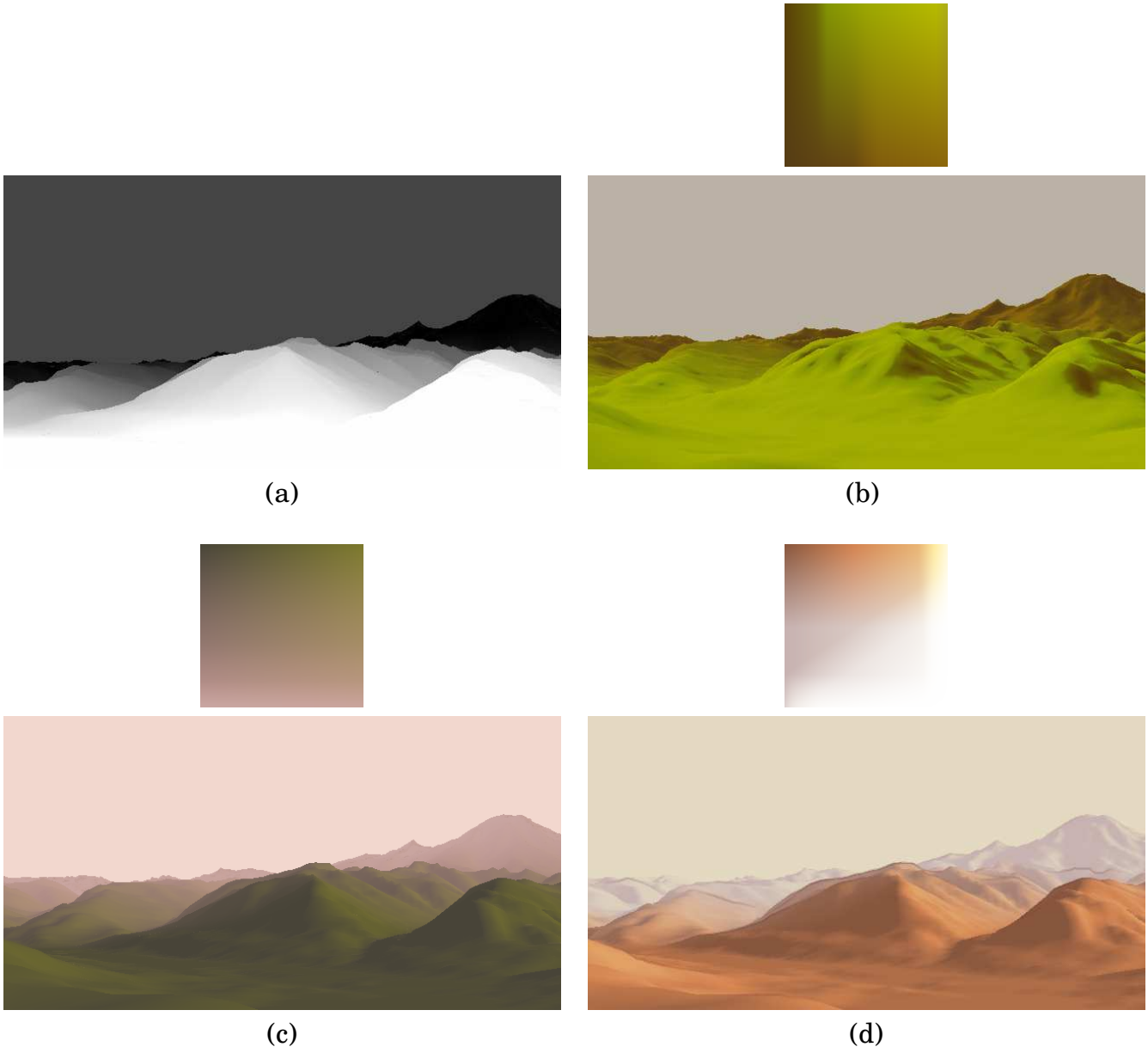
(a)

(b)



(c)

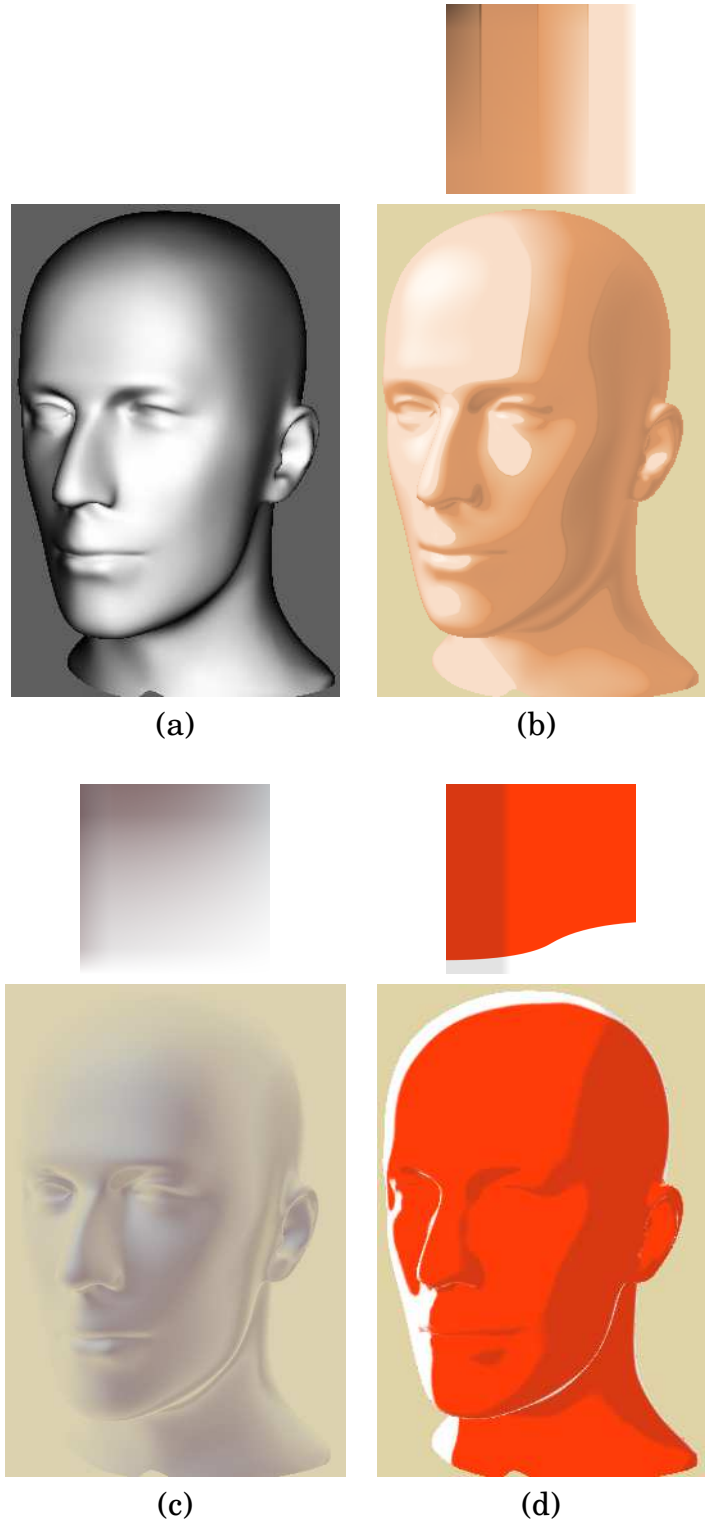
Figure 2.6: Some examples of depth-of-field shading, with the depth-of-field detail map presented in (a) (the blue sphere corresponds to the focus point): In (b), we use a blurred texture; In (c) we use a texture in which contrast and opacity are decreased for intermediate tones.



(a) (b)

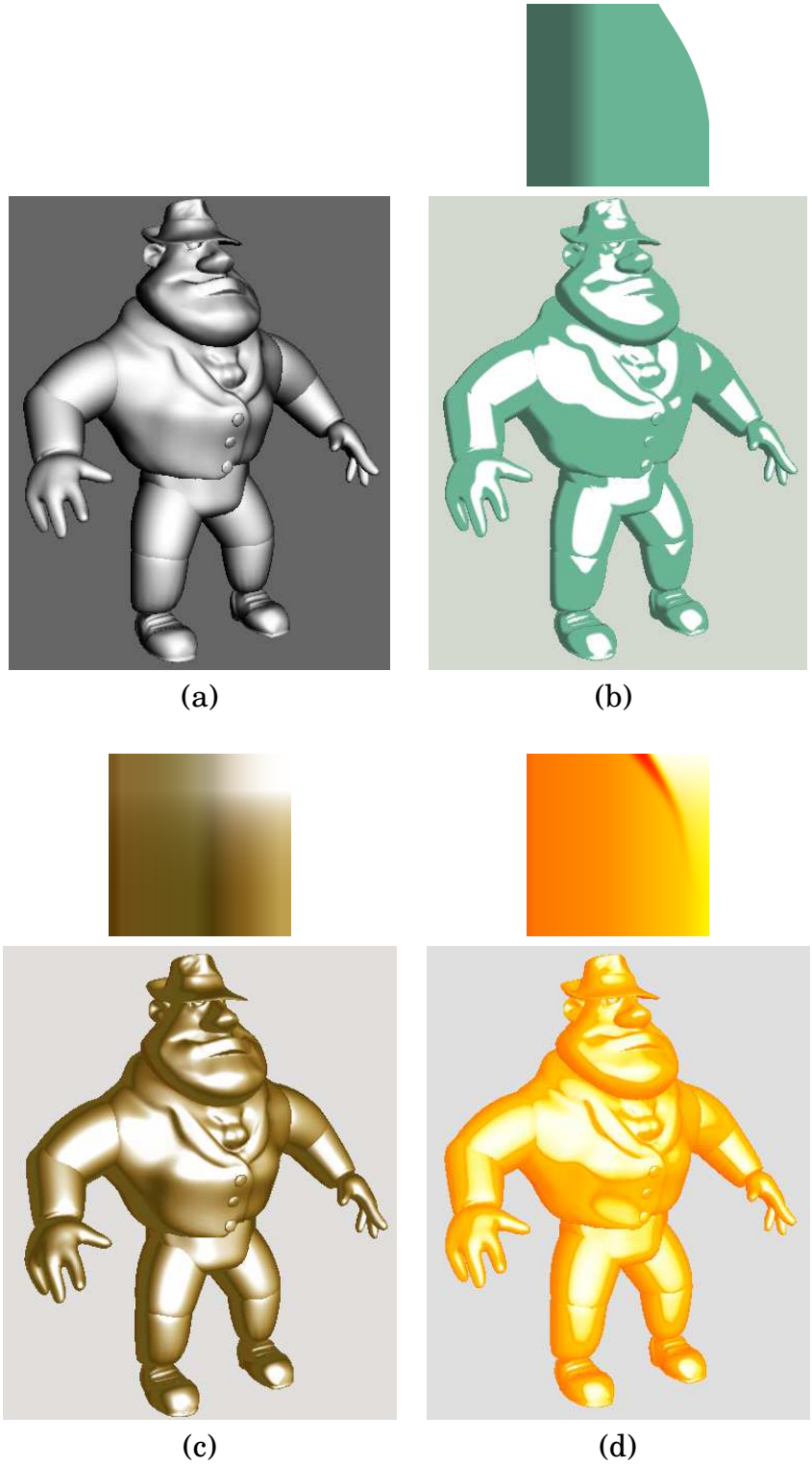
(c) (d)

Figure 2.7: Aerial perspective effects, with the aerial detail map shown in (a). The first texture in (b) blends from a green-to-brown color ramp to a more uniform one consisting of brown tones; The second one, in (c), applies a desaturation and shift toward pink hue, with darker tones modified prior to lighter ones. While in (d), it decreases alpha and saturation, keeping only light gray shadows in the background.



(a) (b) (c) (d)

Figure 2.8: Near-silhouette abstraction and back-lighting. (a) shows the orientation detail map. In (b), we use a texture where intermediate tones are blurred prior to dark or light ones; this has the effect of smoothing out the shading in near-silhouette regions so that hard boundaries are only visible in regions facing the camera. In (c), the same approach is applied, this time to decrease opacity of the shading except for dark tones; this gives a fuzzy rendering of the model. In (d), we apply a simple white back-light that grows thiner in darker tones; we can thus control the thickness of the back-lighting by moving the light.



(a) (b) (c) (d)

Figure 2.9: Plastic and metal highlights. (a) shows the highlight detail map. In (b), we use a highlight texture that decreases in width, but is static with respect to color (white) and profile (hard boundaries); this gives a plastic toon highlight that varies in size depending on the viewing configuration. In (c), we use a texture that creates a "glowing" highlight, resembling a metallic material; this highlight is only present when the view direction is close to the reflected light direction, making the highlight appear suddenly, like a flare. In (d), we modify the color of the highlight, making it redder near its boundaries when it reaches a given scale.

Chapter 3

A painterly rendering metaphor

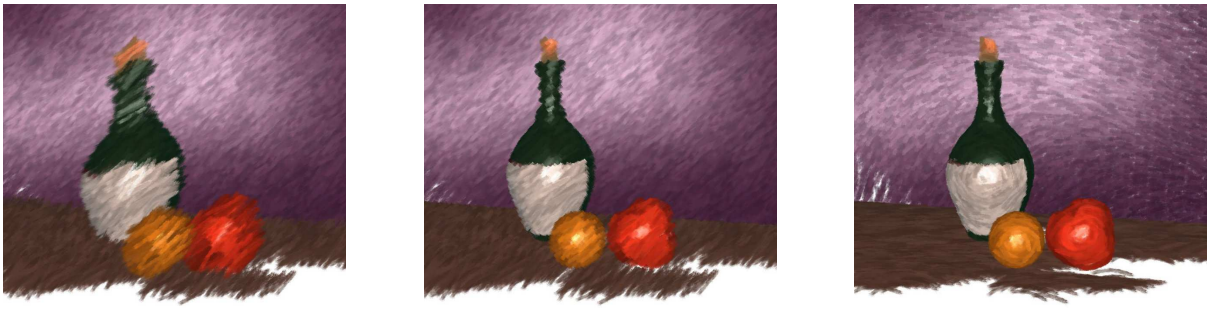


Figure 3.1: Left: when applied with an arbitrary style (here strokes have a common global orientation), previous painterly rendering methods fail to preserve scene colors and silhouettes; Middle: using our approach, we can keep those cues in the rendering; Right: a main benefit is then to be able to use various styles (here strokes follow silhouettes) while still correctly representing the scene.

In the previous chapter, we investigated a specific, though crucial, aspect of the representation model: color design in the attributes system. We now present an explicit instance of Willats and Durand model that aims at producing what has been called *painterly rendering* in the expressive rendering literature, and has first been introduced by the influential paper of Meier [Meier 96]. The development of the presented system have been jointly realized by David Vanderhaeghe, PhD student in ARTIS (my research team) and myself. For the sake of clarity, I will present the system as a whole, but will also point out what specific aspects have been developed by David, myself or both of us. [1]

Painterly rendering is a technique that takes inspiration from traditional paintings, usually focusing on effects similar to those achieved with oil or acrylic paint, where the individual brush strokes are more or less perceived individually (media such as watercolor introduce other issues such as the diffusion of pigments). The main idea is to render a scene projected on the image plane by a set of 2D vector paint strokes holding style attributes (color, texture, etc). This representation has the effect of abstracting the rendering by using primitives larger than pixels, and emphasizing the 2D nature of the resulting image. Its main advantages over traditional

[1] This work has been published as a technical sketch at the SIGGRAPH conference in 2006, see [Vanderhaeghe 06]. An extended version of the system is also in the process of submission.

painting reside in the possible creation of painterly animations: the automatic distribution of strokes makes the approach tractable, and the strokes motion is easily controlled where it is almost impossible to ensure a coherent motion by hand.

A natural question that arises with painterly rendering for animation is "what information does the configuration of paint strokes in the plane represent ?" As with the traditional painting and drawing techniques we described in the previous chapter, they simultaneously represent information about objects in the scene (such as shape or reflective properties of a surface from the current point of view) while following a stroke style provided by the user (straight or curved brush strokes, thick or thin outline, etc). During the animation they also follow the 2D or 3D motion of some scene elements. The main issues in painterly rendering comes from these conflicting goals (as stated by Durand [Durand 02]). **Temporal coherence** of the strokes motion is of primary interest: it arises when one wants to link the motion of a 2D primitive (a stroke) to the motion of a 3D primitive (e.g. a surface). Another important aspect is the **density** of strokes: when zooming in or out from an object, the number of strokes used to represent it must increase or decrease in order to maintain a uniform density in the picture plane while keeping a constant thickness in image space. Finally, an ideal painterly renderer would let the user fully specify the strokes **style** in a way that is independent of the depicted scene, but at the same time should ensure that some properties of the scene are well-represented, such as object silhouettes or lighting.

We propose an interactive system that takes inspiration from previous methods to deal with temporal coherence and density issues, and extends them to lessen the constraints on strokes style specification. The user is then free to design strokes in his own way, while the system ensures a consistent scene depiction at each frame, thus allowing more expressiveness in painterly renderings.

3.1 Previous work limitations

In Chapter 1 we reviewed different metaphors for expressive rendering. Of particular interest to painterly rendering are the methods that deal with *paths* or *particles*, because these primitives can be directly implemented via paint strokes. However, if one aims at producing painterly renderings where the whole picture plane is covered with paint strokes for any arbitrary view, then the use of paths becomes quickly intractable: Indeed, the user then has to draw paths that cover every surface for enough viewpoints. The use of approaches based on particles then seems a more reasonable choice for such a class of renderings: The user controls the rendering only at a global level, by setting strokes attributes and density, and they are then automatically rendered at runtime.

The first work to introduce the idea of using particles for painterly rendering is Meier's system [Meier 96]. The key idea is to distribute particles on a 3D model. Then for each frame of an animation, particles are projected to the screen and embodied by paint strokes that take their attributes from off-screen buffers. No visibility test is performed, thus strokes are drawn by a painter algorithm, in back to front order. The essential advantage of this approach is to guarantee temporal coherence: All the particles of a model are *always* embodied by a stroke, although they may be overlapped by closer particles in depth; Thus there is no insertion or deletion of particles, and the corresponding strokes evolve coherently.

However, it imposes a complete coverage of a surface by paint strokes, unless strokes corresponding to back-facing particles may appear. The choice of the particles density is then important in order to guarantee coverage, and it is done in a pre-process. The drawback is that the resulting rendering will ensure a full coverage only for a small number of viewpoints, for instance nearly forbidding zooming operations. Note also that the set of attributes have to be carefully chosen so that strokes do not "stick out" of the silhouette; In practice, Meier defines orientation as a vector field aligned with surface silhouettes and uses rather short strokes. Finally, in order to represent illumination, the author creates various color layers that are embodied by different sets of strokes, which restricts the rendered scenes to static illumination.

Recently, two new approaches extended the work of Meier to deal with the above-mentioned limitations. Sperl [Sperl 04] first extended the method to work in real-time. To this end, he used the recent advances made with graphics cards, and proposed a simple behavior for paint strokes in order to keep a full coverage when the viewpoint moves: Their thickness grows during "zooming in". This behavior, however, is in conflict with the idea of paint strokes lying onto the picture plane. We also use the GPU to create interactive painterly animations, but we propose a more elaborated scheme to deal with density.

Another approach has been proposed by Chi et al. [Chi 06]. They use a hierarchy of bounding spheres to distribute their particles, and thus allow for a dynamic density management at runtime. They treat density events (insertion and deletion of particles) by proposing smooth transitions made possible by Tonal Art Maps [Praun 01]. Moreover, in order to ensure that the final strokes do not cross any texture or object boundary, they compute a color segmentation in pre-process that is used to direct the bounding sphere hierarchy and they use maximum curvature directions to orient strokes. We take inspiration from their work for density management, but without restricting attributes to any precomputed values. The resulting system thus offers a broader range of painterly styles and is fully dynamic: Both viewpoint and illumination can vary, and the rendering will adapt to those changes.

3.2 Overview

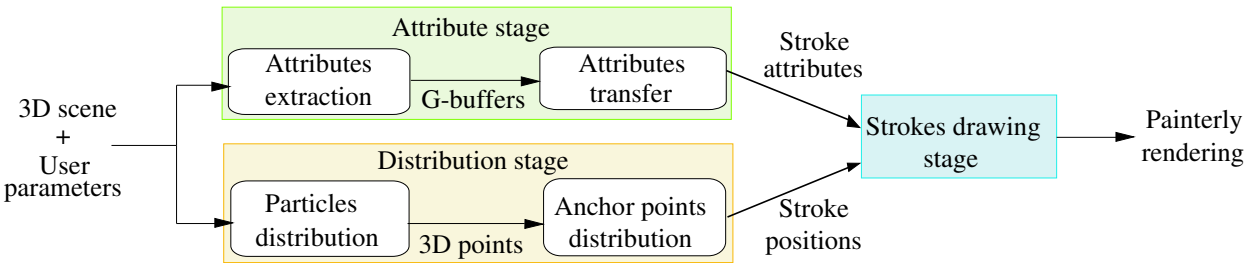


Figure 3.2: Our painterly rendering pipeline. The scene to be rendered is passed to two parallel stages: one transfers attributes from the scene to strokes attributes; the other distributes anchor points in the picture by projecting particles from the 3D object. Finally, strokes are drawn at anchor points locations using transfered attributes, while keeping color and depth cues.

We present an object-space, particle-based system that extends Meier's approach

[Meier 96]. Our main contribution is a fully view- and lighting-dependent behavior that explicitly performs a trade-off between a user-specified stroke style and the faithful representation of the depicted scene. This way, our system offers a broader expressiveness to the user by removing the constraints usually found in previous approaches.

In order to draw a stroke, the system needs to set its 2D position and its attributes (color, orientation, size). Our pipeline therefore consists of three main stages: attributes, distribution and stroke drawing (see Figure 3.2).

The attributes stage (see Section 3.3) takes as input the 3D scene and computes the scene attributes using G-buffers [Saito 90]. Attributes such as normals or curvature can be used to control stroke attributes, e.g. orientation and thickness. Moreover, our painterly system works at a specific scale. Indeed, we mimic the use of a paint brush: since details smaller than the brush thickness cannot be represented, we propose to remove them and we call this minimum detail size the painting scale.

The distribution stage (see Section 3.4) takes as input the 3D scene, distributes 3D particles over the scene surfaces and for each frame projects a subset of the particles to compute the strokes positions while ensuring a complete coverage of the image. This way our system allows to control stroke density by means of a view-dependent particle distribution.

Finally the stroke drawing stage (see Section 3.5) takes as input the 2D strokes positions, attributes and a set of user-defined parameters: length, bending, orientation, texture, etc. Strokes are drawn starting from their 2D position with attributes taken at this 2D point in the attributes G-buffers or set globally by the user. As strokes attributes remain constant over the drawing, we ensure that the important details of the scene are kept by cutting off the strokes whenever the underlying attributes become too different from the starting ones. To measure this incoherence we use a user-specified threshold that controls the maximum allowed distance between starting position attributes and current position attributes.

Compared to Meier's work, each stage provides improved functionality: A large freedom for the choice of strokes attributes, a view-dependent density management, and a faithful representation at a given painting scale. Most of the work is done on the GPU allowing to obtain a dynamic painting algorithm that bears some similarities with traditional painting techniques and runs at interactive frame rates.

In the following we present each stage in detail before showing some results and discussing our method. While I implemented the attributes stage of Section 3.3, David Vanderhaeghe is responsible for the development of the distribution stage in Section 3.4, and we combined together the outputs of each stage into the drawing stage of Section 3.5.

3.3 Attributes stage

How scene attributes are mapped to picture attributes defines what the paint strokes represent. We thus provide the user with a set of modules, implemented with shaders on the GPU, to design this pipeline stage. As already mentioned, rendering a scene with paint strokes has the effect of abstracting the representation: details smaller than a user-defined painting scale that we denote ε cannot be represented and should be omitted. This scale is set by the user and represents the maximum thickness of the

painting strokes. Color and depth are therefore treated in a specific way: since they will be used to guide the stroke painting process they must be abstracted to meet the scale ε. Other attributes can either be chosen by the user or computed from the 3D scene information. We show how to control strokes orientation and thickness from normal or curvature properties; but a benefit of our approach is that the attribute module can be extended to other mappings and the system will still produce faithful representations of the scene.

3.3.1 Color and depth

Our paint strokes take their color from the underlying scene. Before being used in the stroke drawing procedure, the color and depth attributes of the scene are abstracted so that details smaller than ε are filtered out. This ensures a correct depiction of the shape and reflectance of an object by cutting off the strokes when color or depth highly vary.

(a) (b) (c) (d)

Figure 3.3: Filtering the color attribute. (a) Color buffer with and without blur. (b) Painterly rendering without blur. (c) Painterly rendering with blur. (d) A zoom of (b) and (c).

For color, we use simple materials such as Phong or toon shaders; However, our method is independent of the chosen material, thus more complex shaders could be used, like the extended toon shader described in the next Chapter. For depth, we use depth in eye space: as opposed to Z-Buffer that is perspectively distorted due to projection, we prefer the depth values to be linearly distributed in eye space. It makes the control of the depth attribute easier to the user. We render these attributes to G-buffers that are then filtered using a blur of size ε, computed using a box kernel in a pixel shader. Figure 3.3 shows the obtained results for the color buffer. Note that the filtering is only a post-process of the attribute mapping. Therefore other filters could be added using color processings such as aerial perspective, or depth processings such as segmentation in depth planes.

3.3.2 Other attributes

Other stroke attributes such as length, orientation, thickness, bending or wiggling are set by the user as global parameters. These attributes are then optionally jit-

tered per stroke in order to add randomness to the composition, as proposed by Meier [Meier 96]. However, instead of being set globally, they can also be mapped from some scene attributes.

As shown by previous studies [Interrante 97, Girshick 00], strokes orientation can convey a lot of information about properties of the depicted surface. Following the previously mentioned work, our system allows the user to set orientations based on projected maximal principal curvature directions or projected normals. Similarly, thickness can be modified based on surface attributes such as slant (dot product between the view vector and the normal vector) or Gaussian curvature (see Figure 3.4). However, the thickness attribute is constrained to values smaller than ε, unless it would invalidate color and depth filtering.

The stroke orientation can be assigned either a direction perpendicular to the projected normal in order to locally align the orientation with the object's silhouette; or to the projected principal curvature. Note that when one projects a vector in eye space (in our case, a normal or principal curvature vector) perpendicular to the picture plane, the resulting projected vector is null, hence giving no orientation information. We thus modulate the length attribute so that it is equal to the thickness attribute when the projected vector is null. Once projected onto the picture plane, the user can also uniformly rotate the resulting orientation by hand.

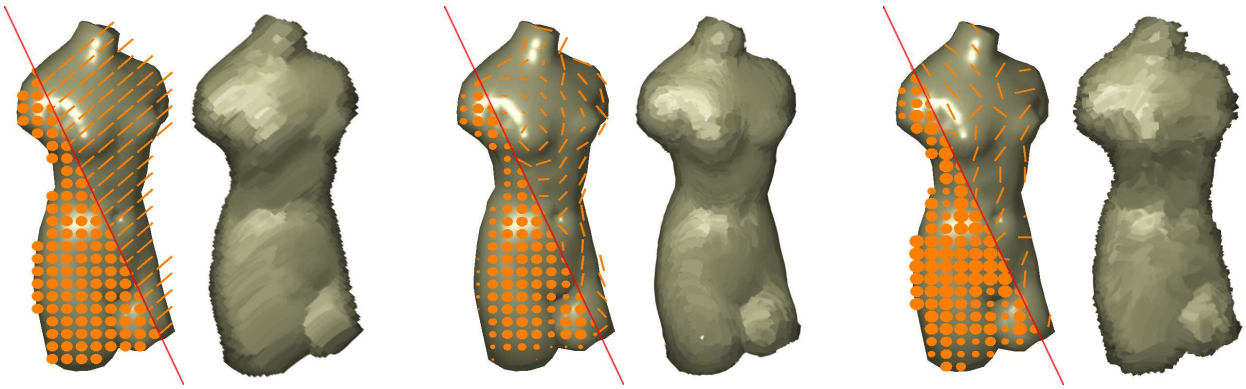


Figure 3.4: Orientation and thickness can be assigned either a global value (a), or computed from the surface normal (b) or curvature (c). For illustration purposes, we use on top artificial strokes (dots for thickness and segments for orientation) distributed along a grid in image space and show the final result with strokes.

The stroke thickness can be assigned either the slant or the curvature magnitude, normalized between 0 and 1. The thickness is thus reduced along silhouettes or in areas of high curvature. However, reducing the thickness size will have an impact on the painting coverage, as discussed in Section 3.4. Like with orientation, the user can finally alter the weight of this modulation, hence controlling the minimum thickness values in the rendering. Note that our curvature extraction suffers from the resolution of our input meshes, and we plan to use radial basis function to smooth it out in the future.

3.4 Distribution stage

From a given viewpoint we want to distribute enough strokes in the image to cover the represented 3D scene. We also want to ensure a temporal coherence of the strokes, that limits as much as possible the appearances/disappearances of strokes. In this stage, our goal is then to distribute particles on a model so that, when projected in the picture plane and embodied by strokes, the resulting painting ensures the coverage requirement.

Our method works in two steps: First we compute a spatial hierarchy per object of the scene that ensures a nearly uniform spatial distribution of the 3D particles. Second we select the 2D points (projected particles) necessary to cover the image from a given viewpoint by traversing these hierarchies.

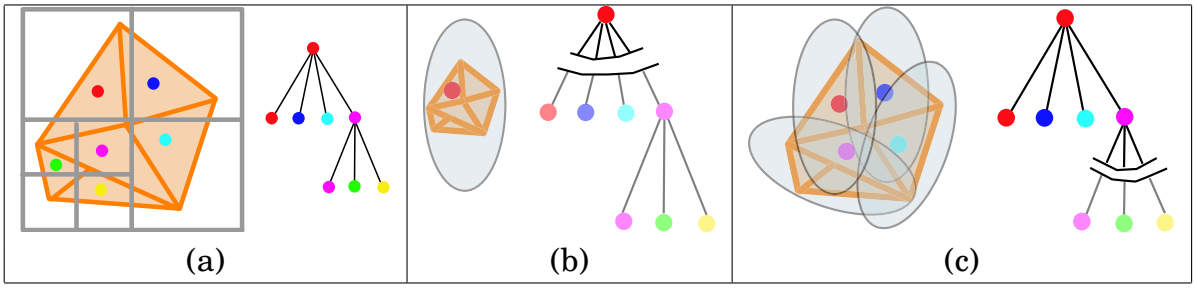


Figure 3.5: (a) In a preprocess, 3D particles are distributed over the mesh and ordered in a hierarchy using an octree. (b,c) At each frame a subset of particles are selected by cutting the hierarchy so that drawn strokes cover the represented objects. In (b) one particle is enough whereas in (c) one more level of the tree has to be selected or added via subdivision to cover the object.

3.4.1 Hierarchy construction

Several methods exist to build a spatial hierarchy of a 3D object. In our case we want to be able to treat any kind of 3D meshes (without imposing a particular resolution for example). However, the coverage requirement forbids us to compute a static hierarchy because we do not know in advance how many particles will be needed. Previous work [Pastor 03, Cornish 01] already addressed a similar question by simplifying and subdividing a mesh to compute a hierarchy of particles. However they both built the hierarchy as a preprocess and thus were not able to treat all the possible viewpoints. Moreover their mesh simplification procedures did not ensure that the particles of a given level would uniformly cover the mesh.

Our method overcomes these limitations by using a different simplification procedure and a runtime subdivision mechanism. One particle is created at the center of each triangle of the mesh, and is slightly jittered. Then an octree is built until each leaf contains a single particle. A particle is chosen for each internal node among the particles of its children (see figure 3.5(a)). In practice we choose the particle that is the closest to the center of the corresponding octree cell. Each cell is then resized to fit the bounding box of the corresponding subtree leading to a bounding box hierarchy. This is done in a preprocess and offers a quasi-uniform spatial distribution of the particles at each level of the hierarchy. Then at each frame, when a single triangle

cannot be covered by a single stroke, we add new particles by subdividing this triangle. In this way we obtain a continuous level-of-details of particles and are able to add as many particles as needed. For a given mesh, the resulting hierarchy is then a tree where each node and its subtree correspond to a subset of the object that can be represented by the node's particle.

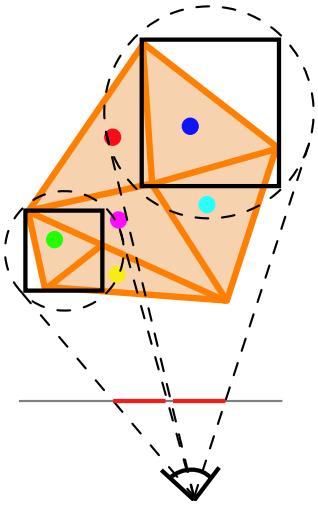
3.4.2 Anchor points selection

At each frame, the hierarchy is traversed in level-order to find a cut in the tree that guarantees that all leaves will be covered by the drawn strokes (see Figure 3.5(b,c)). No visibility computation is made at this point because it is not required by our drawing algorithm, as explained in the following section.

The traversal is made by means of a queue. The root node is first put into the queue and then for each child node, if the corresponding particle *is needed* the node is added to the queue. The algorithm ends when no more node needs to be added to the queue. The resulting truncated tree (consisting of the selected nodes) corresponds then to a view-dependent description of the depicted object.

Deciding when a particle *is needed* while ensuring enough coherence is not an easy task. Recall that a node and its subtree correspond to an object part. Ideally, a node should be selected if the projection of the object part is not already covered. However, drawing exactly the minimum number of particles such that the strokes cover the image is very unstable when the viewpoint is moving: a small change in the viewpoint can reveal many uncovered portions of the surface, and thus trigger the appearance of new strokes (the inverse problem occurs with disappearances). On the other hand, drawing too many strokes does not give satisfying visual results due to exaggerate overlapping: the strokes, then, might not be distinguished.

We thus chose a conservative method that overestimates the number of drawn strokes but decreases the popping artifacts. A node is selected when the size of the projection of its subtree is greater than the user selected scale ε (corresponding to the maximal stroke thickness).



In practice we compute the diameter of the projected bounding box of the subtree and keep the node if this diameter is greater than $k\varepsilon$, with k a factor set by the user that controls the global coverage of the image. Indeed, in a given bounding box the particle can theoretically be anywhere, thus a stroke of thickness ε will surely cover a box of diameter $\varepsilon/2$. Therefore, $k = 0.5$ ensures a complete coverage. In practice $k \simeq 1$ is enough because the particles are in general near the center of the corresponding bounding box and $k > 1$ gives a partial coverage. Note that when strokes thickness is reduced in the attribute stage to values smaller than ε, the coverage property cannot be met anymore; however, the user is still able to manipulate k to get a satisfying distribution.

Our distribution algorithm allows us to manage the density of strokes in image space at runtime, as shown in Figure 3.6.

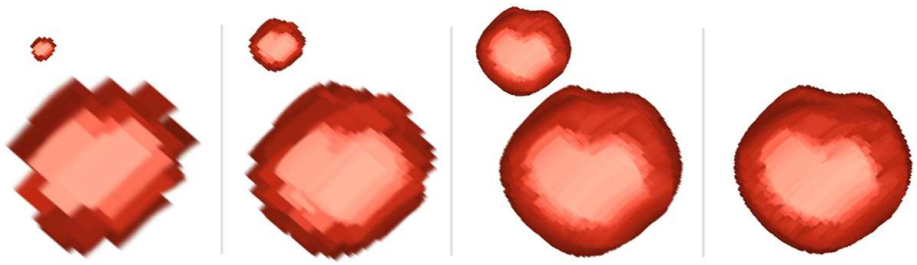


Figure 3.6: When we zoom on an object, the density of strokes used to render it should remain constant; Thus new strokes are introduced. Top row shows the apple model at the proper scale and we also show the expanded image for illustration purposes.

3.5 Strokes drawing stage

The last stage of our painterly pipeline consists in drawing strokes at locations defined by the anchor points, using the underlying picture attributes and stroke style.

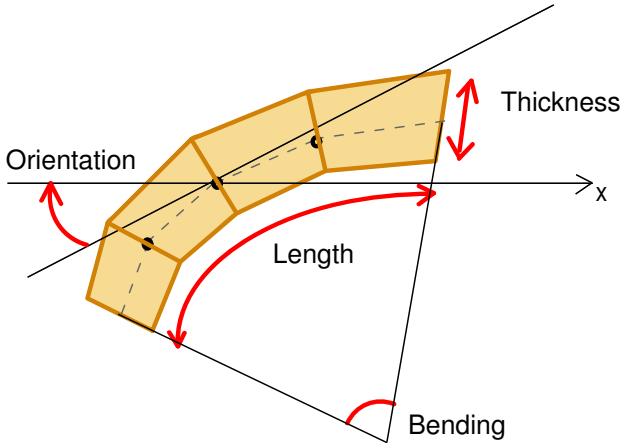
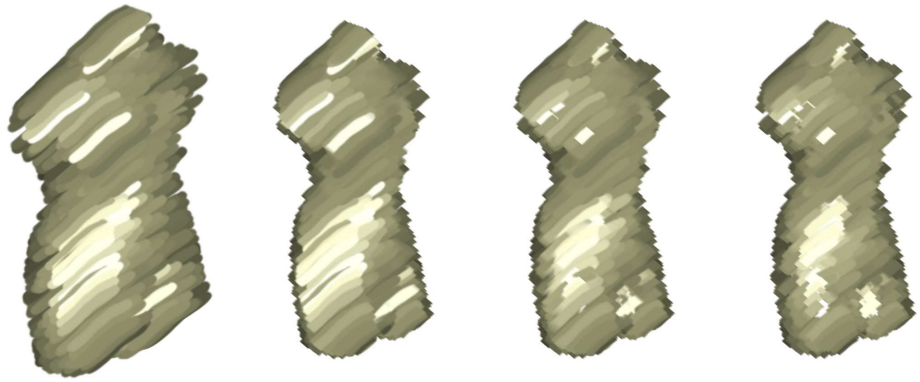


Figure 3.7: A stroke is modeled by a quad strip that is deformed to meet a set of attributes.

In our system, a stroke is defined by a set of six attributes: color, orientation, thickness, length, bending and an alpha texture. The attributes coming from the scene are taken in the G-buffers at the anchor point position: Color is looked up from the abstracted colors; Orientation and thickness can come from picture attributes or be set uniformly by hand like other attributes. Additionally, for each of these parameters, jittering can be applied per stroke in order to add randomness to the composition. In practice, a quad strip is passed to a vertex shader that deforms it to meet the required attributes (see Figure 3.7).

Sorting particles in depth order and drawing them using our procedure results in paintings similar to that of Meier's system: if the density of strokes is high enough, then strokes coming from front facing particles will naturally cover back-facing ones. However, as shown in Figure 3.8(a), depending on the choice of strokes attributes, the resulting painting may miss many of the scene details like silhouettes, bumps, highlights or texture patterns. To overcome this limitation, we introduce a novel



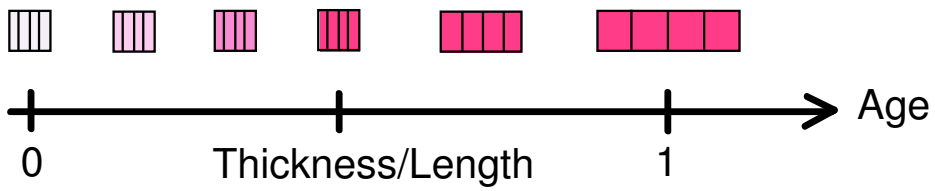
(a) (b) (c) (d)

Figure 3.8: Drawing the strokes without any cut off (a) hides some important details of the object. Whereas applying a cut off on depth (b) or color (c) allows to obtain a faithful depiction of the scene. (d) shows the use of both depth and color cutoffs.

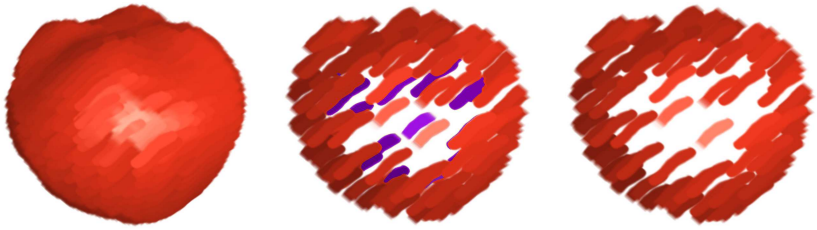
painting behavior that maintains color and depth cues coming from the scene. We first pick a reference pair of color and depth attributes under the stroke's anchor point. Then, growing outward from the stroke center in both directions, we compare the color and depth found under the stroke's backbone to the reference color and depth. If one of the color difference (computed in Lab color space) or depth difference is greater than a user-defined threshold, we fade out the stroke. This is illustrated in Figure 3.8(d). The benefit of this technique is to offer a compromise between the style specified by the user and the scene information to represent. It is performed in a vertex shader on the GPU, but due to current limitations of the graphics hardware (vertices can only be modified in a vertex shader, not created), we could not implement a real stroke marching method. Instead, we evaluate the color and depth differences for each vertex independently, which might give rise to truncated strokes. But in practice, we found this approximation to give satisfying results.

Another difference with Meier's work is that we adapt the distribution of strokes depending on the viewpoint; this has the effect of making strokes appear or disappear when an object comes closer or farther from the camera. These density events will inevitably produce poppings if we do not ensure smooth transitions. We thus add a new behavior to our dynamic painting algorithm: each stroke is attributed an age parameter that goes from 0 to 1 after it appears, and inversely from 1 to 0 after it disappears; then this age is used to temporally fade in or out the stroke.

We illustrate it for a fade in, the fade out being simply the reverse process:



Strokes length is first reduced to their thickness and their alpha is increased from 0 to 1 while age goes from 0 to *thickness*/*length*; then, they are grown outward while age goes up to 1, where they reach their full length. The overall aging process is, again, done in our vertex shader.



(a) (b) (c)

Figure 3.9: (a) Producing a complete coverage without using visibility gives good results. (b) However with a partial coverage, back-facing strokes (in purple) appear if no visibility is computed. (c) We thus add a fuzzy visibility test to discard these unwanted strokes.

Finally, another property of our strokes distribution is that it does not necessarily have to fully cover the depicted object; indeed, interesting results can be obtained with partial coverage, as shown in Figure 3.11. However, in this case, the gaps between the front-facing strokes will let back-facing ones appear; and they will be even more noticeable during animations since back-facing particles motion is usually the opposite of front-facing ones. In order to discard these unwanted strokes, we implemented a fuzzy visibility mechanism: the idea is simply to keep strokes whose depth is within a fuzzy interval around the z-buffer value found under its anchor point. Such a technique is more robust than computing an exact visibility because of possible numerical errors between the particle's depth and z-buffer values. We recently discovered that a similar, but more elaborate method has been proposed by Luft and Deussen in their watercolor rendering system [Luft 06]. However, we found our simple approach to be efficient enough for the purpose of our painterly renderer, see Figure 3.9. Note that visibility events produce problems similar to density events; we thus apply the same aging process in order to smooth them out.

3.6 Results

We show here more results that illustrate some of the styles achievable by the current version of our system. Depending on the scene complexity, the frame-rate ranges between 1 and 20 fps; The bottlenecks of the pipeline being the view-dependent anchor point selection when models are highly subdivided, and the drawing procedure that slows down when using thin strokes. Some poppings remain during the animation; They are mainly due to aliasing problems when an attribute is read from a G-Buffer. Moreover, our drawing procedure is not a real stroke marching algorithm, thus our solution might lead to unstable behaviors, even if they are lessen by the use of blurred color and depth buffers.

We first explained in Section 3.4 how our system was able to produce painterly renderings view-dependently by means of a density management; We now show in Figure 3.10 how it can adapt to lighting variations over the depicted objects. From left to right, the Venus model is illuminated from the left, top right and bottom. Note how the strokes adapt in length to depict big or small scale lighting features.

Figure 3.11 illustrates, on a same model, how various stroke parameters can be chosen. On the left, the character is rendered with bent strokes that follow silhouette

lines in order to emphasis the shape. While in the middle, we use short strokes which thickness vary with Gaussian curvature; And on the right, long strokes are oriented globally in a diagonal direction in order to abstract the shape.

Finally, different kinds of paint media can be suggested by our system by means of strokes alpha texture. Figure 3.12 shows three different textures that give very different painting results. As already noted, strokes parameters should be jittered in order to be able to perceive them. An important parameter is color, and in order to stay close to the scene colors, we employ a jittered color in Lab color space. The effect of jittering is illustrated in Figure 3.13.

Figure 3.10: When the light is dynamically moved by the user, the strokes color and length adapt to depict features such as highlights or self-shadowing. From left to right, light is positioned at left, top right and bottom.

Figure 3.11: A same model rendered in three different styles. Left: Strokes orientation is defined from surface normals and strokes are bent along their orientation. Middle: Short strokes are distributed with a thickness varying with Gaussian curvature. Right: Long diagonal strokes are used to fill in the model.

Figure 3.12: Varying the strokes alpha texture can greatly influence the final look of the rendering. From left to right, we used scanned textures of pencil, thick paint and dense watercolor.

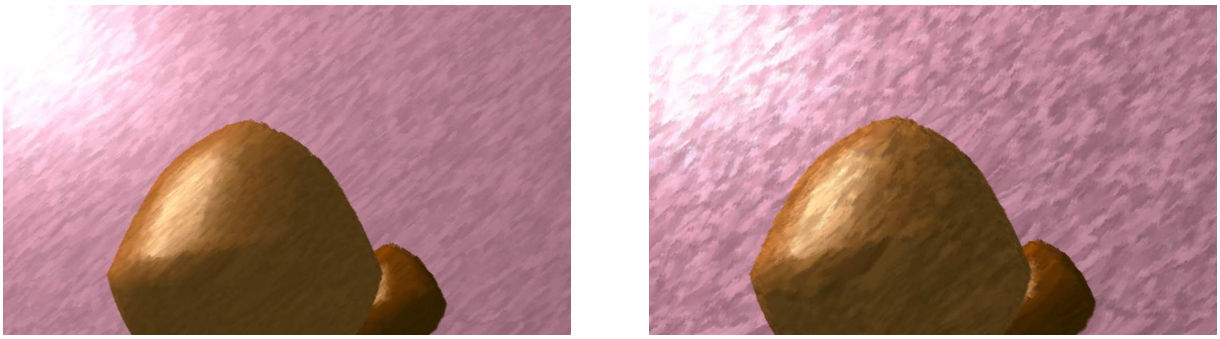


Figure 3.13: On this simple scene inspired from Meier's original paper, we show how jittering can reinforce the perception of strokes. The right image is rendered with approximately twice as much jittering as the left one.

3.7 Discussion and future work

Our system could be extended in different ways. First, as shown by Meier, separating strokes into layers before compositing them greatly augments the creative capabilities of such a pipeline, with different layers for different scales and/or for different illumination effects. The later requires to be able to incorporate new reflectance cues like highlights or shadows in the system. Such an extension will be easily added by means of the extraction of new G-buffers during the attributes stage. In the general case, it would be ideal to have a generic method to transfer attributes from the scene to the picture plane, and the extended toon shader presented in Chapter 2 is a first step towards this generic system.

Another area of future work resides in the distribution of anchor points from a given point of view. Our current mechanism is similar to splatting techniques (see for instance [Rusinkiewicz 00]). It does only part of the job though, since ensuring color and depth cues as we draw strokes is crucial to the quality of the results. A disadvantage of our approach is that it tends to place more strokes near the silhouettes of objects: this is because we do not take surface orientation into account during the distribution stage. Taking inspiration from the splatting literature would help us to

better control the strokes density. In the long term, it would be also very interesting to control non-uniform densities across the picture plane.

Our drawing algorithm could be further extended: In its current version, it only deals with simple depth and color cues. But it could also have more complex behaviors. In particular, we are interested in the painterly rendering of large outdoor scenes, and with these types of environments, it would be promising to consider effects such as aerial perspective. Another requirement for the real-time rendering of large scenes in general is the improvement of the performance of our system: many optimizations are still possible. Moreover, like in Meier's approach, we relied on the alpha texture of each stroke to blend them in the final rendering. It not only hides the poppings that occur when two adjacent strokes switch their depth order, but also gives a natural impression of paint matter. This is enough in the context of a simple paint medium, but would require more attention for other media such as watercolor [Luft 06] or thicker paint [Hertzmann 02a].

Finally, we would like to point out that our approach is, like Meier's renderer [Meier 96] or FreeStyle [Grabli 04b], an instance of the four-representation systems proposed by Willats and Durand (see Chapter 1). The three stages of our pipeline correspond to the attributes, primitive and mark stages (for the spatial stage, we only use perspective). One important aspect of their study is that depiction can be characterized as an optimization process, between the goals imposed by a style and the information to carry from the scene, as will be explained in Chapter 5. In our work, there is no such optimization, but rather a trade-off than can be controlled by the user. In a future work, we would thus like to study how strokes position, orientation and shape can be determined from an optimization between some scene information to represent and a user-supplied style provided, for instance, by means of a sample painting.

Chapter 4

An alternative metaphor: dynamic 2D patterns

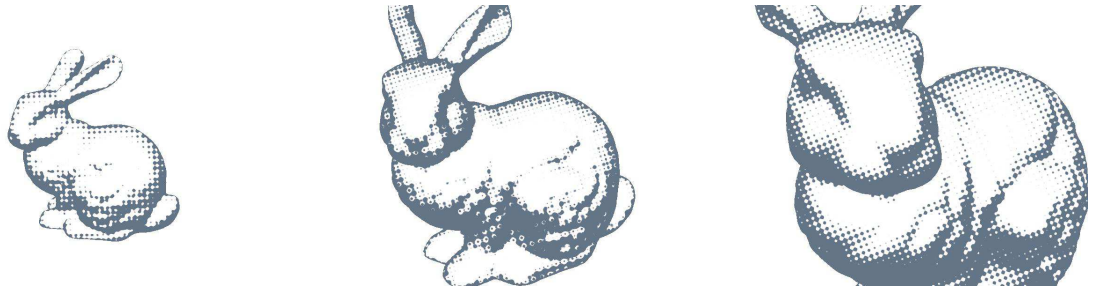


Figure 4.1: The Stanford bunny rendered with a 2D halftone pattern that moves, scales, and adapts its density as the camera changes.

While Chapter 3 described a particular instance of the representation model, there are many other possible approaches, as discussed in Chapter 1, that will lead to many different expressive renderings. Compared to previous work on stroke-based or continuous tone techniques, very little work has been done on representations using stroke patterns. The metaphor we consider here is even more seldom encountered in expressive rendering papers, while at the same time it is widely used in comics and illustrations. Our objective is to represent the tone and motion of a 3D object via *image space* hatching or half-toning patterns. As we will see in the following, this is an ”extreme” metaphor in the sense that it is over-constrained.

Discussions about this work first started during my EURODOC internship at the University of Michigan, with Professor Lee Markosian, his student Simon Breslav and my co-advisor Joëlle Thollot. While we continued discussing about the project after my internship ended, its practical implementation have been performed by Lee Markosian and Simon Breslav. I included this work in my thesis because it allows me to present an alternative metaphor, and to relate it to the previous chapters of this part on expressive animations. I will thus present the main motivations and contributions, some details about the current implementation, as well as preliminary results of the system; but keep in mind that, at the time of writing this thesis, it is an ongoing project that still needs some development before being submitted for publication.

Previous work on this type of rendering (*e.g.*, [Freudenberg 01, Hertzmann 00, Interrante 97, Kalnins 02, Praun 01, Winkenbach 94]) has proposed *object space* pat-

terns, in which strokes or halftone masks are mapped onto 3D surfaces in the scene. Consequently, their *shape* provides visual cues about the shape of the underlying object. In contrast, image space patterns are parameterized within the picture plane and so do not follow the underlying 3D shape. Cues about 3D shape must then come exclusively from tonal variations in the patterns. An example is parallel hatching marks that follow a constant image-space direction.

Why prefer image space patterns? An important reason is that they provide greater visual abstraction, in a sense "flattening" the shape and de-emphasizing small details. This point is especially clear from a casual glance at traditional drawings, illustrations, and comics. Artists often perform shading using stroke or halftone patterns that only loosely follow the 3D shape. In many cases even a loose connection is absent – strokes instead follow an image-space layout with no particular relation to the shape of the shaded object.

In addition to these aesthetic considerations, in computer graphics another consideration is code complexity, ease of authoring, and run-time overhead. Methods based on object-space stroke patterns must address several challenging problems: determining good-quality direction fields for strokes (and hiding the inevitable singularities that can occur), adapting to anisotropic scaling due to perspective foreshortening, and dynamically varying the number of rendered strokes to account for camera zoom. These problems are much simpler to deal with for image-space patterns.

For still images, there is no particular difficulty in shading with image space patterns. The pattern can first be generated so that it completely covers the model in image space, then strokes can be clipped wherever they lie outside the model, and their parameters can be adjusted according to the desired tone at each location. For example, stroke widths, colors, and opacities can easily be adjusted to reflect the desired tone at each image-space location. This was the approach taken by Ostromoukhov [Ostromoukhov 99], for example, who also noted the aesthetic limitations of purely object-space stroke patterns.

The main difficulty of using image-space stroke patterns arises when image *sequences* are rendered, either for animations or real-time interaction. Then, if the image-space stroke pattern does not move with the object, the so-called "shower door" effect is apparent: the viewer experiences a disturbing disconnect between the motion of the underlying shaded object and the fixed stroke pattern.

Our solution is to compute a 2D "similarity" transform that combines translation, rotation, and uniform scaling to match (as closely as possible) the motion of the underlying 3D object. Of course, no such 2D transform can perfectly match the apparent motion of the 3D object, so we use a least-squares technique to compute the similarity transform that best matches the apparent 3D motion.

At each frame, we apply the computed similarity transform to the image-space pattern. When the scaling factor falls outside of a specified range, we initiate a transition from the scaled version of the pattern to an unscaled copy, retaining the current rotation and translation. This transition is carried out over a short time interval by blending or morphing between the two copies of the image-space pattern.

This method accommodates apparent motion that is due to animation of either the camera or the underlying shaded object (or both), and works well for a range of motions. In particular, it closely matches motions that appear in image space as translations, 2D rotations, and uniform scaling. These correspond, respectively, to camera panning (or object motion parallel to the film plane), rotation about the line of

sight, and camera zoom (or object motion toward the camera). It produces plausible results for other motions, such as rotation of the 3D object around an axis that is parallel to the film plane, or for limited motion of animated objects.

We demonstrate our method on several types of image-space patterns, namely halftoning, parallel hatching, and a simulation of media on rough paper (where the image-space pattern is the paper texture). The least-squares solution of the similarity transform is efficient, and the rendering effects are implemented on the GPU. The result is a new class of dynamic image-space patterns that minimize the shower door effect for a wide range of motions, are easy to author, and render at real-time rates.

4.1 Previous Work limitations

Cunzi et al. [Cunzi 03] address a closely related problem, and propose a similar solution. They observe that many non-photo-realistic rendering methods for 3D scenes employ media simulation techniques that make use of a background paper or canvas texture. When this texture is fixed within the rendering window, object motion over the static canvas induces a disturbing "shower door" effect. Their solution is to transform the canvas via a "similarity transform" (consisting of 2D translation, rotation and uniform scaling) to match the image-space motion of certain 3D points in the scene. They compute the similarity transform in closed form based on known camera motion, and apply an additional warping step to better match the motion of the 3D points when the camera rotates. They handle camera zoom via a procedural canvas model that is self-similar under arbitrary scaling.

This method has two important limitations: first, it works best in the limited case of a static scene in which objects all lie at roughly the same distance from the camera; and second, that distance is a parameter that must be supplied by the application. In contrast, our method is automatic and handles both camera and object motion. It can be applied independently to each object in the scene, resulting in a different similarity transform for each object.

The methods of Kaplan and Cohen [Kaplan 05] and Bousseau et al. [Bousseau 06] also address the problem of computing a dynamic canvas for temporally coherent media simulations. Like ours, these methods account for camera motion as well as objects that move independently. Both methods track seed points on 3D surfaces in the scene and use them to generate on the fly a canvas texture composed of small-scale structures that follow the seed points. These methods are limited to paper and canvas textures and do not handle more general types of patterns (*e.g.*, hatching and halftoning) made up of large-scale structures that are readily perceived by the viewer. Our method does handle these more general patterns, as well as those representing paper and canvas textures.

Eissele et al. [Eissele 04] describe a method for rendering 3D scenes with temporally coherent 2D half-toning patterns. They compute per-pixel optical flow, then break the 2D halftone screen into small pieces that move independently with the flow. Where excess distortion builds up in the pattern they blend back to an undistorted piece of the pattern. The resulting animations show significant temporal artifacts as distorted areas appear and disappear in patches across the surface.

4.2 Computing the transform

The method works as follows. In a pre-process we distribute sample points over the 3D surfaces in the scene. At run time, we project the seed points to image space and compare their current positions to those in the previous frame. From these two sets of image-space points (current and previous) we compute a similarity transform that best matches the observed motion of the sample points. We then apply this similarity transform to whatever 2D patterns are being used to render the given object. We handle multiple objects by considering the sample points of each object independently.

We compute the similarity transform from the samples using the least-squares method of Horn [Horn 87]. The same method was used by Wood et al. [Wood 97] to compute a warp between consecutive frames of an image sequence in order to build a multi-perspective panorama. The least-squares solution is efficient, running in $O(n)$ time, where n is the number of sample points. In practice, a few dozen points (or fewer) are sufficient to produce good results.

Wood et al. use just 9 sample points each frame, evenly distributed within the rendering window. They use a ray test to find the 3D surface location corresponding to each sample point, then find its previous image-space location using the camera parameters of the previous frame.

To avoid ray tests, we use a fixed set of 3D sample points distributed over each surface. Each frame, we consider only those samples that lie on front-facing surfaces in the view frustum, in both the current and previous frame. To reduce the chance of finding too few sample points in any frame, we use a moderately dense set of samples for each object.

4.2.1 Generating samples

We generate sample points using the "stratified point sampling" method of Nehab and Shilane [Nehab 04]. This method achieves a relatively uniform distribution of points over each 3D surface, independent of triangulation. Parameters can be tuned to control the density of the resulting sample points. We use a fixed set of parameters that results in several dozen to several hundred sample points for each object, depending on its shape. While this is more samples than needed, the solution still runs at interactive rates, and using more samples reduces the likelihood of finding too few usable samples in any given frame.

Finding too few samples can happen if nearly all samples lie on back-facing surfaces or are off-screen. When this does happen, we relax the condition that samples must be in the view frustum. Instead, we require that they lie in front of the near clipping plane and project to screen space within an expanded rectangle that contains the rendering window. If there are still no usable samples, we set the 2D transform to the identity. We find in practice that when no usable samples are available, the object is always off-screen, so it doesn't matter that the pattern is not transformed.

4.2.2 Least-squares solution

Horn's method computes the 3D similarity transform that best maps one set of 3D points onto another (given a known 1-1 correspondence between points). He represents the similarity transform by a combination of translation and quaternion multi-

plication. Our case is simpler, since the points are all 2D. The transform then becomes a combination of translation and complex number multiplication. We now briefly describe the computations.

We consider only samples that lie inside the view frustum and are front-facing in both the current and previous frames. Let $\bar{\mathbf{c}}$ and $\bar{\mathbf{p}}$ denote the centroids of the image-space locations of the samples in the current and previous frames, respectively, and let $\mathbf{c}_i$ and $\mathbf{p}_i$ denote the 2D location of sample i in the current and previous frames, respectively, each expressed as an offset relative to the corresponding centroid ($\bar{\mathbf{c}}$ and $\bar{\mathbf{p}}$). Treating $\mathbf{c}_i$ and $\mathbf{p}_i$ as complex numbers, we want to find the complex number $\mathbf{z}$ that minimizes the error:

$$E = \sum_i |\mathbf{z}\mathbf{p}_i - \mathbf{c}_i|^2$$

Solving via least squares yields:

$$\mathbf{z} = \left(\sum_i \mathbf{p}_i \cdot \mathbf{c}_i, \quad \sum_i \mathbf{p}_i \times \mathbf{c}_i) \right) / \sum_i |\mathbf{p}_i|^2$$

(Here, "$\times$" denotes the "2D cross product.")

It turns out that the above formulation is not symmetric, in the sense that the computed transform from $\mathbf{c}_i$ to $\mathbf{p}_i$ is not the inverse of the transform from $\mathbf{p}_i$ to $\mathbf{c}_i$ – in general, the two rotations are inverses, but the two scale factors are not. We thus adopt the "symmetric" formulation described by Horn by multiplying $\mathbf{z}$ as computed above by the following scale factor s:

$$s = |\mathbf{z}|^{-1} \left(\frac{\sum_i |\mathbf{c}_i|^2}{\sum_i |\mathbf{p}_i|^2} \right)^{\frac{1}{2}}$$

4.2.3 Mapping to *uv*-space

Each image-space pattern is defined in a canonical uv-space. For an ordinary 2D texture map (such as a canvas texture), uv-space is just the space of texture coordinates. By convention, the pattern fills the unit square $[0,1]x[0,1]$ in uv-space. (We assume the pattern tiles seamlessly to fill all of uv-space.) To render the pattern, we need to map image-space locations to uv-space. For each pattern (or set of patterns that move together) we thus store an image-space point $\mathbf{o}$ that maps to the origin in uv-space, and image-space vectors $\mathbf{u}$ and $\mathbf{v}$ that map to uv vectors $(1,0)$ and $(0,1)$, respectively. In each frame we compute $\bar{\mathbf{p}}$, $\bar{\mathbf{c}}$ and $\mathbf{z}$ as above, then update $\mathbf{o}$, $\mathbf{u}$ and $\mathbf{v}$ as follows (treating them as complex numbers):

$$\begin{aligned} \mathbf{o} &\leftarrow \bar{\mathbf{c}} + \mathbf{z}(\mathbf{o} - \bar{\mathbf{p}}) \\ \mathbf{u} &\leftarrow \mathbf{z}\mathbf{u} \\ \mathbf{v} &\leftarrow \mathbf{z}\mathbf{v} \end{aligned}$$

Given an image-space point $\mathbf{q}$, its uv-coordinates are then simply $((\mathbf{q}-\mathbf{o})\cdot\mathbf{u}/|\mathbf{u}|^2, (\mathbf{q}-\mathbf{o})\cdot\mathbf{v}/|\mathbf{v}|^2)$. This uv-coordinate can be used with any 2D pattern, causing the pattern to track the apparent motion of the object in image-space.

4.3 Rendering patterns

We now briefly describe shaders that render dynamic image-space patterns using *uv* coordinates computed by our method. We created a dynamic canvas using the "paper effect" algorithm of Kalnins et al. [Kalnins 02], a half-toning shader based on the method of Freudenberg et al. [Freudenberg 01], and a simple hatching shader that renders parallel lines that adapt to the tone of the underlying 3D surface. We augment the rendering with silhouette strokes that we generate in software using an approach like that of Kalnins et al., The hatching strokes, halftone patterns, and paper effect are implemented as GLSL vertex and fragment shaders [Rost 04]. Our program computes **o**, **u** and **v** each frame and sends them to the shaders. Most of the computation takes place in the fragment shader, which converts the pixel location to a *uv* coordinate and uses that to determine the output color for the fragment.

Each fragment shader can determine the desired tone at the fragment by looking up the value in a *tone reference image*. This is a simple grey-scale rendering of the scene produced in an initial rendering pass each frame and read from the back buffer into texture memory. Typically we use a toon shader to produce a somewhat abstracted representation of the desired tone.

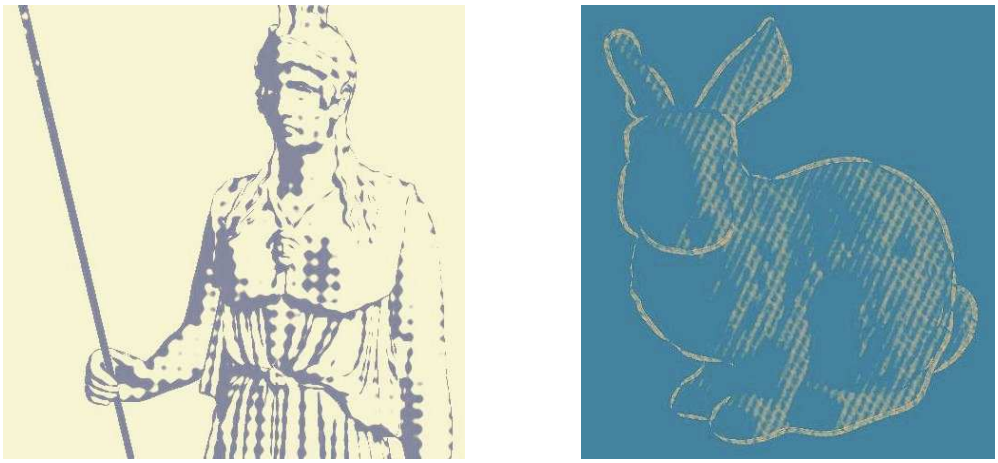
4.3.1 Half-toning

We use the half-toning method of Freudenberg et al. [Freudenberg 01], including the anti-aliasing step they recommend. For Figure 1 we used a procedurally generated halftone screen defined as the average of two sinusoids, one in u and the other in v, with period 1, amplitude 0.4, and vertical offset 0.5. (Using an amplitude less than 0.5 means that nearly black areas receive full ink, and nearly white regions receive no ink.) Given halftone screen value h, desired tone t, and anti-aliasing constant $e = 0.1$, we compute opacity `a = smoothstep(-e,e,h - t)` and use it to alpha-blend the chosen ink color with the underlying fragment color. In this way we can achieve colored inks, as shown in Figure 4.2(a).

4.3.2 Parallel hatching

The parallel hatching method, like the halftone shader, computes an opacity that is used to alpha blend the ink color with the underlying fragment color (usually the color of the background). The shader first maps the fragment's position to *uv*-space, including a scaling that affects line spacing, and a rotation that affects line direction. Then it computes the distance to the nearest canonical line, defined as a line of constant v (for integer values of v). The opacity falls off with increasing distance, and goes to zero if the fragment lies too far from the line. We also vary the opacity with the underlying tone somewhat, gradually reducing opacity as tone decreases. We then apply the paper effect [Kalnins 02] to remap the opacity before blending the ink color with the underlying fragment color. This greatly reduces the mechanical appearance of the lines. Note that both the paper effect and hatching pattern are transformed to the same *uv*-space (with additional separate, fixed scaling). This makes for a more temporally coherent rendering. However, the paper effect can also be applied without hatching.

A useful effect seen in all of our hatching examples is to artificially lighten the tone near silhouettes via a computed virtual "back light" based on the dot product of the surface normal and view vector. (We raise the dot product to a power, *e.g.* 3, to narrow the range of the back light.) This dot product is computed in the vertex program and sent to the fragment shader as an attribute variable. The result is that strokes tend to stop short of silhouettes. Since we draw silhouettes separately (computing them on the CPU), this helps achieve a more natural look. We achieve highlight strokes by simply choosing a light-colored ink, a medium or dark background color, and setting lights to "shine" where we want *darkness* (see Figure 4.2(b)).



(a) (b)

Figure 4.2: (a) A single halftone pattern is applied to this statue model, with the size of the dots varying according to tone. (b) A hatching pattern is applied to the bunny model, with each stroke thickness following the underlying tone, but using a light-colored ink.

The algorithm described above is implemented as a shader subroutine. To achieve layers of hatching, we run the subroutine more than once. The subroutine takes parameters such as line width and spacing, line direction, and stroke color. The fragment shader computes the opacity value for each layer in sequence, compositing the desired ink color with the underlying fragment color using the opacity computed at each pass.

4.3.3 Level of detail

When patterns are scaled very large or small they lose their original appearance. We support a kind of "level of detail" whereby patterns are restored to nearly their original size when the cumulative scale factor falls outside of a given range. To achieve this, we track the cumulative scaling s applied to the pattern in the current frame. We set two thresholds, $s0$ and $s1$. If $s < s0$ we leave the pattern as it is (scaled by s). If $s > s1$ we apply scale factor $s/2$ to the pattern (instead of s). For $s0 < s < s1$ we compute a corresponding interpolation parameter $t = (s - s0)/(s1 - s0)$ and use it to combine two versions of the pattern: one scaled by s and one scaled by $s/2$. For s outside the range $[1,2)$ we map it into that range and do equivalent computations.

For paper textures and hatching patterns, we use t to blend between the patterns (one scaled by s, the other by $s/2$), see Figure 4.3. For half-toning we implemented a

different approach, motivated by the observation that half-toning should to avoid the mid-tones that alpha blending would produce. Instead, we compute the quantity of ink for each layer using the halftone subroutine (using scale factor s and $s/2$), but we divide the tone between the two layers. Initially, the first gets all the tone, but as the transition unfolds the second layer gets more of the tone, until at the end it gets all of the tone and the first layer gets none. The result is a kind of morph from one halftone pattern to the next. The effect can be seen in Figure 4.1. While the intermediate renderings are less regular than the ordinary halftone screen, the effect appears (to us) to work well.

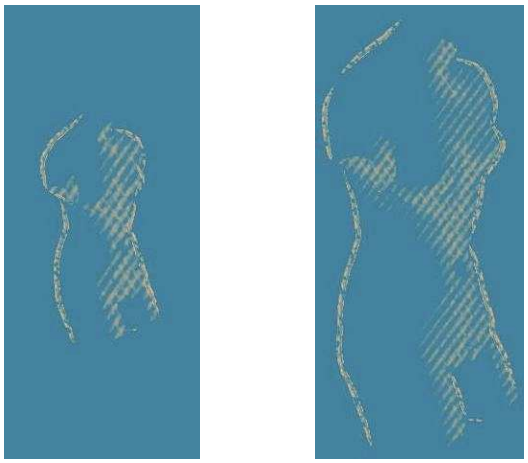


Figure 4.3: When zooming on the model, the pattern changes its level of detail so that stroke thickness stay close to constant in image space.

4.4 Results and discussion

For many motions, especially pure panning and zooming, the patterns appear extremely stable. The rendering time was over 30 fps for all of the examples shown. Our test machine has a 3.2 GHz Pentium 4 CPU and 256MB GeForce 7800 GTX GPU. The total rendering time included separate preparation of two reference images (a tone image copied to texture memory and an ID image copied to main memory and used for rendering silhouettes on the CPU). Model sizes were in the range of 30 - 100K triangles.

We don't claim that 2D patterns should always be used in place of patterns that are mapped onto surfaces in 3D. Rather, we argue that having the option to use such patterns will be attractive to designers and animators, because they are easy to create, they perform well for many motions, and they provide a greater degree of visual abstraction. We envision interactive applications and animations that combine dynamic 2D patterns with other rendering techniques to adapt the visual and animated style to the needs of the moment.

The main limitation of our method, though, is that 2D similarity transforms do not closely match all 3D motions. A particularly bad case is an elongated object rotating around an axis that lies parallel to the film plane, as in Figure 4.4. The 2D pattern inevitably "swims" over the 3D surface in that case. A natural extension of our approach is to use not a single pattern, but a set of patterns to represent the surface.

This way, various coarse scale motions in 2D could be followed by individual patterns.

However, this raises two problems. We first need to choose the number of patterns to use, which relates to a trade-off between motion abstraction and sliding. From one side we have the dynamic pattern approach presented in this chapter, that uses a single pattern per object and may exhibit sliding. On the other side, if one uses one stroke per pattern and a multiple strokes per object, then the motion is much less abstracted, but no sliding occur, as seen from the results of Chapter 3. It would be very interesting to design a system able to range from those two extreme representations, taking into account the complexity of the motion flow in an animation. With such a system would occur a second issue: How do we blend various patterns when they meet at their boundaries? In particular, if two patterns overlap, what will be the resulting perceived motion? This will be discussed in the next chapter in the context of motion transparency.

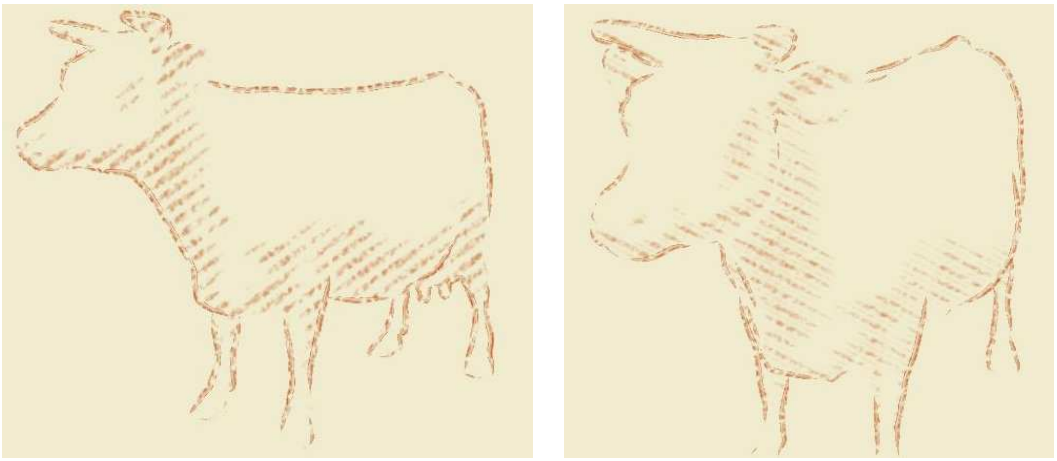


Figure 4.4: For a rotation around an axis parallel to the image plane and an elongated object, sliding becomes noticeable.

Chapter 5

Remarks on low-level perceptual criteria

So far we have presented theoretical as well as practical methods to translate the qualities of traditional painting and drawing techniques to expressive rendering. While most of the previous work concentrated on a specific style, we tried as much as possible to *broaden the representations* allowed by our approaches, at the same time referring to the theoretical model of pictorial representation.

In Chapter 2, we showed how to implement various effects found in paintings or drawings, using a single shader for the processing of attributes. In Chapter 3, we designed a painterly system where the user is able to explicitly control the trade-off between the low-level style to use in the representation and the scene features to depict, thus offering a broader range of possibilities when compared to previous work. Finally, in Chapter 4, we proposed a technique to map the 3D motion of an object to the 2D motion of a stroke pattern, with various applications such as half-toning, screening or hatching.

As noted at the end of the previous chapter, there is still room for *generalisation*: We believe that there is a continuum from pixel-based representations, to stroke-based representations, to pattern-based ones, in the same way one can use a blend of continuous tone and individual brush stroke techniques in traditional paintings and drawings (see Figures 1.2 and 1.3). An ideal system would allow us to produce animations at these various levels of granularity, while keeping an ease of use for the artist (who is not necessarily a skilled computer scientist). In Section 5.1, we present how such a system can be formulated as the solution of a rather open optimisation problem and discuss what requirements it should fulfill. Then, in Section 5.2, we consider how this ideal system should behave when used for animations, which leads us to a discussion of motion perception.

5.1 Guiding shape and color

As formulated by Durand in his inspirational work [Durand 00],

> "Depiction consists in producing the picture that best satisfies the user goals: This is an optimization process. It should most of the time be solved by the user, but the optimization nature of the process requires the design of specific tools for efficient user interaction."

Durand gives the example of photorealistic rendering, where the problem is reasonably stated, available methods providing a very close initial estimate; Estimate that is nevertheless frequently refined via additional techniques such as model touch-up, photographic lighting or post-processing. In the case of expressive rendering, there is no such automatic "close estimate": The representation of a scene has to be strongly tied to the user's intention. Hence it is more the result of a compromise between a *style* and some important scene *features* to depict. In other words, the role of the optimisation process is even more crucial, since it will allow to intuitively balance between these two *a priori* conflicting goals.

What is the nature of this optimisation process? Durand presents a series of requirements that it should fulfill. We consider them as belonging to two different categories: Enabling intuitive user interactions and providing means to define the "energy function" for the computer to solve the optimization.

Intuitive user interactions The first class of requirements advocates, for instance, the definition of controls in image space and the development of purely pictorial techniques with fast feedback to the user. It also recommends to linearize any parameter space involved in the manipulation, and to provide relevant degrees of freedom.

Some simple methods can be designed in this way. Our X-Toon shader, presented in Chapter 2, is an example: While the 2D toon texture (a linear mapping from parameter space to color) is set in preprocess, it is looked up using up to two view-dependent controls (the relevant degrees of freedom), in real-time (for fast feedback).

Another example is layerization: Many computer graphics production studios render animations into multiple layers that correspond to various scene features such as diffuse lighting, shadows and highlights (the number of layers can go up to many dozens). Then these layers can interactively be edited in image space via a compositing software in order to "touch up the initial estimate". Note that it also has some biological plausibility. Indeed, as noted by Durand [Durand 02],

> "A very important characteristic of our visual system is the ability to decompose the image into different layers, e.g. object reflectance, lighting, specular highlights, transparency, etc."

Therefore, as already proposed in Chapter 2, and implemented in the pioneering work of Meier [Meier 96], we should design expressive rendering techniques in a series of layers that run through the pipeline and are composited in the end. However, this might raise some issues in the perception of motion, as discussed in Section 5.2.

More complex methods can also be devised, that take as input a small, localized example of what the user expects to obtain across a picture or along an animation. These techniques belong to the class of *example-based synthesis* methods, or, when the example is directly put in correspondence with some scene features, *analogies*. We do not discuss here the challenges raised by these approaches, since it will be the subject of Parts II and III of this thesis.

Defining the energy function As discussed by Willats [Willats 97, Willats 05], another important factor in the use of marks to depict a scene is *loose control*: The extent to which marks may be tightly or loosely related to scene features. Recall that a picture is made up of marks which represent picture primitives and attributes,

which denote corresponding scene primitives, which represent objects in the scene. Willats relaxes this condition:

> "This configuration is not rigid. For example, variations observed on marks should be intended to carry information, but they could also be accidental. Moreover, we can make use of a given picture primitive to denote another, and the picture primitive may only be implied by other primitives."
>
> "The relations between marks and their attributes and the picture primitives they represent can also be very varied. The dimensional extensions of the marks may not always correspond to those of the primitives they represent."

Allowing marks to be more "free" to depart from their underlying primitives seems to be a rather simple idea, but it hides an issue that has not really been addressed in the past: How can one guarantee that relaxing the mark-primitive connexion will still faithfully depict the scene? An answer is that it can be done through an optimisation process that performs a compromise between depicted scene features and the loosely controlled, user-defined style. The definition of the energy function that has to be minimized is then of primary interest. We believe that such an energy can be defined via perceptual models of low-level vision. In other words, we should be able to give a computational answer to the question: "How the low-level features of a representation perceptually match the corresponding low-level features of the scene it depicts?"

This is a vague question that awaits to be further refined in order to find practical applications in expressive rendering. However, we can give some hints about what kind of processing it implies. In Chapter 3, we presented a painterly rendering method that explicitly performs a trade-off between the depicted scene and its representation: A paint stroke is drawn from an anchor point of a given color and depth (in the scene), until the underlying scene color or depth exceeds a threshold, or a length specified by a user-defined style is reached. This can be seen as the simple minimisation of a simple energy function, that places a bound on the color and depth differences along a stroke. The only perceptual flavor here is that we use the Lab color space for computing color differences.

This energy minimisation formulation also implies that the scene features to represent are well known and easily attainable in the first place. This is not always the case, though. For instance, despite great advances made recently in the expressive rendering community, the body of algorithms intended to draw contour lines from a geometric model does not yet give satisfactory results in all cases. As presented in Chapter 1, various techniques allow to get *some* of the good lines, and they are in no way mutually exclusive. This lack of unity in contour drawing algorithms reveals that a part of the energy minimisation question has not been answered yet: What are the scene features to depict when using contour lines? The intuition is that contours are able to reveal efficiently the shape of objects, but to go further, we might need to study the literature on shape perception, or at least surface perception. Once the salient features will be identified, we will be able to minimize the (yet unknown) energy. We give some hints concerning surface perception in Chapter 9.

5.2 Guiding motion

Let us now discuss the additional constraints that arise when one considers expressive rendering animations. First, the granularity of marks might considerably influence the properties they have to obey when put in motion. For example, at a fine granularity, marks should follow motion information found in the scene, while keeping two-dimensional texture properties: e.g., watercolor pigments should not be distorted too much. At a coarser granularity, strokes should in the same way follow motion in the scene, while keeping a satisfactory distribution, which might be a complex task with the more structured distributions encountered when dealing with stroke patterns.

These intuitive requirements lead to the definition an energy composed of three components, and we should minimise each components simultaneously in order to get a good depiction. The first component relates to *color and shape*, as discussed in the previous section, and states that, for any frame, marks should well depict color and shape features of the scene. The second component relates to the *spatial distribution* of marks in the picture plane: they should be held close to a user intended distribution. This can also be defined in terms of an energy functional to minimise. Finally, the third and last component correspond to a notion of *motion fidelity*: Whether marks motion perceptually matches scene motion. Note that this component includes temporal coherence. Indeed, poppings and flickerings are usually not features of the scene motion, so that they are perceived as artifacts of the representation.

Again, in order to illustrate these three energy components, we shall give some examples. The painterly rendering system in Chapter 3 implements simple behaviors for each component. The first one, color and shape, have been discussed in the previous section. Concerning spatial distribution and motion fidelity, we made a series of decisions which lead to a simple implicit minimisation. First, we decided that strokes could in no way “slide” onto the depicted object; They are sticked to it via their anchor point, so that their motion will be exactly matched with motion in the scene. Then, in order to ensure a user-defined density of strokes, we use a bounding sphere hierarchy and a projection test in order to ensure a picture plane coverage. The use of a hierarchy also allows us to select consistently anchor points from frame-to-frame.

However, due to variations of density and visibility of anchor points arising in a dynamic environment, strokes appear and disappear suddenly in time. Moreover, due to quick changes in illumination (highlights for instance), their attributes may also vary abruptly. If nothing is done, it will result in poppings and flickerings respectively. To alleviate these problems, we introduced transitions for the insertion and deletion of strokes (they appear/disappear progressively), and smoothed out the color channel to reduce the abrupt changes in stroke colors. Such techniques can be seen as simple approaches to minimize the energy, resulting in the impression of a painting that “adapts” to give a good depiction of the scene, which is more perceptually acceptable than the apparition of artifacts.

In contrast, in Chapter 4, we had to relax the “no sliding” condition chosen above, because the use of a single 2D stroke pattern to depict a 3D object will inherently abstract its motion. Regarding the energy formulation, the amount of allowed “sliding” should be controlled by some motion fidelity measure: If the strokes motion is too far from scene motion, then an artefactual motion transparency will be perceived, with resulting sliding sensations, and in the extreme case of a static pattern on top

of an animated scene, a shower-door effect (i.e. the perception of two, unrelated, transparent layers). Therefore, we need a motion transparency model of the human visual system, that will give us an energy function to minimize, probably in the form of a near-noticeable threshold. There has been a variety of studies on motion transparency in the field of natural vision [Andersen 89, Stoner 90, Watson 94, Qian 94] and some computational models seem close to our needs (see, e.g. [Qian 94]). We plan to investigate the possibilities offered by motion transparency models in a future work.

Finally, let us discuss an interesting property of hand-made animations. Because of the time required to produce each frame of an animation, artists usually do not draw every frame. If one draws a frame every two frames, it is said to be "on 2". Similarly, we have animations "on 1" where every frame is drawn, or "on 4" where every fourth frame is drawn, etc. Apart from reducing the amount of work, this stop-motion technique offers artistic qualities. First, it reduces temporal coherence problems, since poppings are harder to notice at lower frame rates. But also, it gives more room for imagination in-between frames: The brain seems to fill in the blanks naturally. Relying on the observer's perception has this advantage that, in a way, he will perform the energy minimisation by himself, and thus the artist is more free to experiment with abrupt transitions.

The fact that stop-motion animation has such interesting qualities motivates its use in expressive renderings. However, the way it is performed by artists is empirical, hence there is no robust rule to build on. In other words, before transposing the qualities of stop-motion to expressive rendering, we shall first answer some vision-related questions. What makes an animation look more expressive, more lively when fewer frames are used? The brain seems to fill in the blanks naturally, but also to perform better inference for fast and brief motion. Why? This is especially true for the motion of a character, but can we use less frames for camera motion in the same way? If one uses a frame-to-frame blending technique more sophisticated than classical stop-motion, how will it influence the final perception of motion?

To my knowledge, none of these questions have been investigated in the vision community. What is particularly interesting in the case of stop-motion, is that it is an example where expressive rendering has the potential to **motivate** research in vision.

Part II

Line drawings analysis and synthesis

Chapter 6

What do line drawings represent ?

In Part I of this thesis, we investigated expressive rendering issues related to the *representation* of image sequences. Among the various possibilities to design systems that deal with these issues, we mentioned the advantages for a user to provide a style directly via an example, either drawn by himself or contained in a reference picture (say, a drawing or a painting). In Parts II and III, we explore this idea in the more general context of *acquisition*: starting with a drawing or a painting, we aim at extracting information about its style. Then, one can either transform the input representation based on that additional knowledge, create from scratch another representation with a similar style, or transfer it to a target representation. In this part, we focus on line drawings.

(a) (b)

Figure 6.1: (a) A line drawing example. (b) Shadow is suggested by line thickness.

6.1 Examples from traditional drawings

Hand-made line drawings have been widely used in the past for scientific illustration, as well as artistic purposes. In illustration, it has in particular many advantageous

properties with respect to reproduction. A drawing is usually done at a bigger scale and is reduced when printed, as pointed out by Wood [Wood 94],

> "The purpose of black lines and stipples is to give the viewer an impression of various values of gray."
>
> "The reduction process improves the drawing by diminishing imperfections and by crisping up the lines and dots, making them appear as various values of gray."

On the other hand, for creative applications, line drawings allow a remarkable depiction with an economy of means, leaving to the observer the task of "filling in the blanks", suggesting more than illustrating.

Regardless of the targeted application, traditional drawings can contain an arbitrary number of elements, ranging from long and smooth lines to small figures; They may exhibit juxtaposed layouts of strokes, regular patterns with a common orientation, over-sketched lines, junction points, etc. To better analyze drawings, we first distinguish two categories of elements: **contours** and **patterns**. We motivate this choice by taking inspiration from introductory painting books (e.g. [School 95]) and present their main properties below.

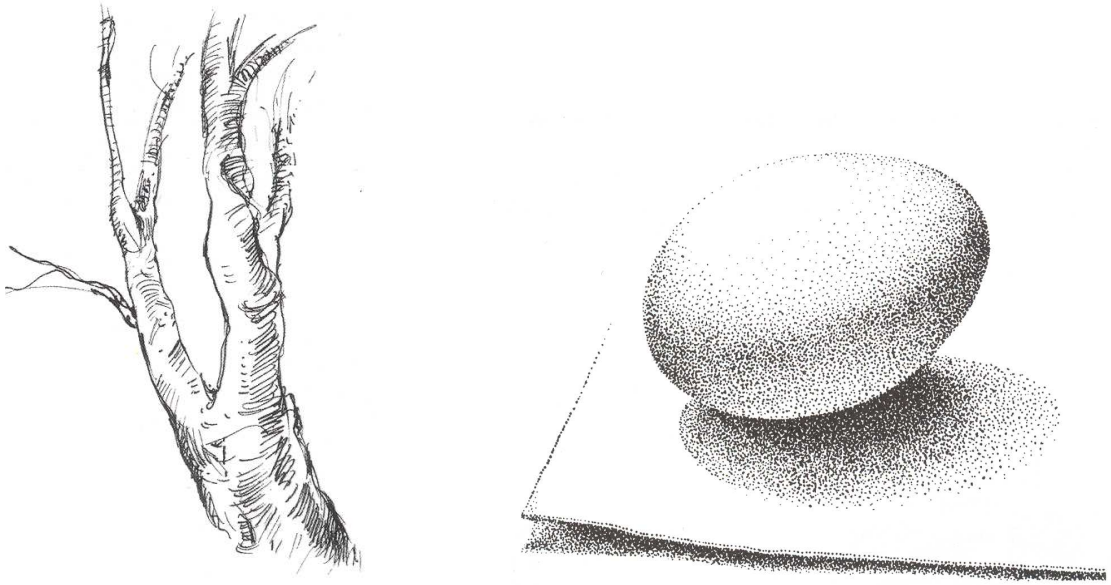
What contours represent A line is the most elementary form of drawing, but the strength or softness of a gesture provides with multiple expressive possibilities (see Figure 6.1(a)). Using heavy or light contours, for instance, permits one to suggest shadowed or enlightened areas (see Figure 6.1(b)).

Probably the most prominent quality of a line is to guide one's observation towards the shape of a subject and, depending on the gesture, to suggest volume and matter. Observe the drawing of the little girl in Figure 6.1(a): The slightly curved silhouette lines used to draw her legs reveal that they are bandy; The simple, sketchy silhouettes of her hands add to the somewhat tense feeling that emanates from the drawing. Figure 6.1(b) is even more suggestive: The woman's spine is drawn via very coarse, thick lines and yet the representation is perfectly clear to any observer.

What patterns represent The most common form of line pattern is hatching. Hatchings are composed of distinct lines that are roughly parallel. They convey both shape by their local orientations, illumination when they are used to depict shadows, and texture via the subtle variations of the gesture. For instance, curved hatchings evocate a tubular form (see Figure 6.2(a)).

Additional hatching techniques permit to make variations on patterns. One example is cross-hatching, where two patterns with nearly perpendicular orientations are drawn on top of each other, allowing darker values for the depiction of shadows. A second example is the use of "zebra" which consist in long, regular curved lines, this time to better suggest the shape of objects.

Another form of pattern is stippling (see Figure 6.2(b)). It is simply a collection of dots. Like hatchings, they are used to represent shape and tone via subtle variations of their density and size. Finally, patterns can be composed of more complex elements than points and lines, like small figures, small or long wavy brush strokes, etc.



(a) (b)

Figure 6.2: (a) A hatching example. Slightly curved lines reveal a tubular shape. (b) A stippling example. Variations in points density suggest both shape and illumination [Wood 94].

6.2 A digital drawing toolbox

An important challenge facing researchers in expressive rendering is to develop hands-on tools that give artists direct control over the stylized rendering applied to drawings or 3D scenes. An additional challenge is to augment direct control with a degree of *automation*, to relieve the artist of the burden of stylizing every element of complex scenes. By complex, we mean scenes that incorporate significant repetition within the stylized elements, either inside a single view (think of an illustration containing a pattern composed of thousands of hatching lines), for multiple views at different scales (e.g. a silhouette viewed at varying depths in different panels of a comics), or multiple frames of an animation (like a set of zebra which may vary in density based on viewing and lighting configurations).

While many methods have been developed to achieve such automation algorithmically outside of expressive rendering (e.g., procedural textures), these kind of techniques are not appropriate for many renderings where the stylization, directly input by the artist, is not easily translated into an algorithmic representation. An important open problem in expressive rendering research is thus to develop methods to analyze and synthesize artists' interactive input. We now propose some (non-exhaustive) research directions intended to enrich the digital drawing toolbox with smart tools that fulfill the above-mentioned requirements. As in the previous section, we differentiate between tools manipulating contours and patterns.

Contour line tools A first category of tools in digital drawing is interactive tools. Their goal is to answer the question: How a user gesture might be transformed into a contour line? The simplest answer being: Just draw the gesture as it is. But it can also be done in a context-dependent manner. For instance, in car design, lines

of a primal sketch are intended to be smooth, in order to give a clear, overall idea of shape; But user gestures, even for a trained drawer, always have some irregularities. Therefore, it is useful to fit a smooth curve to the input gesture. Of course, outside of car design, this constraint might be too strong.

An interesting direction of research would then be to create a low-level, *multi-purpose* interactive drawing tool. We shall then define how gestures have to be processed, probably in a parametrized way to adapt to various applications. Junctions might be treated as well, since they give important cues about shape. However, they work at a slightly higher-level of processing since they are determined not by one, but at least two gestures. Finally, such a generic interactive drawing tool might incorporate an over-sketching functionality: By drawing on top of existing lines, the user should be able to replace them with new ones. Note that each operator cannot be done independently of the others, using e.g. ad-hoc rules: An interactive analysis of the drawing being performed should be done such that the proper decisions (i.e. leading to a possible interpretation of the drawing) are made for each new user interaction.

Another category is what we call simplification (also called cleaning or beautification). Here the idea is to start with a full drawing, and to reduce its *complexity*. For example, when reproducing a drawing, it is important to adapt it to the scale at which it will be viewed, unless too many lines may clutter the picture plane and make the drawing incomprehensible. The same problem arises when one wants to create a series of panels of a same drawing at different scales (as in some comics): He or she cannot simply rescale the drawing, and thus has to re-draw each new panel from scratch. Finally, it is also efficient for correcting some form of inaccuracy, for instance to clean up a diagram drawn on a white board and captured for archival, to smooth out a sketch drawing, or to reduce the often inappropriate density of computer-generated line drawings from 3D objects.

However, in order to efficiently design tools that perform such useful tasks, one has to define complexity. To my knowledge, this has not been done in a general way in the past; In other words, each new method brings its new definition of complexity. It would be interesting to investigate this question in order to perform a better task-oriented cleaning of a drawing. Moreover, note that these problematics are in no way trivial since, as with interactive drawing tools, we need to ensure that the cleaned up drawing "makes sense": The original and cleaned drawing should convey essentially the same shape cues, and this requires an analysis.

Finally, a third category is that of re-drawing. It is in-between the two categories presented above, and belongs to the class of analogies. The principle is simple: The user re-draws a contour in an input drawing, and all *similar contours* are re-drawn in a same way. This sounds like a very useful tool for a lot of digital drawing applications: Think for example of applying a style (like some waving) to a contour, and all the similar contours are interactively updated in a similar style, allowing to design contours in a very efficient, yet expressive manner. And it does not have to be limited to contours, junctions could also be re-drawn, for example to create haloing effects.

This time, the key feature to an efficient re-drawing tool is the notion of contour similarity. It is an interesting question since, again, it requires to analyze the deep structure of the drawing in order to identify contours or junctions that depict similar shape cues. However, it is a complex question which, to my knowledge, has not received any answer yet.

Pattern tools At "first sight", patterns seem to be well structured so that their analysis is a well stated problem: Hatchings are regularly spaced and have a main orientation, while stipplings are properly defined via their distribution. However, more complex arrangements, such as brush stroke patterns, can exhibit significantly complex structures (see, for instance, the patterns found in Van Gogh paintings). Or their regularity might impose constraints on structure that are difficult to analyze, even more to reproduce, like with patterns that resemble tilings (think of a drawn brick wall). To summarize, a pattern's structure may vary from very irregular to very regular, and tools for manipulating them might be adapted correspondingly.

We can distinguish two categories of pattern tools. The first one analyzes and synthesizes patterns out of context. In practice, the user draws a small sample that is analyzed and then replicated via synthesis on a user-specified broader area. The main issue is that the reference and synthesized pattern are perceived as *similar patterns*. Patterns are either one-dimensional (like hatchings along a curve to suggest fur or a doted line to represent hidden features of a 3D object) or two-dimensional (like the oblique lines used to depict rain drops in comics or the stipples used to depict shadowed regions in archaeological illustrations). They can either be applied directly onto the picture plane, or mapped to a surface so that they will also suggest shape when viewed in perspective. Such tools should provide the user with intuitive controls to specify the area where a pattern is synthesized (either a curve for 1D patterns or an oriented region for 2D patterns).

The second category of pattern tools this time analyzes and synthesizes patterns in context, i.e. in relation to some feature values coming from an underlying pictorial or 3D scene (tone, depth, etc). In practice, the user draws a small sample that is analyzed in relation with an underlying feature, and then automatically replicated via synthesis onto instances of the same feature. The issue of synthesized and reference pattern similarity is of course still relevant. But in addition, we have to find some *correlation* in-between the reference pattern and features, in order to be able to automatically synthesize it at proper locations in the scene. This approach is, like the re-drawing of contour lines presented above, an analogy process. Another interesting functionality would be to define variations of a pattern (a set of reference patterns) with respect to variations in the scene; This would allow to synthesize dynamic, viewing- and lighting-dependent patterns for instance.

We have not defined pattern similarity nor correlation yet. Indeed, these are the key aspects of the analysis that will permit us to get meaningful results with pattern synthesis. Similarity is a complex problem, though, since there is no single way to characterize two patterns as being similar: Some may consider that they have to be indistinguishable (this is equivalent to establishing a near-noticeable difference between patterns); Some may consider that they only need to perceptually belong to a same (fuzzy) category of patterns (hatchings have to be different from stipplings, regular hatchings different from wavy ones, etc). Previous work on texture discrimination might help to define such measures.

Concerning correlation, the number of possible measures can be even greater: when a pattern is drawn on top of a 3D scene, then it can in theory represent many various features at the same time. However, in practice, as we have seen in the previous section, patterns give some specific clues about the scene they represent. Investigating those clues in a more rigorous way will allow to significantly reduce the difficulties of establishing the correlation. However, as we will see in the next section,

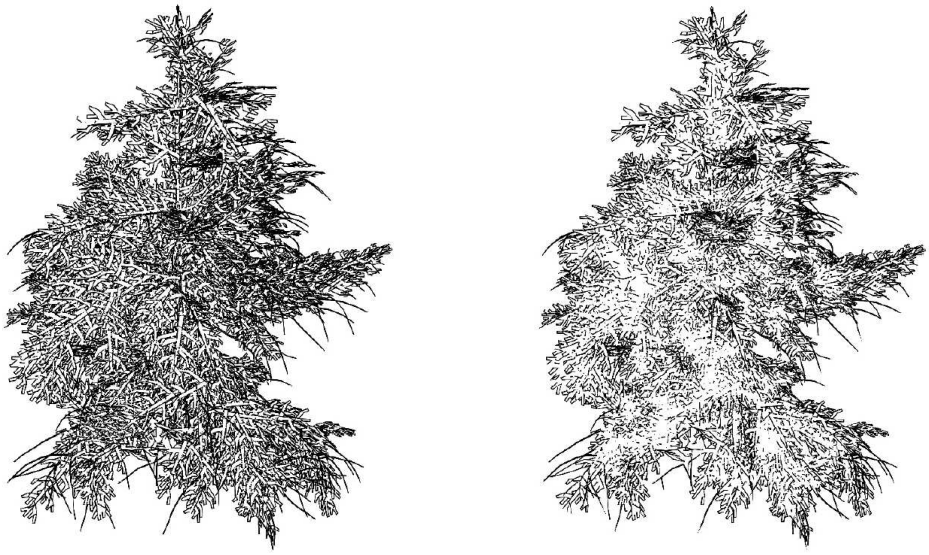
analogies with patterns is still an open avenue for research work.

Restriction to vector-based graphics For the sake of simplicity, we consider that line drawings (contours and patterns) come in a vector form, and do not mention techniques that extract vectorized versions of, for instance, scanned drawings. The line geometry is thus represented explicitly as connected vertices with attributes such as width and color. While this vector representation is typically less efficient to render, it has the important advantage that lines can be controlled procedurally to adapt to changes in the depicted regions (they can vary in opacity, thickness and/or density to depict an underlying tone). This is an important feature discussed in Chapter 5, related to the notion of "energy minimisation".

6.3 Previous work

Line drawing synthesis systems have been studied in the past, for example to generate stipple drawings [Deussen 00a], pen and ink representations [Salisbury 94, Winkenbach 94], or engravings [Ostromoukhov 99]. Several papers also dealt with level-of-detail (LOD) rendering for animated scenes. Praun et al. [Praun 01], for instance, present an image-based method to handle LOD in hatching. Their tonal art maps (TAM) are mip-mapped textures allowing real-time display of hatching styles.

However, all these systems have relied primarily on generative rules, either chosen by the authors or borrowed from traditional drawing techniques. We are more interested in analysing reference drawings input by the user and synthesizing new ones with similar perceptual properties.



(a) (b)

Figure 6.3: (a) An input line drawing. (b) A simplified version of the drawing by Grabli et al. [Grabli 04a]. The idea is to keep a few complex regions at the borders of visually dense regions to suggest their overall complexity.

Manipulation of contour lines Progressive drawing tools [Igarashi 97, Baudel 94] are being developed in the context of sketch-based modeling, or within vector graphics packages such as Adobe Illustrator™. These dedicated tools assist the user in adjusting the shape of a line: they are essentially semi-automatic, work iteratively and are not designed to edit more than one line at a time. Moreover, they often rely on ad-hoc rules which are specific to their target application, and are thus not easily adapted to another one. Even if dedicated strategies will always be preferable in some domains such as car design, we believe that a more general, free-hand and under-constrained method would be of particular interest.

Several approaches to simplification have also been proposed in the past. Preim et al. [Preim 95] and Wilson et al. [Wilson 04] measure density in image space in order to limit the number of lines drawn for complex objects. Similarly, Grabli et al. [Grabli 04a] introduce density measures in image space, used to select the most significant lines. The use of information extracted from the 3D scene (silhouettes, creases, etc.) allows them to evaluate this "significance" and order the lines by decreasing priority. The simplification process is then carried out by deleting the least significant lines (see Figure 6.3). Here, complexity thus means "number of lines". However, it would be interesting to consider other forms of complexity.

Similar approaches have been proposed in the context of resolution-dependent display and printing, since they are related to half-toning [Salisbury 96, Zander 04]. Here again the authors add a notion of priority to ensure that the most important lines are drawn first for any tone level and then offer a selection mechanism of some lines among the original ones. The cleaning of sketched diagrams is another form of simplification that has received some attention in the User Interfaces community, see for instance the work by Saund et al. [Saund 94, Saund 03]. A notable quality of these approaches is that they consider perceptual implications.

(a) (b)

Figure 6.4: (a) An input line drawing. (b) The lines of (a) re-drawn in a different style, by Freeman et al. [Freeman 03].

Finally, re-drawing methods emerged recently, with very interesting applications. Kalnins et al. [Kalnins 02] described an algorithm for synthesizing stroke "offsets" (deviations from an underlying smooth path) to generate new strokes with a similar appearance to those in a given example set. They applied this technique to the stylisation of computer-generated line drawings. One big advantage of using a 3D scene as

input is that the instances of a feature (here, all the silhouettes of a same model) are readily available without the need of an analysis. Hertzmann et al. [Hertzmann 02b], as well as Freeman et al. [Freeman 03] address a similar problem (see Figure 6.4), but this time in drawings without any 3D information, using the notion of analogy.

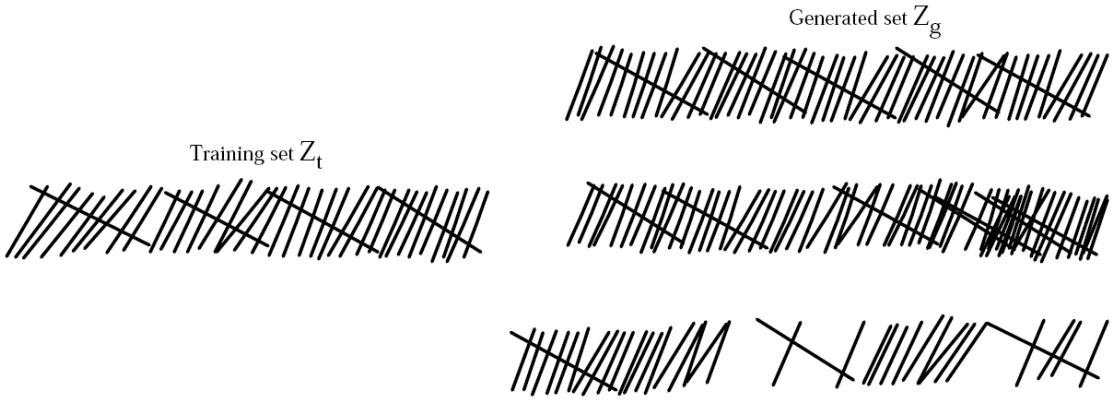


Figure 6.5: Left: The reference input pattern (or training set) (b) Synthesis results for three window sizes (10, 6, and 4) by Jodoin et al. [Jodoin 02].

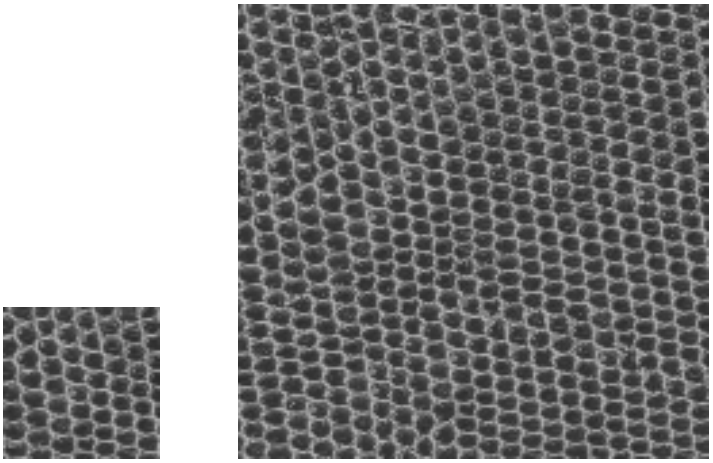
Pattern analysis and synthesis Related work can be found in two different research fields: stroke-based rendering methods that aim at reproducing various artistic styles, and texture synthesis methods that aim at generating a texture from a given sample image.

Above-mentioned methods on single line synthesis [Kalnins 02, Hertzmann 02b, Freeman 03] will fail in synthesizing patterns since neither method reproduces the inter-relation of strokes. Very little work has been done on the subject. A notable exception is the work of Jodoin et al. [Jodoin 02], that focuses on synthesizing one-dimensional hatching patterns, which is a relatively simple case in which simple strokes are arranged in a linear order along a path. Their approach works by analyzing a sample pattern (Training set) with a window of fixed size. Figure 6.5 shows the importance of selecting a sufficiently large window size.

In a first attempt to extend synthesis to more general stroke patterns, we developed a statistical approach for the case of hatchings and stipplings in 1D and 2D [Barla 06a]. No neighborhood comparison was taken into account and the class of possible elements was reduced to single-colored points or lines. We extend this technique to a wider range of patterns in Chapter 8. Our new approach bears some similarities with the graphical search and replace method of Kurlander et al. [Kurlander 88].

While pattern analysis is a relatively young branch of expressive rendering, it is a domain by its own in computer and human vision (the interested reader is referred to Journal issues like, for instance, PAMI, which stands for Pattern Analysis and Machine Intelligence). And the idea of synthesizing textures, both for 2D images and 3D surfaces, has been extensively addressed in recent years. Previous work can be organized in two categories: parametric and non-parametric methods.

Parametric methods aim at giving a compact description of textures: they make use of statistical analysis to characterize an input texture by a set of parameters, and then try to synthesize similar textures in order to validate the parametric model. They are thus more oriented towards Vision problems. The reader can refer to the paper by Portilla and Simoncelli [Portilla 00] for a good overview of parametric models.



(a) (b)

Figure 6.6: (a) The reference input texture (b) Synthesis results using Efros and Leung's pioneering work [Efros 99].

On the other hand, non-parametric methods work exclusively from the reference texture [Efros 99, Wei 00, Wei 01, Turk 01, Ashikhmin 01]. Even if this representation is less compact, the synthesis results are usually more convincing and the type of synthesized textures more general (see Figure 6.6). This is the reason of their success in Computer Graphics applications. These iterative algorithms work with neighborhood comparisons between the reference and target textures. The size and shape of the neighborhoods vary from one technique to the other. Some methods work in scanline order, while others grow the texture from a central starting point; Synthesis is sometimes hierarchical. Still, one cannot directly apply pixel-based non-parametric texture synthesis to vector-based patterns, because stroke patterns are composed of individually perceived elements.

6.4 Contributions

In the following two chapters, we present our contributions in the manipulation of line contours in drawings, and the analysis and synthesis of stroke patterns, respectively. Our general approach to both problems is to try to generalize as much as we can the scope of their applicability.

In Chapter 7, we present a framework that handles both the simplification of line drawings and their interactive creation through over-sketching. This work presents for the first time methods that deal with free-hand as well as computer-generated drawings with very little or even no knowledge on the subject represented. It also makes use of the Gestalt theory in human vision, also known as perceptual organization.

Then, in Chapter 8, we describe a stroke pattern synthesis tool that addresses many of the previous work limitations, while taking inspiration from classic texture synthesis. Synthesis is thus extended to more complex pattern elements (not only points and lines, but also small figures, sketched or overdrawn lines, with color) and pattern dimensions (i.e. in 1D and, for the first time, in 2D).

Chapter 7

Line drawing simplification

Figure 7.1: The two stages of our method. Lines of the initial drawing (left) are first automatically clustered into groups that can be merged at a scale ε (each group is assigned a unique color). A new line is then generated for each group in an application-dependent style (at right, line thickness indicates the mean thickness of the underlying cluster).

Artists have since long learned to cleverly tune line density across their drawings. However, most of the time in computer graphics, lines do not come with an appropriate density. Simply scaling a drawing, for instance for displaying on low-resolution devices, creates a need for density adjustment; moreover, density reduction in 3d is not yet mature and most non-photo-realistic rendering (NPR) systems extract far too many lines. Thus there is a need to adapt the number of lines in a drawing, otherwise the effectiveness of such a representation may be compromised. In this chapter, we present a simplification technique that solves this density problem for various classes of drawings.[1]

There is not a single way to simplify a set of lines, depending on the envisioned application. In the context of density reduction, we may want to adjust the line density of a drawing where too many lines project in a given region of the image. This is needed when scaling a line drawing, as well as when rendering from a 3D scene. In this context only the most "significant" lines should be drawn. Level-of-detail (LOD) representations for line-based rendering (contours and hatching), where the number of lines must vary with scale, constitute another simplification approach. Finally, in the context of progressive editing (sometimes called over-sketching), the user refines a curve by successive sketches. It can be viewed as an iterative simplification of the set of line sketches provided by the user.

[1]This work has been done in collaboration with my advisors Joëlle Thollot and François Sillion. It has been published in the Eurographics Symposium on Rendering (EGSR) in 2005, see [Barla 05].

Problem statement

We consider a drawing to be a digital image composed of a number of vectorized 2D lines. Such images can be obtained in various ways: by scanning and extracting lines from a hand-made drawing; by direct digital creation using appropriate input devices (mouse, tablet, etc); by detecting contours in an image [Ziou 98]; by rendering a 3D scene in a line style [Strothotte 02, Gooch 01, Grabli 04b].

We are focusing on simplification methods for such sets of lines, *i.e.*, the creation of another set of lines containing fewer lines than the original set. We propose a generic approach for this type of problem, where simplification is controlled by a single distance-based scale parameter ε.

Of course, this rather restricted view (in which spatial proximity is used as the main discriminating criterion) implicitly assumes that all lines belong to a coherent set, over which simplification can be carried out using a very low level semantic description. In particular this approach is not tailored to regular structures or other higher order arrangements. However, nothing prevents the user from preprocessing the data to organize lines in different categories and to apply our method to simplify independently each category.

Previous methods [Igarashi 97, Baudel 94, Deussen 00b, Preim 95, Wilson 04, Grabli 04a] **only delete** the less significant lines and do not consider any **perceptual** aspect of line simplification. On the other side, perceptual grouping approaches provide effective ways of consistently grouping lines, even if they do not address the problem of simplification directly. Most of the work in this area deals with the extraction of closed paths in drawings [Saund 03, Elder 96], focusing on grouping criteria such as good continuation and closure. However, other criteria are more relevant for simplification purposes, e.g. proximity and parallelism. Unfortunately, even if each criterion has been studied in isolation [Rosin 94], their relative influence is yet to be determined.

Contributions

Our main contribution is an attempt to model the common properties of target applications of line drawing simplification. To do that, contrary to previous methods, we construct a partition of the original set into **consistent groups** that can be replaced by an **entirely new line**. This approach is new since it combines advantages of perceptual grouping while allowing many simplification behaviors, not only deletion.

To this end, we decompose the process into two main stages (see Fig. 7.1): a clustering stage in which we group the original lines and a geometric stage where a new line is created for each group. While the former is entirely automatic and common to all applications, the latter is oriented toward the specific needs of each of the envisioned applications. Our approach is general in the sense that it considers a set of minimal and low level goals shared by those applications.

We begin by describing our methodology in Section 7.1. The common, automatic clustering stage is presented in depth in Sections 7.2 and 7.3. Our contribution here is the definition of a modular algorithm that clusters any kind of line.

Section 7.4 shows how to adapt our clustering algorithm to different needs, giving simplification results in the contexts of density reduction, LOD and progressive drawing; for lines coming from different sources: scanned drawings or non-photo-realistic renderings. Finally we discuss limitations of our method in Section 7.5.

7.1 Methodology

We now present the principles of our simplification method, including formal definitions that will help to clarify our approach.

7.1.1 Input lines

We define a line-drawing as a set of 2D lines holding a set of attributes (color, thickness, style parameters, etc.), without any assumption on their nature. Thus a line l is only defined by two end points and a continuous path between them.

$$l : [0,1] \rightarrow \mathbb{R}^2 \times \mathbf{A}$$

where $\mathbf{A}$ is the space of attributes.

In the following, we will denote by $l|_{[a,b]}$ the part of l restricted to $[a,b]$ with $0 \leq a,b \leq 1$. We are not interested in an exact parameterization of lines, but only on their geometric properties. Therefore in the following, we will refer to a line l indiscriminately to refer to the set of geometric points that constitute it.

7.1.2 Objectives

As already stated, our main goal is to create a set of lines containing fewer lines than the original one. For that, we first need to control the amount of simplification accomplished by our method. Our target applications all have a single common parameter: the simplification scale. This scale, which we denote by ε, is thus the only parameter needed by our approach. Intuitively, we only simplify the existing information at a scale smaller than ε, keeping all the information present at a higher scale.

In the applications we envision, we first need to ensure that the overall configuration of the original drawing is respected in the simplified one. This **coverage** property consists of creating new lines only in regions where initial lines can be found. Regarding perceptual grouping, it accounts for proximity and continuation effects simultaneously.

(a) (b) (c) (d)

Figure 7.2: (a) The simplified line (in pink) has a fold while the initial group (in black) does not - (b) Two simplified lines are preferred to represent this group - (c) The simplified line does not reflects the initial orientation and shape of the group - (d) Using three simplified lines better maintains the shape of the hatching group.

However, we are not only focusing on the line positions, but also on their shape. We want the new lines to respect the way the initial lines have been created. In Fig. 7.2-(a), the new line folds onto itself to cover the original lines that form a fork[2], although no such fold was initially present. We prefer a solution such as the one

[2]Y-shaped configuration

shown in Fig. 7.2-(b), using one more simplified line, but with more fidelity to the global shape of the original lines. The same problem can be found in other examples, such as hatching groups used to shade regions (see Fig. 7.2-(c),(d)). In both cases, we found that imposing a **morphological** property that prevents new lines from folding onto themselves preserves the shape of the original drawing. It has the benefit of ensuring parallelism of the lines clustered at the scale ε.

Finally, still following perceptual grouping, we want to be able to reject the simplification of a pair of lines if their attributes (e.g. colors) are too different.

Following our objectives, we now give formal definitions related to our simplification approach.

7.1.3 Definitions

We begin by the definition of an ε-line and use it to define a group that can be simplified by a single line at the scale ε.

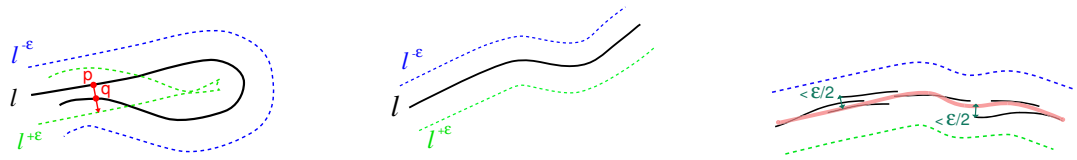
Following our **morphological** property, an ε-line is a line that does not fold onto itself at the scale ε. It corresponds to the fact that, for each point of l, there is no point along the normal at a distance less than ε that also belongs to l (see Figure 7.3). We assume l to be G^1 in order to ensure that its normal is uniquely defined at each point:

Definition. *(morphological property) Let l be a line. l is an ε-line if and only if l is G^1 and*

$$\forall p \in l,\ \nexists q \in l,\ \begin{cases} q = p + \sigma \vec{n}_l(p) \\ \sigma = ||p-q|| \leq \varepsilon \end{cases}$$

where $\vec{n}_l(p)$ is the normal vector of l at point p.

This definition is equivalent to saying that l is not intersecting one of its two offset curves $l^{+\varepsilon}$ and $l^{-\varepsilon}$ (see Fig. 7.3(a-b)).



(a) (b) (c)

Figure 7.3: (a) q is along the normal at p thus the line l has a fold and hence is not an ε-line - (b) l is an ε-line - (c) An ε-group is a group (in black) that can be covered by an ε-line (in pink) at the scale ε.

We now define an ε-group as a group of lines that can be simplified by a single ε-line, as stated in our **coverage** property (see Fig. 7.3(c)):

Definition. *(coverage property) A group of lines G is an ε-group if and only if there exists an ε-line l such that* [3]*:*

$$d_{SH}(l,G) < \frac{\varepsilon}{2}$$

[3] $\frac{\varepsilon}{2}$ ensures that two lines of the same ε-group are at a distance smaller than ε

where d_{SH} is the symmetric Hausdorff distance defined by:

$$\begin{aligned} d_{SH}(P,Q) &= \max(h(P,Q),h(Q,P)) \\ h(P,Q) &= \max_{p\in P}(\min_{q\in Q}||p-q||) \end{aligned}$$

with $P,Q \subset \mathbb{R}^2$.

7.1.4 Our approach

Following our definitions, our approach states that a simplified line-drawing is a set of ε-lines that covers the original drawing at a scale ε. We first need to cluster the original lines in a set of ε-groups before being able to create any new line. Therefore our simplification method is organized in two stages:

1. A clustering stage first groups the lines of the original drawing in ε-groups. No line is created at this stage and the process is entirely automatic using a greedy algorithm to iteratively group the original lines.

2. A geometric stage then creates a single line for each cluster of the original drawing. For each of the three target applications, we use the clusters differently and apply dedicated strategies.

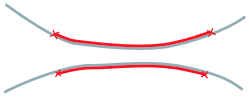
The clustering stage is our main contribution in this work, thus it is presented in detail in the next two sections. We then give in Section 7.4 various examples of the geometric stage and show that our approach can address specific applications without demanding too much effort on the user side.

7.2 Clustering

We use a greedy algorithm to partition the set of input lines. It is based on the iterative clustering of pairs of ε-lines and maintains the ε-group property during the entire process: clustering a pair of ε-lines that each represent an ε-group results in a new ε-line that represents the merged group. The first step consists of converting the original lines into ε-lines by splitting them at their points of intersection with their offset curves (see Section 7.3.1). Our clustering algorithm then iteratively clusters the pairs of ε-lines (Section 7.2.1) that have the minimum error (Section 7.2.3) until no more clusters can be created. At each step we store the hull of the clustered pair in order to take into account the result of previous clusterings in the next steps (Section 7.2.2).

7.2.1 Clustering pairs of ε-lines

Following our definitions, a pair of ε-lines (l_1,l_2) can be clustered if and only if (l_1,l_2) is an ε-group. This definition is not constructive: it only says that there must exist an ε-line that covers (l_1,l_2). We now describe a way of building this new ε-line from l_1 and l_2.



(a) Fork

(b) Two paths: closed curve

(c) Four possible configurations with one path

Figure 7.4: Possible configurations of a pair of ε-lines. Only (b) and (c) can form an ε-group.

Possible configurations for (l_1, l_2) to be an ε-group

Observation. *The coverage property implies that if (l_1, l_2) is an ε-group then there exists a point p_1 (resp. p_2) of l_1 (resp. l_2) such that $||p_1 - p_2|| < \varepsilon$. If not, it would not be possible to find a line l such that $d_{SH}(l, (l_1, l_2)) < \varepsilon/2$.*

We call *overlapping zones* the portions where l_1 and l_2 are at a distance less than ε, formally defined as:

Definition. *An overlapping zone, Z, is a pair of line portions $(l_1|_{[a_1,b_1]}, l_2|_{[a_2,b_2]})$ such that:*

$$d_{SH}(l_1|_{[a_1,b_1]}, l_2|_{[a_2,b_2]}) < \varepsilon$$

where $\{a_1, a_2\}$ and $\{b_1, b_2\}$ are the extremities of Z.

Observation. *The morphological property implies that (l_1, l_2) is an ε-group if it is not a fork. Indeed, if it were the case, then any line l representing (l_1, l_2) would have to fold onto itself.*

This implies that the overlapping zones must not fork, thus there must be at least one of the two lines that ends at each extremity of the zone. A simple forking configuration is shown in Fig. 7.4-(a). Other forking configurations exist, but are not represented because we mainly direct our attention to valid zones (Fig. 7.4(b) and (c)).

We call a *path* such an overlapping zone and define it by:

Definition. *A path on a pair of lines (l_1, l_2) is a maximal overlapping zone, Z, such that there is at least one extremity of l_1 or l_2 at each extremity of Z.*

Knowing that each line has 2 extremities, there are five combinations for the paths between two lines, illustrated in Fig 7.4-(b), (c). Thus, if a pair of lines does not correspond to one of these five combinations, it is not an ε-group.

The only configuration that corresponds to a closed curve is when a pair of lines (l_1, l_2) has two paths (see Fig 7.4-(c)). For the sake of brevity, we will not detail this case in the following, as it is essentially equivalent to the others. However Section 7.2.4 shows that our algorithm correctly handles it.

Building the new ε-line

Now that we have identified valid configurations (paths), we build the new line l, and make sure that it is an ε-line, i.e. that it respects the morphological property. We create an ε-line l that passes from l_1 to l_2 and lies in the middle of the path.

l is obtained by concatenating the portions of lines outside the path (in purple) with a line created inside the path by interpolating between l_1 and l_2 from one extremity to the other (in pink).

Such a construction insures that l is an ε-line outside the path since l_1 an l_2 are themselves ε-lines. However for zones inside the path, some particular cases when l is not an ε-line exist. Indeed, l may fold onto itself if the curvature of l_1 or l_2 is too close to $1/\varepsilon$. In these cases, (l_1, l_2) is simply not considered as an ε-group.

7.2.2 Building the hull of an ε-group

The new line l we created only ensures that (l_1, l_2) is an ε-group; we cannot use it iteratively since it would not take into account the error made by the clustering of l_1 and l_2. Indeed, consider an ε-line l_3; determining if (l_1, l_2, l_3) is an ε-group is not directly equivalent to determining if (l, l_3) is an ε-group.

In order to propagate the clustering result of (l_1, l_2) to l, we define the hull of an ε-group by assigning a varying thickness to the ε-line that represents it. This thickness describes the result of the clustering of two or more lines and is used in subsequent clusterings (see Fig. 7.5).

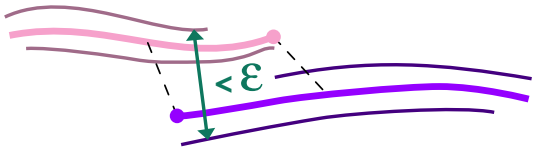


Figure 7.5: To decide if the four thin lines are an ε-group we use their two representatives (in pink and purple) and compute the distances between the farthest lines, which will be represented by the hull.

For each point $l(x)$, the points of the hull $l^+(x)$ and $l^-(x)$ are obtained by taking the extremal intersections along the normal with (l_1, l_2) (see Fig. 7.6).

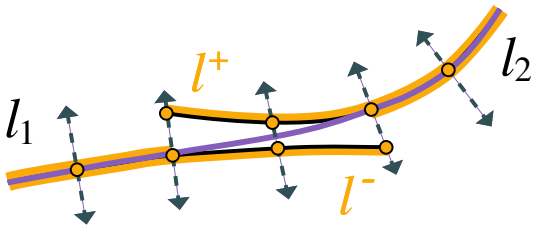


Figure 7.6: The hull (in orange) of a line l (in purple) representing a pair of lines (l_1, l_2) (in black) is defined by the farthest points of (l_1, l_2) along each normal of l (dashed line).

All the definitions given in the previous section are easily extended by considering the two hulls instead of the two ε-lines. Indeed, to decide if a pair of ε-lines (l_1, l_2) is an

ε-group we only need to compute distances between pairs of points. By considering $l_1^+, l_1^-, l_2^+, l_2^-$ for the distance computation, we can determine the overlapping zones and then the paths between l_1 and l_2 while taking into account the two ε-groups they already represent. Therefore, while computing overlapping zones between two ε-lines, the distance between $l_1(x_1)$ and $l_2(x_2)$ will be taken as:

$$\max\{\ ||l_1^+(x_1), l_2^+(x_2)||,\ ||l_1^+(x_1), l_2^-(x_2)||, ||l_1^-(x_1), l_2^+(x_2)||,\ ||l_1^-(x_1), l_2^-(x_2)||\ \}$$

Note that the hull of an ε-line l is not defined on points p where there is no intersection with l_1 and l_2 and the normal at p. For those points, we project the closest points of the two hulls as shown in Fig. 7.7. This will only have an impact on the error computed on the hull as explained in the next section and our choice favors pairs of aligned lines (i.e. with good continuation).

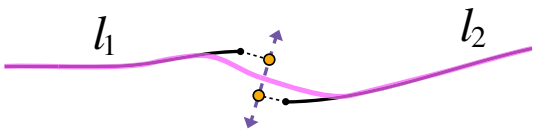


Figure 7.7: In places where there is no intersection with the normal, we project the closest points on the pair of lines.

7.2.3 Error measure of an ε-group

In order to use a greedy algorithm we now need to choose which pairs of ε-lines we want to cluster at each step. To this end we define an error measure.

Intuitively we want to cluster the closest lines first. By closest we mean not only spatially close but also with similar attributes. We first show how to compute the spatial error, then we explain how to incorporate an attribute error in order to orient the simplification toward a given application.

When computing the spatial error of a pair (l_1, l_2) of lines, we want to favor pairs of lines that could be clustered with the smallest possible ε. We thus define the spatial error $E_s(l_1, l_2)$ of an ε-group relative to the ε-line l chosen to represent it by the maximum thickness of the hull associated with l. This heuristic favors the clustering of the thinest groups first. This error is normalized between 0 and 1 using a division by ε:

$$E_s(l_1, l_2) = \max_{x \in [0,1]} ||l^+(x) - l^-(x)||/\varepsilon$$

The user can also define an attribute error measure $e_a(p_1, p_2)$ (normalized between 0 and 1) for a particular attribute space if he or she wants to take it into account in the clustering process. For the single attribute we used in our implementation (e.g. color), we found that a mean was better than a max to give a good estimation of the total error between two groups. This gives the following attribute error:

$$E_a(l_1, l_2) = \int_0^1 e_a(l^+(x), l^-(x))dx$$

The spatial and attribute error measures are then classically combined in a multiplicative way to give the error measure $E(l_1, l_2)$:

$$E(l_1,l_2) = 1-(1-E_s(l_1,l_2))*(1-E_a(l_1,l_2))$$

The attribute error is only computed for ε-groups, that is groups of lines that can be spatially clustered. In order to forbid clustering if the attributes of the lines of the group are too different, we add the constraint that for an ε-group (l_1,l_2) to be clustered, it must satisfy $E(l_1,l_2) < 1$.

7.2.4 Closed curves

Most of this method holds for closed curves and the algorithm is very similar. However, we need to implement some additional processes. First, the lines closed at the scale ε are detected. Those are the lines whose endpoints are at a distance less than ε. Moreover, when identifying paths, if the path configuration found in Fig. 7.4-(c) arises, the resulting ε-line becomes closed.

7.3 Implementation details

We have implemented the greedy iterative clustering by an edge collapse algorithm applied on a graph whose edges represent pairs of ε-lines which are ε-groups.

7.3.1 Preprocessing input lines

The lines we take as input can be of any kind. The only constraints are that we need to sample them. In our implementation, we use regularly-sampled Catmull-Rom splines. For all distance computations involving such samples, we use an acceleration grid of cell size ε, allowing us to quickly find candidate samples.

In order to initialize the algorithm, we need to convert an initial line l into an ε-line. To do that, we follow l, progressively creating its two offset curves, and we split l as soon as it crosses one of the already created offset curves. Note that this splitting process gives different results depending on the extremity at which one starts. We have not found any remarkable difference in the results, however one may want to choose a more symmetric way of splitting. After this initialization step, each line is its own hull.

7.3.2 Building the graph

Once the input lines have been converted into ε-lines, we build a graph with a node for each input ε-line, and an edge between each pair of nodes whose ε-lines form an ε-group.

A pair of ε-lines can only be clustered if it corresponds to one of the configurations shown in Fig 7.4-(b),(c). Thus, for an ε-line pair, if there are more than two extremities at a distance greater than ε from the other ε-line, we can reject it directly, saving a lot of computation time.

In practice, we compute a hull for each potential cluster and store it on the corresponding edge along with its error.

7.3.3 Updating the graph

Then, at each step of the algorithm, we collapse the edge with minimum error and update the graph edges locally. Collapsing an edge is done by creating a new node that stores the edge's ε-line and hull. The collapsed edge is deleted and by definition of a hull, we only have to inspect the edges incident to the collapsed nodes. Those edges are removed from the graph and new edges are created between the new node and its neighbors. We also compute the attributes of the new ε-line by linearly interpolating the attributes of the two original ε-lines.

Finally, the two collapsed nodes are removed from the graph. But instead of deleting these nodes, we keep them in a history of collapse sequences which is stored as a tree under the newly created node. This gives us access to the underlying input lines and the collapsing scheme of each cluster.

The algorithm stops when no more clusters can be created.

7.4 Results

In this section, we give some results to illustrate the overall simplification process, i.e. both clustering and geometric stages for each of the target applications: density reduction, level-of-detail and progressive drawing. The geometric stage is clearly a more specialized operation since the choice of the new line to be drawn is left to the chosen strategy. We implemented two "standard", pre-defined strategies:

- **Average line:** the new line interpolates all the original lines in the cluster (with application-defined weights);
- **Most significant line:** the new line is one of the original lines, chosen according to an application-defined priority measure (based on length, nature...)

In the worst case, the total process increases quadratically with the number of input lines at a fixed scale parameter ε. In practice, for our examples, it ranges from several seconds to a minute. For each example we give the number of input lines and resulting clusters. The simplification scale is shown by a circle of diameter ε.

Density reduction Fig. 7.8 shows a straightforward illustration of our approach. Lines have been extracted from a scanned line drawing. The user chooses a simplification scale ε and the lines are simplified. We applied an average line strategy without smoothing the results, so that simplified lines exhibit the ε-line of each group.

Fig. 7.9 shows a similar scenario that takes the color attribute into account. The attribute error is a $L^*a^*b^*$ color distance.

Figure 7.10 shows the use of categories to separate lines of different nature: external contour on one side and internal and suggestive contours [DeCarlo 03] on the other side. In examples coming from 3D renderings like this one, we make use of object IDs and line nature to detect categories automatically. This allows for the use of two different geometric strategies: for the external contour an average line is drawn, whereas the longest line of each cluster is drawn for the internal and suggestive contours. Moreover, the external contour is simplified at a larger scale than the other lines. Resulting lines are better organized and keep the most salient features of the model.

(a) (b) (c) (d)

Figure 7.8: **Density reduction:** (a) The original scanned and vectorized drawing: 357 input lines - (b) The resulting simplification: 87 clusters - (c,d) Zoom on the above images. The scale ε is indicated by the circle in the upper left corner.

Level-of-detail Fig. 7.11 shows an example of a LOD sequence produced with our approach. Progressively scaling down a drawing is equivalent to choosing an increasing ε. Thus we apply a series of simplifications with an ε step, each time starting from the previous, finer level. Here again, two different geometric strategies are used: the average line for the contour and the longest line for the hatchings. To do that, we created five categories by hand: one for the contours, and one for each of the four orientations for hatchings. Note that although no particular treatment was applied to preserve tone across simplifications, this result is quite convincing. Tone preservation could be explicitly included in the method at the geometric stage, by choosing appropriate line attributes such as width and/or color.

Progressive drawing Fig. 7.12 shows a drawing sequence using our progressive drawing tool. Here, the clustering algorithm is applied iteratively: The user chooses a sensitivity ε and draws a sketched line over an initial drawing; the lines are then simplified; and finally, the resulting lines constitute the initial drawing for the next step. This tool requires an additional feature: we only want the simplification to be done between initial lines and the new sketch. Thus the input lines are organized in two sets: the initial lines and the new sketched line. During the clustering, only the edges between pairs of nodes that lie in different sets are built.

Finally we choose a priority-based strategy because we want the last drawn line to have a greater priority than initial lines. In practice, that consists of using an average line strategy, giving greater weights to samples belonging to the last drawn line. This is made possible by the history tree stored at each cluster. We found this tool to be very intuitive, particularly for modifying lines coming from 3D renderings or extracted from images.

7.5 Discussion

In this work we opted to remain very general, trying to find the common properties of some target simplification methods. However, it is clear that such a low-level method can still be specialized to adapt to other specific applications. In particular, we believe

that the separation of the clustering and geometric stages is crucial for all simplification methods: they correspond to analysis and transformation stages.

Other attributes than color could be used in the attribute error definition. However, we did not consider input lines exhibiting wiggling patterns and implicitly assumed that they come at an appropriate scale. The problem of extracting the so-called natural scale of a line has been previously addressed (e.g. [Rosin 98]).

Our method is invariant under rotation, scale, and translation, since it operates only on euclidean distances between pairs of points. However it has two limitations: it is not transitive and it prevents simplifying forks. The former means that simplifying a drawing at scale ε_1, then simplifying the result at scale $\varepsilon_2 > \varepsilon_1$ is not guaranteed to provide the same result as a direct simplification at scale ε_2. However, this is not a problem in the applications we envision, for instance generating a discrete set of LOD representations. The latter assumes that the problem of forks is rather separate from geometric clustering (appearing at a higher level of processing and depending on the application) and thus is left as a post process.

The choice of a greedy algorithm for clustering implies that we only reach a local optimum in general. This turns out to be sufficient in practice for the applications we have tested. Other optimization techniques could be used if reaching a global optimum is important.

The evaluation of a simplification method for line drawings is not an easy task. Indeed, there is no simple and obvious quality measure for a simplified drawing. Visual evaluation involves a number of high-level interpretation processes, which are difficult to model and quantify. Our approach offers the convenience of a guaranteed geometric criterion: the resulting drawing is "within a distance ε" from the original drawing. The direct evaluation of the result is the number of clusters. However, in an attempt to provide finer evaluation tools we identified two other measures of what we termed *complexity* in Chapter 6. First, the reduction in the number of lines composing the drawing; Second, the variation of the total arc-length in the drawing. Both are strongly related to the geometric strategy chosen for an application: keeping a line per cluster clearly decreases the total number of lines, and the arc-length may strongly vary depending on the new lines created. For instance, in Fig. 7.8 the number of lines was divided by 4 and the arc-length reduced by 60%.

In the examples shown, we observe that a purely distance-based simplification should generally not be applied to all lines of the drawing at once, because there are categories of lines that should not be clustered, or with a lower priority: think for instance of lines depicting different objects placed near each other. Segmenting the drawing and applying the simplification algorithm to each category is better for the scope of an automatic process. Naturally this opens the question of how to segment or define the categories automatically, which is discussed in Chapter 9.

Finally, we think that our approach could be extended to animation. The idea would consist of guiding the clustering stage temporally, not only to ensure temporal coherence, but also to cluster lines that go altogether across time. This could be accomplished by incorporating another perceptual grouping criterion: common fate, which states that the visual system tends to group elements with similar velocity.

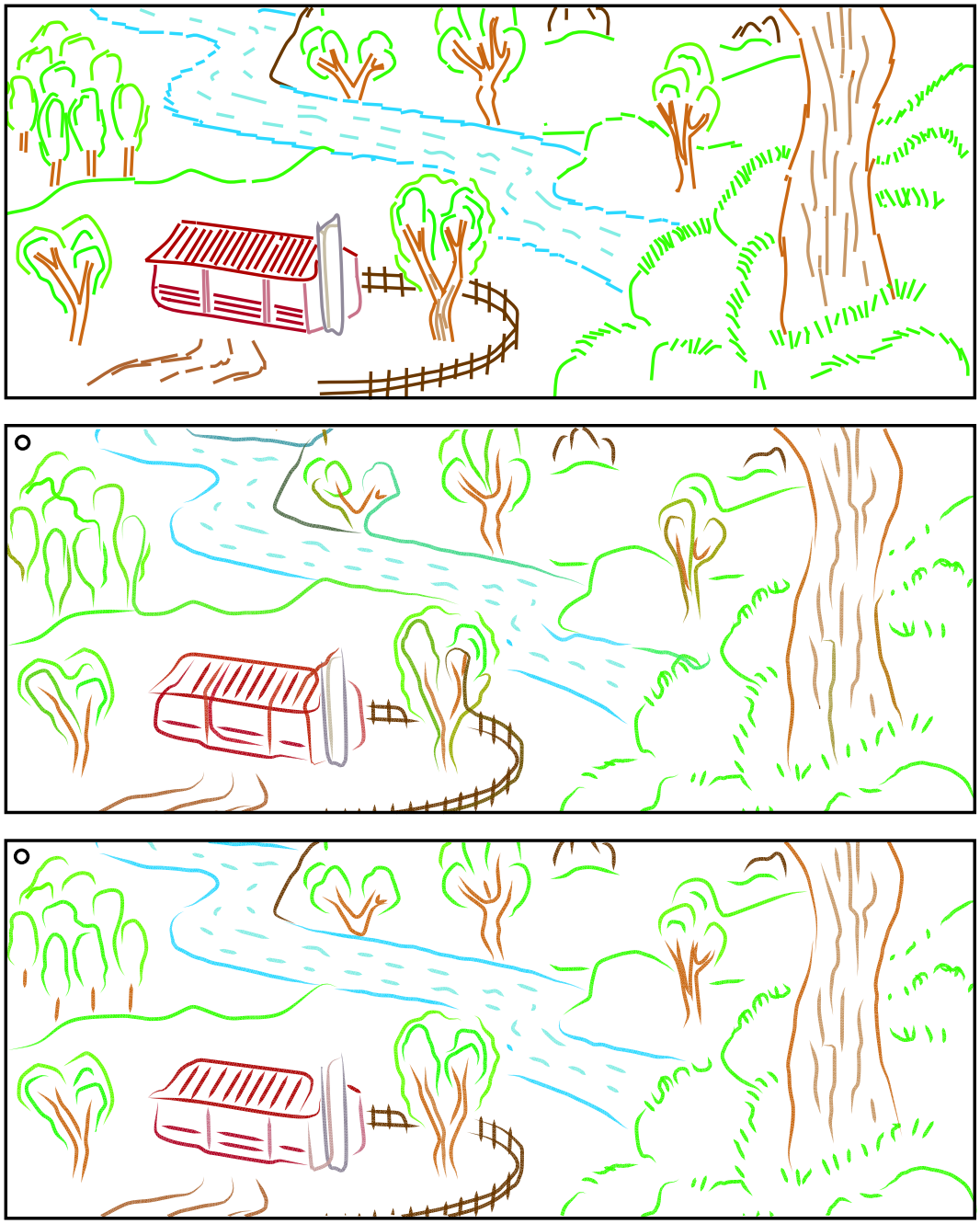


Figure 7.9: Top: input drawing. Middle: simplified drawing without taking color error into account during the clustering stage. Bottom: taking color error into account better preserves the original drawing (see the fence and the tree, the trunks and the leaves...).

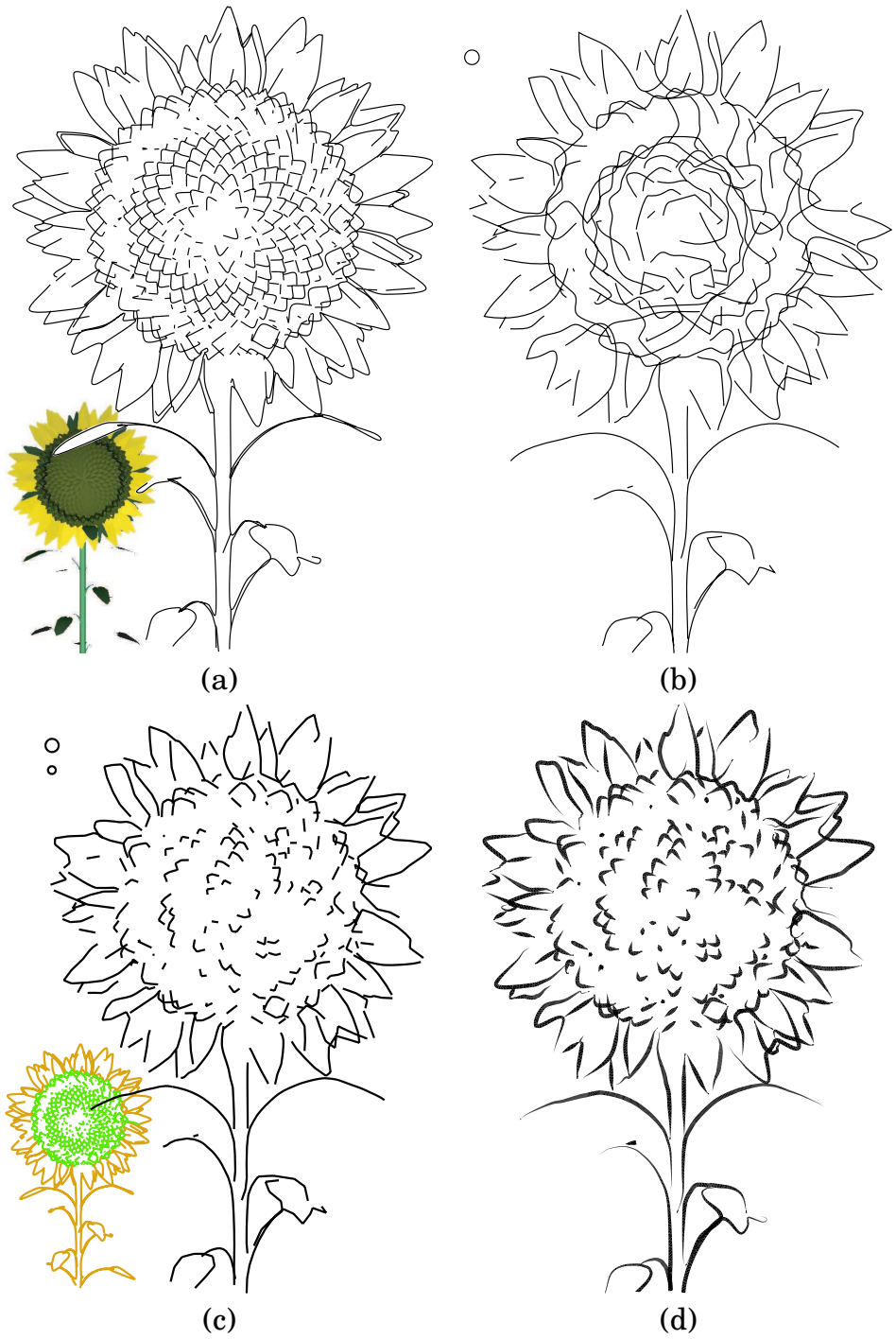


Figure 7.10: **Simplification of a 3D rendering:** (a) 3D model and its line rendering using silhouettes and suggestive contours (531 input lines) - (b) Simplification without any category (256 clusters) - (c) Using two categories (external and internal contours) each with a different scale and geometric strategy (294 clusters) - (d) Same result in a calligraphic style.

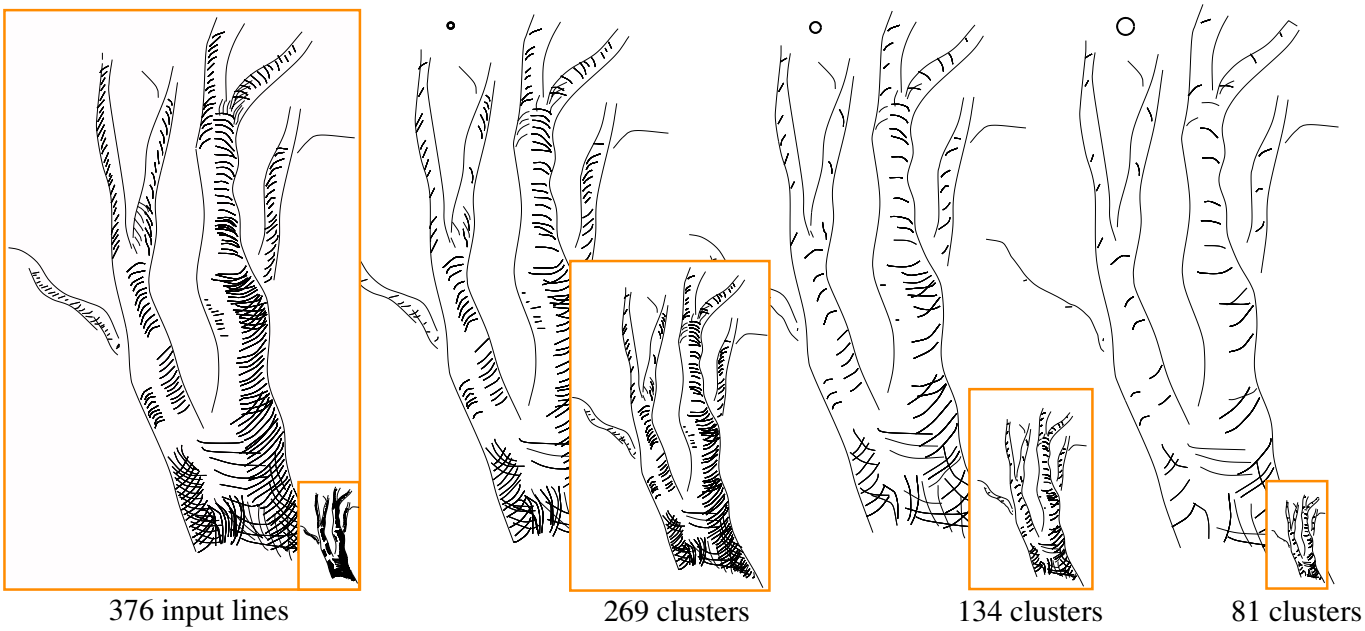


Figure 7.11: **LOD:** A series of LODs made by progressively increasing ε. Using different categories and geometric strategies prevents undesired hatching lines from merging. Compare the small resized images with (right) and without (left) simplification.

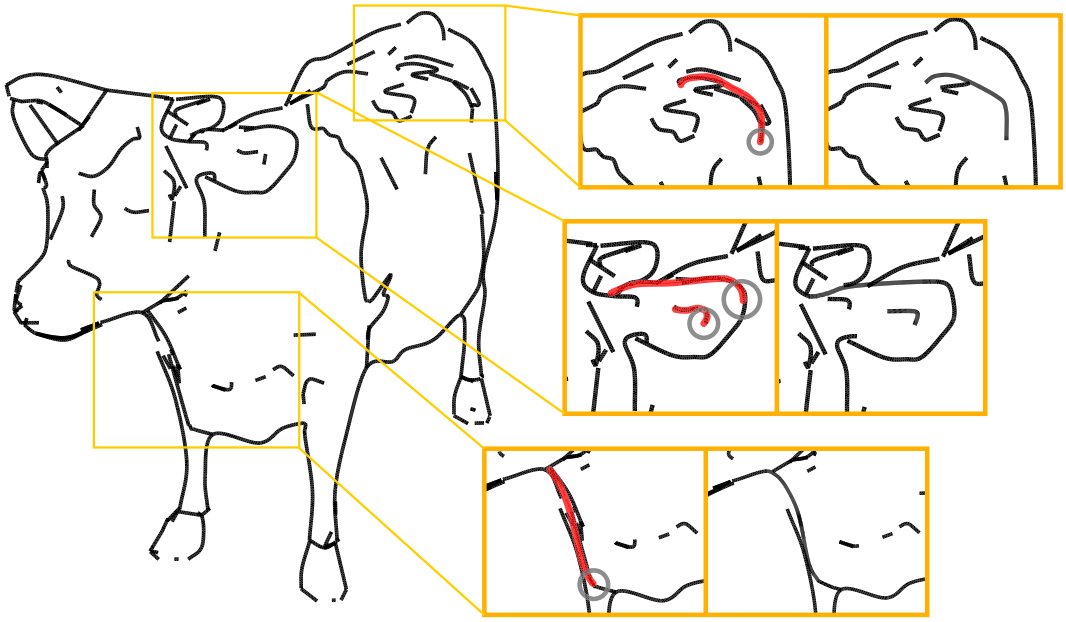


Figure 7.12: **Over-sketching:** A 3D model has been rendered in a line drawing style. The user adds new lines (in red) which are clustered with the old ones; the scale ε (gray circle) can be changed at each step.

Chapter 8

Stroke pattern synthesis

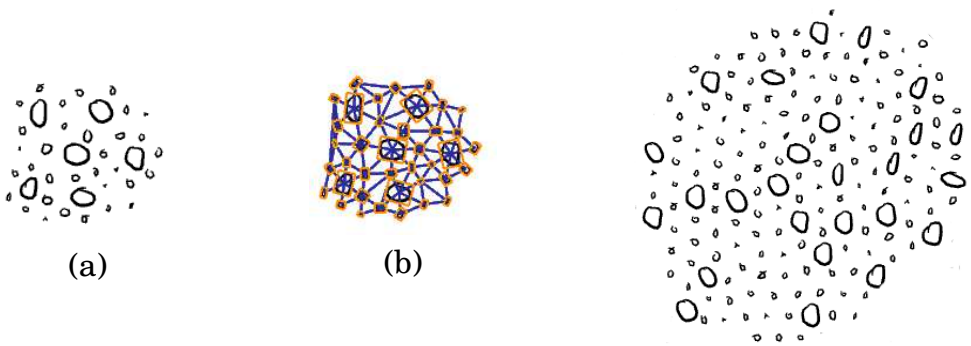


Figure 8.1: Our method (a) takes as input a reference vectorized stroke pattern, then (b) analyses it to extract relevant stroke pattern elements and properties in order to (c) synthesize a similar pattern.

In the previous chapter, we explored the analysis of drawings composed mainly of silhouette lines, sometimes with the addition of simple hatching patterns to depict illumination and shape. Another particularly important class of drawings is that of stroke-based images. Various styles such as etchings, pen-and-ink and oil painting renderings can be thought of as stroke-based styles as described in Hertzmann's survey [Hertzmann 03]. The rendered strokes can be either used to fill in 2D regions, or to annotate 1D paths. In both cases, the generation of appropriate stroke arrangements for these styles remains a difficult or tedious process to date. We propose a method that is a compromise between automation and expressiveness to create stroke patterns from examples.[1]

Synthesis by example appears to be the best way to address this question. However, pixel-based texture synthesis is not well suited to strokes, in part because each stroke is individually perceptible, in contrast to pixels. Organized stroke clusters such as those found in hatchings are difficult to extract and reproduce at the pixel level. Moreover, some variation in the synthesized drawing is desirable to avoid too

[1]This work has been done in collaboration with Professor Lee Markosian and his student Simon Breslav at the University of Michigan, and my advisors Joëlle Thollot and François Sillion. It has been published in Computer Graphics Forum (Proceedings of Eurographics) in 2006, see [Barla 06b].

much regularity, and it would be difficult to achieve such variation with pixel-based texture synthesis.

We therefore propose to use a vector-based description of an input stroke pattern supplied by the user. This allows for greater expressiveness and higher-level analysis than would be afforded by a per-pixel approach. We target any kind of stroke patterns (stippling, hatching, brush strokes, small figures) with a quasi-uniform distribution of positions in 1D and 2D (i.e., along a path or inside a region). The stroke attributes can vary in non-uniform ways and the only parameter required from the user is the scale of the meaningful elements of the pattern. Then, in a manner analogous to texture synthesis techniques, we organise our method in two stages (see Figure 8.1). An analysis stage where we identify the relevant elements in terms of stroke patterns and their distribution, and a synthesis stage where these elements are placed in the image so as to reproduce an appearance similar to the reference pattern.

In a first attempt to extend synthesis to more general stroke patterns, we developed a statistical approach for the case of hatching and stippling in 1D and 2D [Barla 06a]. No neighborhood comparison was taken into account and the class of possible elements was reduced to single-colored points or lines. Here we target a wider range of patterns (including color patterns) and take into account the global organization of the pattern by means of neighborhood comparisons.

Contributions

To define a stroke pattern we draw upon research in human vision, more specifically in the field of perceptual organisation. Indeed, there is a common agreement that the human visual system, in the early stages of perception, structures 2D information into elements based on a set of criteria such as proximity, parallelism, and continuation [Palmer 99]. In the case of stroke patterns, this means that some sets of input strokes are perceived as single elements, and at a higher level, the distribution of elements defines the pattern.

Our first contribution is an analysis method that extracts an intermediate-level description of vector data, using perceptual organisation criteria applied to input strokes. In particular, we are able to analyse strokes of different styles: not only stippling and hatching strokes, but also brush strokes and small figures (see Figure 8.6 and Figure 8.7).

Our second contribution is a synthesis method analogous to texture synthesis, but operating on vector data. We propose a perceptually-based neighborhood matching algorithm that allows comparisons between neighborhoods even when they have dissimilar connectivity.

8.1 Analysis

The first step of our method aims at analyzing a reference stroke pattern to extract the meaningful elements that constitute the pattern, as well as their distribution. We define an element as a cluster of strokes that is perceived as a single feature by the user. Previous work [Saund 94] described methods to analyse diagrams where junctions and closure properties are of primary importance. In our approach, we target a quasi-uniform distribution of elements that have sensibly the same size; Here

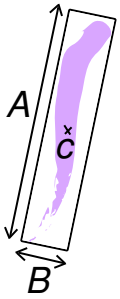
parallelism, proximity and continuation are more important properties.

Moreover, there is a characteristic scale of the pattern that gives the size of pattern elements. We let the user specify it so that there is no ambiguity regarding user intention in the drawn pattern. Note, however, that the whole analysis process is interactive, hence providing enough feedback to easily set a convenient scale. This allows us to target a wide range of patterns as shown in Section 8.3.

Once the scale is chosen the rest of the process is fully automatic and works as follows. We first fit an element to each input stroke; We then cluster elements iteratively using perceptual organisation criteria; Finally, we relate elements to each other to derive properties about their distribution.

8.1.1 Element fitting

We define an element by its center and its two elongation axis. In order to fit an element to a stroke, the user can either choose to only consider the skeleton of the stroke (*i.e.* the gesture input by the user) or to also take into account its style (thickness, fading, tapering, etc). In the first case, we fit an element to the points that define the skeleton, while in the second case, we fit an element to the points of its contour. The following fitting method is efficient for the elements we consider.



An element E is constructed by fitting an oriented bounding box to the chosen set of points (skeleton or contour) that will prove useful for approximating geometric measures between elements. We first fit a Gaussian distribution to the points to compute the two principal elongation directions given by the eigenvectors A and B of the points' covariance matrix. Each point is then projected onto each axis to compute the size and center c of the bounding box.

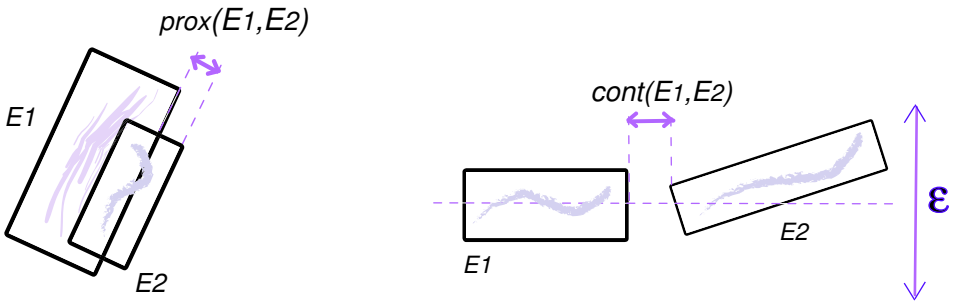
8.1.2 Element clustering

Now that each stroke is represented by its bounding box, we want to cluster them into elements at the chosen scale. For example, a hatch element can be made of several overlapping strokes, and a small figure (think of a flower) is often drawn using a small number of individual strokes.

To decide whether two elements can be clustered, we draw upon the work of Etemadi et al. [Etemadi 91] on perceptual line segment grouping. Two elements are clustered if they meet a proximity constraint (*e.g.*, for a flower), or if they meet a continuation constraint (*e.g.*, for a hatch element). These tests are performed against the user-defined scale ε, that represents the minimum distance at which two elements are perceived separately. When two elements are clustered, we merge their respective strokes and fit a new element using the method explained in the previous section. For more complex elements, a more general technique like the one we presented in the previous Chapter should be employed.

Clustering is performed by a greedy algorithm that processes the strokes in the order they have been drawn. The fitted elements are first placed into a queue. At each step, an element E^* is popped and every other element E_i in the queue is tested for clustering based on an element pair comparison. If the test is successful, we merge E_i into E^* and remove E_i from the queue. After all the elements of the queue have been tested, if any clustering occurred, E^* is pushed back into the queue. Otherwise, E^* is

added to the output list of clustered elements. The algorithm repeats until the queue is left empty.



(a) (b)

Figure 8.2: Element clustering uses two perceptual measures: (a) Proximity and (b) Continuation.

Proximity measure

The proximity between two elements E_1 and E_2 is computed using Hausdorff distances so that nested, or very close objects are clustered together (see Figure 8.2(a)):

$$\begin{aligned} prox(E_1,E_2) &= \min(d_H(E_1,E_2),d_H(E_2,E_1)) \\ d_H(E_1,E_2) &= \max_{q_1\in E_1}(\min_{q_2\in E_2}(d(q_1,q_2))) \end{aligned}$$

where $d_H(E_1,E_2)$ is the directed Hausdorff distance.

In practice, it is computed using point-line distances between the bounding boxes. If $prox(E_1,E_2) < \varepsilon$, then E_1 and E_2 are clustered.

Continuation measure

Continuation has to be checked on all pairs of axes between the two elements E_1 and E_2. For each of the four configurations, we first have to ensure that the elements are near-collinear; then we compute a continuation measure. Without loss of generality, we only consider the measures of E_2 relative to the axis A_1 of E_1. E_2 is near-collinear to E_1 iff:

$$\forall p \in E_2, d(p,A_1) < \varepsilon/2$$

If the above condition is met, then a continuation error is computed as the gap between the projected points of E_1 and E_2 on A_1 (see Figure 8.2(b)):

$$cont(E_1,E_2) = \min_{p\in E_1^*, q\in E_2^*}(d(p,q))$$

where E_i^* is the set of points of E_i projected on A_1, $i = 1,2$.

In practice, we also use the bounding boxes to speed up this computation. If $cont(E_1,E_2) < \varepsilon$ for any of the four configurations, then E_1 and E_2 are clustered.

8.1.3 Element distribution

Having identified the elements of our reference pattern, we can now extract connectivity information among them in order to characterize their distribution (see Figure 8.1(b)). We use the center position of each element. For 1D patterns, we extract the neighbors along a chain, while for 2D patterns, we extract a Delaunay triangulation and keep only edges that are part of at least one unskewed triangle (*i.e.* a triangle that do not have an angle greater than $\frac{2\pi}{3}$). The pattern input by the user is supposed to be uniform, but in practice, the distribution of elements is only close to uniform. We measure locally this variation by computing a shift vector S_{ref} that expresses the displacement between an element's position and the barycenter of its neighbors' positions. We will use those measurements to add variation to a synthesized pattern in Section 8.2.3.

8.2 Synthesis

Thus far, we analysed the reference stroke pattern in order to get a higher-level, perceptually meaningful description of it. In this section, we show how to take advantage of this knowledge during synthesis. Since we want to address the synthesis of texture-like patterns, it makes sense to take inspiration from the texture synthesis literature. In our approach, we draw comparisons with Efros and Leung's pioneering paper [Efros 99]: like them, we use a causal synthesis procedure that starts with an element at the center of the pattern and expands it outward using neighborhood comparisons on the previously synthesized elements.

Our method exhibits some important differences though: contrary to the distribution of pixels on a grid, our element positions are not supposed to be aligned. This has an impact on the neighborhood comparison procedure that has to match relevant neighbors between the reference and target patterns. Moreover, elements are easily identifiable and perceived in isolation, thus their comparison should consider the whole set of their parameters: orientation, length, width and color. To do that, we draw inspiration from the field of perceptual organisation once again and show how a trade-off between variation and fidelity can be obtained. The algorithm below summarizes the synthesis process.

Algorithm 1 Stroke pattern synthesis

$D_{tar} \leftarrow$ InitialiseDistribution(D_{ref})
$E_{start} \leftarrow$ GetCenterElement(D_{tar})
for each E_{tar} in D_{tar} growing outward from E_{start} **do**
  $E_{ref} \leftarrow$ FindBestMatch(E_{tar}, D_{tar}, D_{ref})
  $E_{tar} \leftarrow$ SynthesizeElement(E_{ref}, D_{tar}, D_{ref})
end for

We first present our neighborhood comparison in Section 8.2.1, before describing in Section 8.2.2 how it is applied iteratively to create the target pattern. Finally, in Section 8.2.3, we explain how the user can add variation to the synthesized pattern while keeping a strong similarity with the reference.

8.2.1 Synthesizing one element

Let E be an element in the synthesized pattern and assume for the moment that all elements in the pattern except for E are known. Let $\omega(E)$ be a neighborhood around E. To assign properties to E, as in [Efros 99], a set Ω of neighborhoods similar to $\omega(E)$ is extracted from the reference pattern using various perceptual measures described below. Then one of the neighborhoods in Ω is randomly chosen and the center element of the picked neighborhood is used for E.

Neighborhood comparisons are more complex for stroke pattern synthesis than for texture synthesis for two reasons: the number and position of neighbors vary in our distributions, and elements are more complex entities than pixels. The computation of the similarity between two neighborhoods ω_{ref} and ω_{tar} is thus performed in two steps. First, relevant elements of this pair of neighborhoods are found. Second, a set of perceptual organisation measures is tested against perceptual similarity thresholds for the whole candidate reference neighborhood. Comparisons between vector-based elements has already been studied in the context of graphical search and replace by Kurlander et al. [Kurlander 88]; However, their method is more tailored to exact matches, and only compares pairs of elements, not neighborhoods.

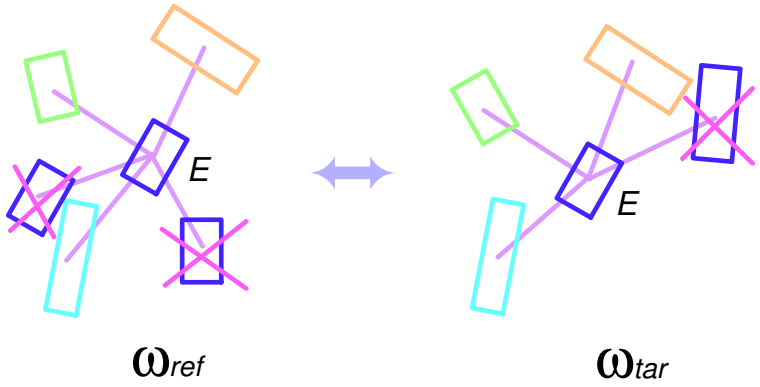


Figure 8.3: Neighborhood comparison: Pairs of relevant elements are extracted based on their position.

For the determination of relevant elements, we only consider pairs of closest reference and target elements by comparing their positions, see Figure 8.3. This heuristic locally matches the distribution of element positions between the reference and target patterns. We keep pairs of elements $E_{ref} \in \omega_{ref}$ and $E_{tar} \in \omega_{tar}$ such that

$$\begin{aligned} E_{tar} &= \arg\min_{E \in \omega_{tar}} (d(E_{ref}, E)) \\ E_{ref} &= \arg\min_{E \in \omega_{ref}} (d(E_{tar}, E)) \end{aligned}$$

where $d(E_1, E_2)$ is the euclidean distance between the centers of E_1 and E_2.

Once the elements are matched, we compute a set of four different perceptual measures that can be organized in two categories (see Figure 8.4): a *shape-matching* measure that compares two elements as point sets by computing a symmetric Hausdorff distance; and a set of three measures - *parallelism, overlapping and superimposition* - that compares elements using the higher-level description extracted during the analysis. The shape-matching measure is useful when there is an ambiguity between

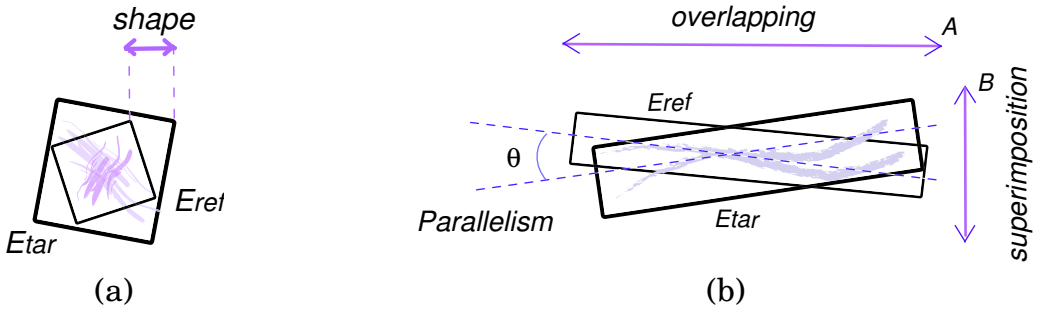


Figure 8.4: Element neighborhood matching uses various perceptual measures: (a) isotropic elements; (b) anisotropic (elongated) elements.

the principal and secondary axis of elongation of an element (*e.g.*, a circle), while the other measures exploit the perceptual properties of elongated strokes.

Each measure is computed independently on each pair of elements in their respective frame; *i.e.*, the element centers are first aligned before the following measures take place:

$$\begin{aligned} shape(E_{ref},E_{tar}) &= \max(d_H(E_{ref},E_{tar}),d_H(E_{tar},E_{ref})) \\ par(E_{ref},E_{tar}) &= |\theta(A_{ref},A_{tar})| \\ ov(E_{ref},E_{tar}) &= \max\left(\frac{|A_{ref}|}{|A_{tar}|},\frac{|A_{tar}|}{|A_{ref}|}\right) \\ sup(E_{ref},E_{tar}) &= ||B_{ref}|-|B_{tar}|| \end{aligned}$$

For the computation of the shape measure, we approximate the directed Hausdorff distance d_H using bounding boxes as previously. The parallelism measure is simply taken to be the norm of the angle θ between the two elements principal axes. Overlapping is the maximum ratio of lengths between the target and reference principal axes. And superimposition is the difference in thickness (length of the secondary axis) between the target and the reference.

In addition to these geometric measures, any attribute can also be taken into account during the synthesis. We illustrate this ability with a simple color distance:

$$col(E_{ref},E_{tar}) = d_{RGB}(C_{ref},C_{tar})$$

where C_{ref} and C_{tar} are the colors of E_{ref} and E_{tar} in RGB.

All the measures are then averaged over the element pairs of ω_{ref} and ω_{tar} to give a set of perceptual measures between neighborhoods. They are then tested against a set of perceptual thresholds. These thresholds control the amount of selected candidate neighborhoods: they have to be sufficiently large to provide enough candidates, but small enough to avoid incoherence. In our experiments, we use $\sigma_{shape} = 0.1L$ where L is the average length of the reference elements, $\sigma_{par} = \frac{\pi}{20}$, $\sigma_{ov} = 1.5$, $\sigma_{sup} = 0.1L$ and $\sigma_{col} = 0.15$. We observed that our algorithm is robust to small variations in these thresholds. Two neighborhoods are then considered similar relative to a given measure m iff $m(\omega_{ref},\omega_{tar}) < \sigma_m$.

The measures are finally combined to determine the similarity of the candidate reference neighborhood to the target one. If the colors or any other attribute does

not match, we simply discard the matching. Otherwise, we test whether the neighborhoods match by considering them as sets of points or sets of elements. In our approach, we thus use the following combination

$$col \textbf{ and } (shape \textbf{ or } (par \textbf{ and } ov \textbf{ and } sup))$$

In Section 8.3, we show and comment examples that exhibit the role of each measure: hatching strokes are more discriminated by parallelism, overlapping or superimposition, while small figures rely mainly on the shape measure; color is independent of the shape of elements, but further refines the above measures.

8.2.2 Synthesizing a pattern

As mentioned previously, our synthesis algorithm begins at the center of a uniform distribution similar to that of the reference one. Hence, we first build a distribution of element positions that we call seeds, and connect seeds together to get neighborhood relationships. To this end, as in our previous method [Barla 06a], we use Lloyd's method [Lloyd 82] in 1D and in 2D: it is an iterative algorithm that starts with a random distribution of seeds. Then, at each step, a Voronoi diagram of the seeds is computed, and each seed is moved to the center of its Voronoi region. It converges to a centroidal Voronoi tessellation, close to regular. The only parameter of the method is the number of seeds, that we set to $N_{tar} = N_{ref}.\mathscr{A}_{tar}/\mathscr{A}_{ref}$, where $\mathscr{A}_{ref}$ and $\mathscr{A}_{tar}$ are the areas of the reference and target regions respectively. Finally, when the algorithm has converged, for 1D patterns, we extract the neighbors along a chain, while for 2D patterns we extract a Delaunay triangulation and keep only the edges which are part of an unskewed triangle, in order to avoid degenerate edges at the border of the triangulation.

In the previous section, we have discussed a method of synthesizing an element when its neighborhood elements are already known. Unfortunately, this method cannot be used directly for synthesizing the entire pattern since for any element, only some of its neighbors will be known during the propagation. Like in Efros and Leung's approach, the element synthesis algorithm must be modified to handle unknown neighborhood elements. This can be easily done by only matching on the known values of $\omega(E)$ and normalizing the error by the total number of known elements. This heuristic, illustrated in Figure 8.5 appears to provide good results in practice.

8.2.3 Adding variation

The patterns synthesized with our method exhibit strong similarities with the reference, since they consist of elements that have been copied from it. One might also wish to introduce some amount of variation relative to the reference pattern. In order to add such a variation, we developed a post-processing mechanism that slightly changes the parameters of the synthesized elements and is controllable by the user via a slider. For each element, and for each of its parameters independently, we select a set of similar values in the reference pattern. E.g. , we select a set of orientations close to the synthesized element's orientation. Then, we pick one value from this set and use it in place of the parameters of the synthesized element. This mechanism

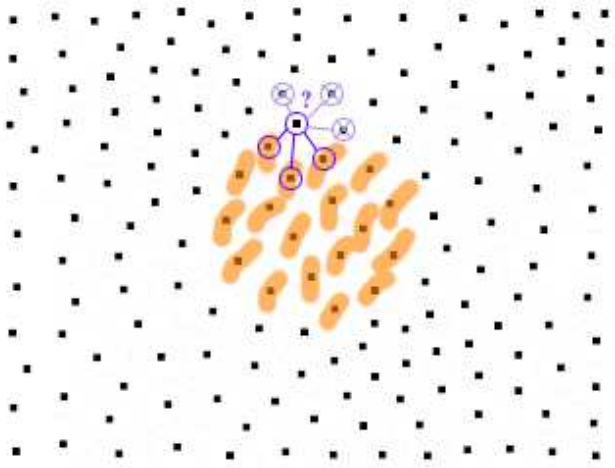


Figure 8.5: To synthesize a 2D pattern, we first distribute seeds using Lloyd's method, then we synthesize elements growing outward from the center using partial neighborhood comparisons.

lets us exchange parameters between similar reference elements without producing elements that are too different from those of the reference pattern.

Another noticeable difference between our synthesized patterns and their reference is the distribution of positions: while the distribution of a synthesized pattern can be considered uniform, this is not the case of the pattern input by the user. The variation present in the input might be desired by the user, and we thus propose a heuristic to reintroduce variation in the distribution of element positions as a post-process. For each synthesized element E_{tar} that has $n \geq k$ neighbors, we get the reference shift vector S_{ref} of the corresponding E_{ref}, computed during the analysis (see Section 8.1.3); Then we position all the E_{tar} in parallel at the barycenter of their neighbors, and translate them by $S_{tar} = (S_{ref}\mathscr{A}_{E_{tar}})/(n\mathscr{A}_{E_{ref}})$ where $\mathscr{A}_{E_{ref}}$ and $\mathscr{A}_{E_{tar}}$ are the areas of the neighborhoods of E_{ref} an E_{tar} respectively. In practice, we use $k = 2$ for 1D patterns and $k = 4$ for 2D patterns.

8.3 Results

We show here some results of our synthesis method using various types of elements in 1D and 2D. Computation times are of the order of a second for 1D patterns; and between 5 and 10 seconds for 2D patterns, depending on the neighborhood size.

Figure 8.6 shows 1D synthesis. A simple example is shown in Figure 8.6(a) where curved hatching strokes are drawn with sketchy gestures and are properly analysed and synthesized. Our post-process that adds variation to both position and element parameters is illustrated in Figure 8.6(b) with a simple hatching pattern; Notice how the vertical positions of elements are reintroduced in the synthesized pattern. We then show how we can reproduce smooth variations in the parameters of the elements along the 1D path. In Figure 8.6(c), we use a 5-ring neighborhood synthesis to reproduce the smooth change in the orientation of elements along the path; Here the parallelism measure plays a major role in the synthesis. Figure 8.6(d) shows the influence of the neighborhood size on the quality of the result. Synthesised patterns with 1-ring, 3-ring and 5-ring neighborhoods are shown from top to bottom: the

smooth change in stroke length is only well captured with the 5-ring neighborhood. Here, the overlapping measure is discriminant. Figure 8.6(e) shows a smooth change in element thickness captured with a 5-ring neighborhood, and made possible by our superimposition measure. Finally, we show at the bottom an example using brush strokes of alternating colors: the synthesized pattern exhibits the same alternation, thanks to our color measure.

Figure 8.7 shows 2D synthesis. Figure 8.7(a) shows an example of bars oriented in multiple directions. Here, our synthesis method is able to reproduce the complex relations among similar elements. It can also synthesize elements that are less similar, as in Figure 8.7(b) with water drops, small figures like the flowers of Figure 8.7(c) or alternating two different kinds of elements as in Figure 8.7(d). Finally, the variation of element positions is illustrated in Figure 8.7(e), where the distribution obtained with Lloyd's method is modified to be more similar to the input pattern.

8.4 Discussion and future work

8.4.1 Analysis

Our analysis method can extract a wide range of elements (stipples, hatches, brush strokes, small figures), but our element representation (a center and two axes) is too simple to correctly extract long curved strokes. Moreover, we do not target structured patterns, such as a brick wall, where the overall organization should be extracted along with each element. We thus plan to address these two issues in the future by modifying our element model and adding multiple levels of analysis to be able to capture more structured patterns. Another issue is the use of additional perceptual criteria such as closure or junctions in order to perform a deeper interpretation of the input pattern.

In our approach, the analysis stage is user-assisted in order to determine the scale of the pattern. Since the clustering of elements is dependent on this scale and is implemented with a greedy algorithm, it can produce flickering on rare occasions when the user interactively modifies the scale. However, this has no impact on the final synthesis result. On the other hand, if one wants to consider scanned drawings as input, it would make sense to extract both the elements and the scale automatically. The greedy nature of the clustering algorithm might then lead to problematic behaviors. Moreover, for elements that significantly overlap, our approach may not work at high scale values since it uses proximity computations. Some ideas to address these limitations are given in Chapter 9.

8.4.2 Synthesis

Our synthesis method currently generates quasi-uniform distributions of elements via the Lloyd algorithm. In the future we will target non-uniform distributions that can take into account density variations within the pattern. We also plan to use the whole element shape rather than only its center position in the distribution definition. The heuristic we used for finding relevant neighbors also suffers from a limitation: it can happen that no pairing is found between the reference and target neighborhoods. However, in practice, the even distribution produced by Lloyd's method prevents this worst case scenario from happening.

Another interesting point is the ability to take into account attributes of the input strokes during synthesis. We only investigated color, but other attributes such as thickness, opacity or texture might give convincing results. Finally, our variation technique is tailored to stroke pattern synthesis and has thus no equivalent in texture synthesis. We plan to extend this ability to take into account the shape of the strokes.

8.4.3 Extension to synthesis on surfaces

Our system works in the picture plane and we plan to extend it to synthesis on surfaces in the future. However, stroke-based rendering raises several specific questions in terms of rendering. Indeed, the strokes need to be of roughly constant size in 2D if one wants to maintain the same style for every viewpoint. Therefore an LOD mechanism has to be defined, and we believe that this can be achieved with a dedicated synthesis algorithm. An automatic generation of mipmaps or Tonal Art Maps [Praun 01] would be a simple way to address synthesis on surfaces. However, we believe that direct synthesis on surfaces will open more interesting avenues: for example, the rendering could be done by varying the attributes of each synthesized stroke depending on the viewing and lighting conditions. By keeping the analogy with texture synthesis, it should be possible to devise such a method in the same way we did in the present paper.

Figure 8.6: 1D synthesis results. (a) A simple hatching example that uses sketchy strokes; (b) another hatching example with a uniform distribution of elements (on top), and the same pattern after variation have been added (at bottom); (c) a smooth change of element orientation is analysed and synthesized with a 5-ring neighborhood; (d) a smooth change of element length is analysed and synthesized with increasing neighborhood sizes (from top to bottom: 1-ring, 3-ring and 5-ring neighborhoods); (e) a smooth change of element thickness is analyzed and synthesized with a 5-ring neighborhood; (f) the alternation of strokes colors is captured and reproduced in the synthesized pattern.

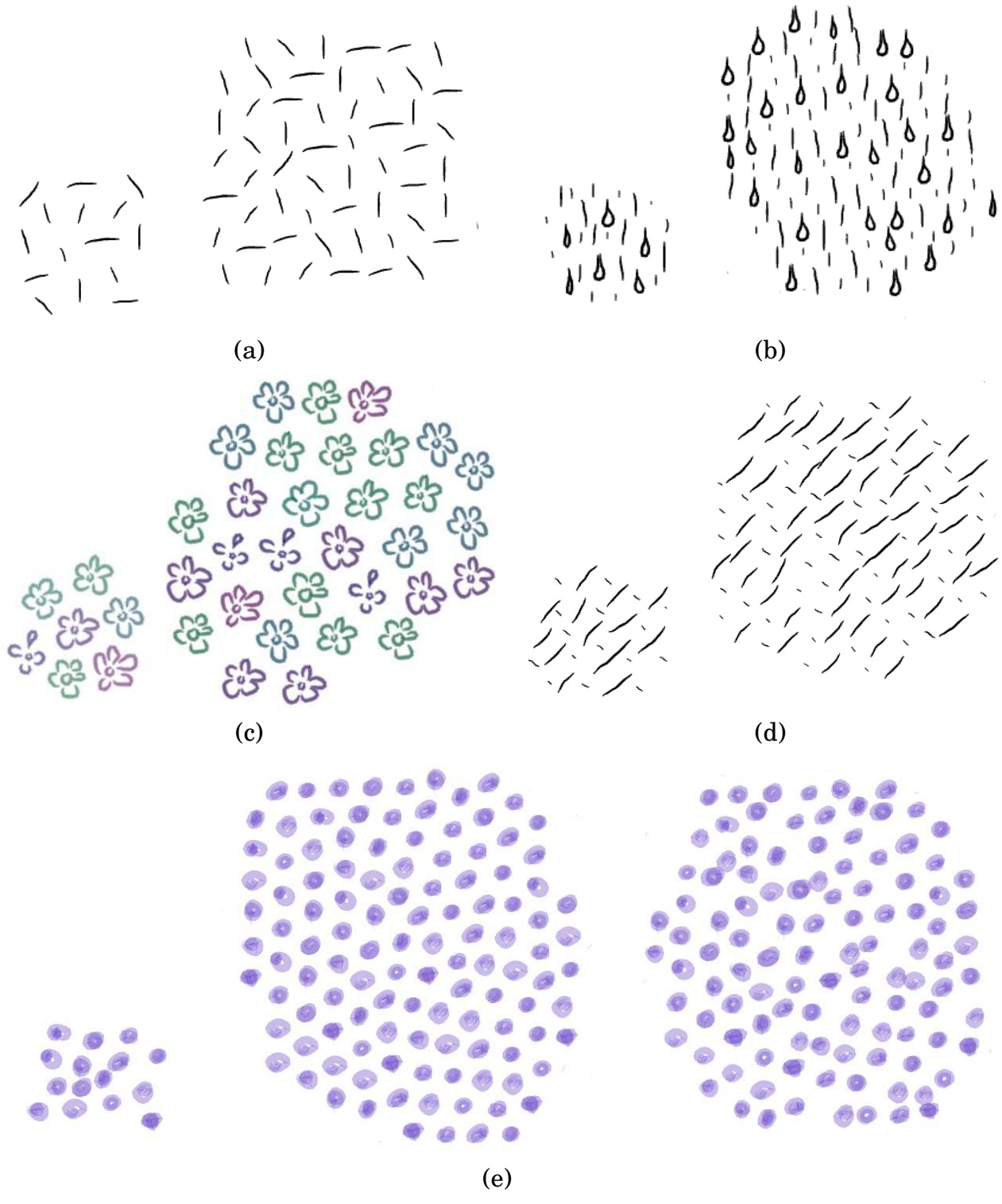


Figure 8.7: 2D synthesis results. (a) A pattern of hatching strokes in multiple directions is synthesized using a 3-ring neighborhood; (b) elements of various nature (water drops, hatches) are analysed and synthesized with a 3-ring neighborhood; (c) small figures like flowers of different colors can be analyzed and synthesized by our method using a 2-ring neighborhood; (d) a pattern composed of two different types of hatching strokes is synthesized with a 2-ring neighborhood; (e) the addition of variation in the position of elements is able to break the uniform distribution of Lloyd's method.

Chapter 9

Remarks on drawing segregation

In the previous two chapters, we proposed two specific methods to enrich the digital drawing toolbox, one for the simplification of a drawing (Chapter 7), the other for the synthesis of stroke patterns (Chapter 8). More importantly, in order to create meaningful representations (a simplified drawing or a synthesized pattern), we relied on the *analysis* of the input data from a low-level perceptual point of view.

However, besides their contributions, both approaches suffer from a delicate problem: Their analysis step is only semi-automatic. Indeed, they both ask the user to set an intrinsic scale ε, and possibly to separate a drawing into categories. While it seems intersting to provide semi-automatic tools for synthesis, so that the user can intuitively control the final result, analysis should ideally be fully automatic. To this end we should answer two questions: How to automatically segregate a drawing into coherent parts? What does each part represent and how can it be described in a compact way?

As we have discussed before, there seems to exist at least two categories of lines in a drawing: contours and patterns. Contours usually depict shape, and to a smaller extent, shading, and are inherently one-dimensional. Patterns, on the other hand, either depict texture or shading, and can also provide hints for the perception of shape by conveying an orientation. They are more two-dimensional and are used to fill regions with a given density of similar primitives. These observations are qualitative, and in order to perform a finer analysis, we first need to determine precisely what information is carried out by contours and patterns.

Recall that in the representation model of Willats and Durand (see Part I), marks implement picture primitives, that are mapped from scene primitives. Finding which clues are depicted by a contour or pattern is thus equivalent to traverse the model in reverse order. When the scene is completely unknown, it corresponds to the questions found in low-level human vision. In the following, we will thus present some important low-level *perceptual cues* revealed by contours and patterns.

Note however that low-level vision is only the first step of human vision, and in addition, experience and memory play an important part (we will discuss that point in Part IV). In particular, the knowledge of our environment has a determining influence on low-level perception. In expressive rendering, this knowledge corresponds to having access to 3D information in the scene, and is used, for instance, in some methods that work by analogy.

9.1 Perceptual cues revealed by contours

Previous work in human vision addressed partially the perception of line drawings by a series of experiments and mathematical analysis. In particular, Koenderink studied "the shape of smooth objects and the way contours end" [Koenderink 82] and showed that some drawings actually do not correspond to a possible shape because of some stylistic variations at the end of their contours. However, he also demonstrated via experiments how line drawings are efficient in depicting shape [Koenderink 96, Koenderink 01]. Some authors studied how surface could be inferred from contours via a mathematical analysis of *intrinsic geometric properties* [Barrow 81, Koenderink 84, Beusmans 87]. In other words, they concentrated on view-independent surface features, which goes beyond the scope of low-level perception, and belongs more to an internal representation of shape.

In contrast, recent studies by Adelson et al. on material perception [Fleming 03] revealed that the human visual system is very efficient in reconstructing surface features from material properties like specular reflections [Fleming 04]. The authors compare their approach to shape-from-texture and shape-from-shading models. They relate them by observing that they allow to extract progressive approximations of surface geometry in the direction of the point of view: Surface slant (the surface orientation in the viewing direction) can be extracted from local distortions of texture patterns; While surface radial curvature (the surface curvature in the viewing direction) can be extracted from local specular reflection distortions. Therefore, there is evidence that the human visual system is able to infer surface from *extrinsic geometric properties*, i.e. without the need to build an internal shape representation, which corresponds better to the processings attributed to low-level vision.

We conjecture that a main reason why line drawings are so efficient to depict shape is because they represent locii of objects where view-dependent surface features exhibit a discontinuity in image space. Or to say it in a different way, a thin contour line will appear in a drawing only if a sharp discontinuity is present in the view-dependent approximation of the depicted surface. If this appears to be true, then we will be able to state that contours reveal discontinuity cues corresponding to depth, slant and radial curvature (i.e., progressive extrinsic surface approximations). Note that the suggestive contours of DeCarlo et al. [DeCarlo 03] indeed use one of these cues (radial curvature); However, the resulting line drawings seem satisfying only for some point of views, thus a deeper study of depicted surface features is of great interest.

Although each line of a drawing might depict local surface features, their combination will lead to a global percept of shape, in accordance to the above-mentioned studies in intrinsic shape properties. Of particular importance are line junctions, and at a more global level, the dependence on context (i.e. the influence of nearby distinct lines). See the work by Barrow and Tenenbaum for a deeper treatment [Barrow 81].

Moreover, as discussed for instance by Solso [Solso 94], something must differentiate an object from its context, and a commonly perceived demarcation is achieved through figure-ground separation, i.e. which region of a picture is in the foreground, and which is in the background. He argues that, since contours are often used to demarcate an object from its background, contours are more part of the figure than they are to the ground. Thus they help to clarify depth ordering in the scene.

Finally, not all the lines of a drawing are intended to give a perceptual cue of some-

thing present in the scene. For instance, McCloud, in his book on comics [McCloud 94], evokes the power of "cartooning" in line drawings:

> "When we abstract an image through cartooning, we are not so much eliminating details as we are focusing on specific details. By stripping down an image to its essential *meaning*, an artist can amplify that meaning in a way that realistic art can't. Cartooning can thus be seen as a form of amplification through simplification."

For instance, lines can be used to depict motion, their style (wiggling, zig-zagging or noisy lines) is often employed to portray the world of senses and emotions (e.g. to represent clearly a character's surprise), while some other lines are not pictures anymore; they are more visual metaphors, or symbols (think of the little steam figures found for instance in manga and used to portray anger or frustration). Thus, the extraction of perceptual cues from contour lines might not be always justified, but our belief is that they correspond in most cases to features of a possible view.

9.2 Perceptual cues revealed by patterns

While contour lines seem to essentially convey shape cues, patterns offer a broader variety of information. First, like contours, they can reveal the shape of a surface, but in a different, quite complementary way, since they do not represent discontinuities but rather smooth surface regions. In the vision literature, the study of so-called surface markings bears many similarities with patterns properties. Surface markings are dense sets of curves that lie on a surface. By progressively testing and eliminating the constraints required for a marking to reveal the shape of an object, Stevens [Stevens 81], then Todd et al. [Todd 90] and Knill [Knill 92], refined the perceptual cues they can reveal. To summarize, the three-dimensional structure of a depicted surface is determined from the statistical distribution of lines orientations within local neighborhoods. In practice, it means that the local curvature of a surface can be revealed by the *local oriented density* of the pattern that represents it. This formulation could fit nicely in a low-level perceptual analysis of pattern density, and motivates future work in that direction.

Patterns are also often used to represent tonal variations, like the self-shadowing effect of a surface. However, a pattern usually consists in a distribution of individually perceived elements. How such a distribution can be integrated in order to yield tonal information? If we consider, again, low-level vision models, it can elegantly be explained by the use of a *multi-scale representation*. In their article on image analysis, Alvarez and Morel [Alvarez 94] give a comprehensible presentation of existing multi-scale models for human vision, as well as a general model for multi-scale analysis. The main idea is that there exists, in the first milliseconds of the perception process, a series of parallel, fast and irreversible operations that integrate local information and already yield very rich and useful information to further understanding of the image. Intuitively, the human visual system builds a visual pyramid by applying local operators (similar to smoothing) at each stage. Hence, patterns are perceived as individual elements at the finest level of the pyramid, but also as more homogeneous regions at a higher level, and these regions might efficiently carry tonal cues. For

more information on multi-scale analysis, the reader is referred to the book edited by Romeny [Romeny 94].

Finally, apart from representing some scene features, patterns also possess an internal structure that relates them to texture analysis, as we have seen in Chapter 8. We took inspiration from non-parametric texture synthesis models, but it appeared to us that they are quite limited because the definition of a neighborhood in a stroke pattern is ambiguous and pose many problems. Instead, in future work, we would like to investigate parametric models of texture, that analyze a pattern via *statistical measurements* and without relying too much on a notion of neighborhood. A successful parametric texture model has been presented by Portilla et al. [Portilla 00]: They were able to synthesize many textures with their approach, except for some stroke-based patterns. Moreover, they proposed an efficient method of evaluation based on user experiments, where each statistical measure is proved to be essential to the analysis. We thus envision to take a similar approach to the analysis problem. We hope that the use of vector input in the form of stroke-based elements instead of pixels will allow us to avoid the problems encountered in [Portilla 00], suggesting that those elements are first analyzed independently by the human visual system before being related to each other.

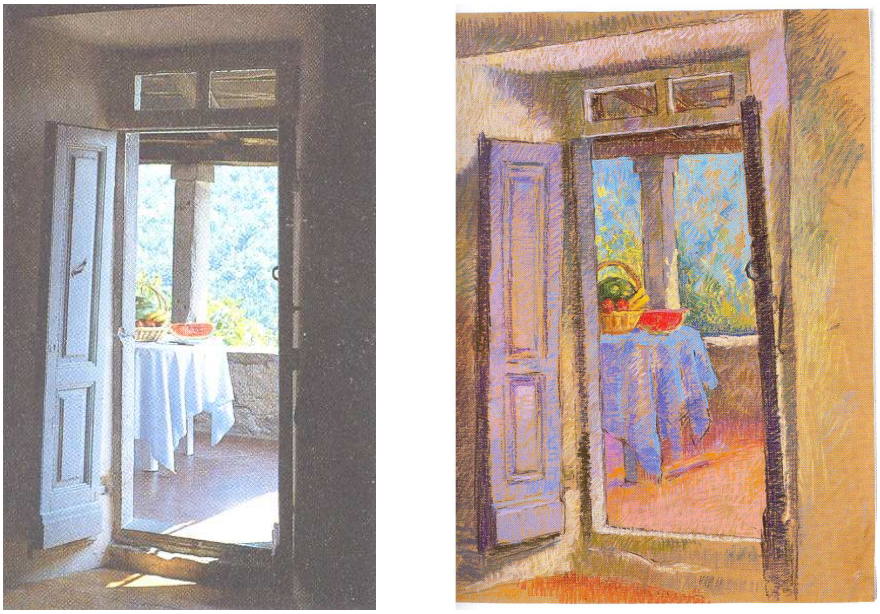
Part III

Picture color composition

Chapter 10

How are colors distributed in a picture ?

While Part II was devoted to the analysis of line drawings, either input by the user or extracted from an existing drawing, we now turn our attention to another mean of expressivity: color composition. Our goal then is to understand the process that leads to the distribution of colors in a picture, find tools to acquire such distributions (e.g. from existing paintings) and propose methods to manipulate them and apply them to other representations. As in the previous parts of this thesis, we begin by describing some common schema found in traditional techniques.



(a) (b)

Figure 10.1: A reference photograph (a) is reproduced using colors that have been altered in order to increase contrast and control temperature (b).

10.1 Examples from traditional paintings

Before starting to paint on a canvas, an artist has to choose a set of basic colors that he will use directly or in combination with other colors; It is usually called a *color*

palette. Depending on the chosen medium, the palette might contain more or less colors. For instance, with pastels, it is recommended to possess a large set of colors, because they cannot be blended. To the contrary, oil paint is usually blended in later steps of the painting, thus a minimal set of basic colors is sufficient.

Then, various possibilities are offered to the artist in order to blend the colors of the palette: A "physical" blending, onto the palette, usually made by progressively adding to a light color small quantities of a darker color; A "superposition" blending: The colors are then directly blended on the canvas to give new tones; And an "optical" blending: The juxtaposition of very small quantities of pure colors that, via an optical "illusion", will create another color (e.g., like in pointillist paintings).

(a) (b) (c)

Figure 10.2: (a) Natural colors in illustration show the real colors of a specimen. (b) Symbolic colors help to differentiate structures. (b) Design colors are used for clarity and impact.

Of course, the choice of a color palette can be based on many different motivations. For instance, the original colors of a scene can be altered in order to emphasis some illumination effects (see Figure 10.1): contrast, diffusion, cold or warm tones can be manipulated by the artist to direct the observer's attention. This way, the artist can also control the color harmony of the final representation, setting a specific mood in the composition.

Another example is given by the use of color in scientific illustration, which can be divided in three general categories [Wood 94] (see Figure 10.2): natural colors, which represent the true colors of the represented specimen; symbolic colors, which utilize assigned or familiar colors for specific parts of the subject; and design colors, which are used as an aid to recognition.

Once a color palette has been chosen, the artist is then ready to start painting. He applies colors from the palette to regions of the canvas. Those regions can overlap to create new colors absent from the palette, as mentioned above. Their shape may be arbitrarily close to the shape of depicted elements: Small elements might be ignored, close elements agglomerated together, etc. These abstraction choices either come from stylistic decisions from the artist (like ignoring some objects in the background to give an aerial perspective effect), or result in the constraints set by the medium: Simply using a brush of medium size makes it impossible to represent small details. Note, in addition, that many artists exploit the "shape" of the brush stroke in order to create

some interesting effects. Finally, details might also be added from the interaction between the chosen medium and canvas, like the slightly inhomogeneous color densities found in watercolor paintings.

10.2 A color distribution work-flow

During a real painting, the medium plays an important role through the whole process: its density governs how colors are blended, in the image as well as in the palette. However, its effects are only visible in the texture and high-frequency details present after the paint have been laid down on canvas. From a computational point of view, the effects of medium only appear at the very end of the painting process. We can therefore describe a computational color distribution work-flow *independently* of the chosen medium, which is then applied as a post-process.

First create a palette Since the description of a color palette as defined in the previous section is highly dependent on the chosen medium, we must give it a more generic definition. Therefore, in the following, we will refer to a color palette not as the physical tool used by painters, nor as a set of basic colors that can be blended on a canvas (and thus give colors that are not in the palette); We rather define it as a high-level descriptor, that characterizes the colorimetric composition of the final painting by a set of basic colors **regardless of their spatial distribution**. The reason for using a set of distinct basic colors is motivated by findings in the perception of colors, as presented, among others, by Solso [Solso 94] (here basic colors are called foveal colors):

> "Our brain access a color category in response to a limited range of wavelengths. Of course, some neighboring colors may be similarly classified, but the *best* candidates are restricted to a limited range. Such central colors are called *foveal colors* and are prototypical."

The choice of a palette and its number of prototypes (or basic colors, or foveal colors) allows the user to perform a colorimetric abstraction of the composition.

Then distribute colors In this second stage, the color prototypes of the palette *and only them* are used to represent colors from the depicted scene. At this stage, the user can perform a spatial abstraction. First, he can selectively remove some small and meaningless elements, or smooth out high frequency details. He can also simplify the borders of some regions (note that this is what happens when using a "medium-sized" brush tool) or group regions of similar color or depth for instance. All these operations should be intuitive to apply and defined in the picture plane.

An important remark is that the colorimetric and spatial abstractions performed through this work-flow, even if done in a sequential order, are dependent on each other: There is no need to have a basic color in the palette if it is not used in the final distribution (e.g. because it corresponds to a detail that has been removed); While the agglomeration of two regions might be forced by the fact that their respective colors are described by a single basic color in the palette.

Acquisition issues The main reason to define such a generic work-flow is to allow to use it indiscriminately with expressive renderings, paintings, drawings or photographs. This way, the properties can more easily be transfered from one medium to the other. However, it raises the question of acquisition, which is the main topic of this part on picture color composition, that is to say: How can we reverse this work-flow so that color palette and distribution can first be acquired from a reference picture, and then manipulated or applied to a target representation? We will only address the issues related to the extraction of a color palette in this thesis. But we will also give some ideas of how the whole work-flow might be implemented, and inverted.

10.3 Previous work

Many image processing algorithms can fit in the work-flow we described above, e.g. segmentation, quantization, denoising, equalization, filtering, etc. The list is long, and giving a complete presentation is out of the scope of this thesis. We prefer to focus on those techniques that have been used in the context of expressive rendering, or that have the potential to bring new ideas in the field, especially for the problem of acquisition. In the following, we distinguish between colorimetric and spatial methods, which correspond to each stage of the work-flow.

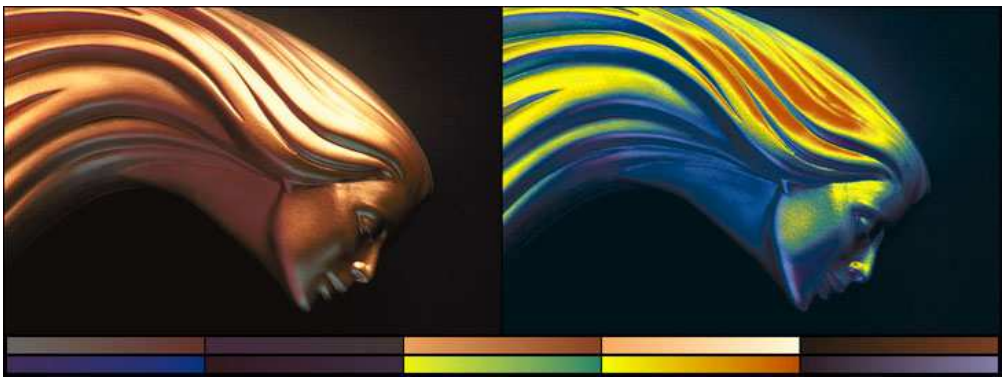


Figure 10.3: The colors of an input picture (left) are manipulated to yield a target picture (right), by matching reference and target color gradients (bottom) [Grundland 05b].

Colorimetric methods Among the many methods that allow to manipulate the colors of an image, recent work provided means to acquire some color descriptor that is then used either to manipulate the image at a more intuitive, higher-level, or to transfer the colors of a reference image to a target image by matching their respective descriptors. One interesting approach is the “color search and replace” approach by Grundland [Grundland 05b]. Their system is semi-interactive and relies on the matching of color gradients (their color descriptor). The user first picks a set of reference gradients in an input image, then he defines, for each input gradient, a target gradient, and the system warps the colors of the input image to match the target descriptors (see Figure 10.3). The target gradients can either be created by modifying the reference ones, or by picking gradients in another image, in practice performing a color transfer.

However, as already noticed in Chapter 9, while user interactions are often desirable during a synthesis process (here the pairing of gradients), any analysis should ideally be fully automatic. There is thus a need for a more systematic color descriptor acquisition. Previous work have considered mainly two methods to deal with this problem: Statistical and category-based approaches.

(a) (b) (c)

Figure 10.4: A reference photograph (a) is modified using the colors of a target image (b), yielding the image in (c) [F. Pitié 05].

Statistical methods extract probability distributions in a chosen color-space and then allow to match two distributions to perform a manipulation like color transfer. For instance, the method of Reinhard et al. [Reinhard 01] uses color distributions independently on each axis of the L$\alpha\beta$ color space as color descriptors. This color space has the advantage of matching the properties of the retina, with logarithmic axes. Unfortunately the method gives satisfying results only for outdoor, natural images, which is an inconvenient if one wants to use the method in an expressive rendering context. Other methods perform more complex statistical analysis [Grundland 05a, F. Pitié 05] and obtain better results (see Figure 10.4). However, the statistical representation is not easy to manipulate, compared to a small set of descriptors as in color search and replace; And the matching process is entirely automatic, thus there is no simple way for the user to control it (Grundland et al. [Grundland 05a] propose a method to manually enhance some colors, but it gives limited control). In practice, user interaction is an important feature though, as for example the user might want to match very different descriptors, as in Figure 10.3.

(a) (b) (c)

Figure 10.5: A reference photograph (a) is modified using the colors of a target painting (b), yielding the image in (c) [Chang 03].

Category-based methods, on the other hand, rely on the property of the human visual system to focus on so-called foveal colors. The work of Chang et al. [Chang 03]

used in particular eleven basic color categories (the perceptual theory behind those categories is presented in Chapter 11) and built convex hulls in Lab that enclose the colors belonging to each category in an image. Thus, their color descriptors are the convex hulls themselves, and they use interpolation methods inside each hull to match two descriptors. This method gives interesting results (see Figure 10.5). However, it might give incoherent results for colors at the boundary of two categories. An important feature of this approach that has not been investigated by its authors is to manipulate the descriptors by hand, or to create matchings between different categories. Note that the category-based method has been adapted to the processing of image sequences by Wang et al. [Wang 04a].

Spatial methods A lot more work has been done on spatial methods, especially in expressive rendering, some of them have already been presented in Chapter 1. We now briefly describe the image processing algorithms they use in order to give an inspiring list of possible spatial abstractions.

A common technique used to simplify the distribution of colors in an image is segmentation. Two algorithms have proved efficiency in expressive rendering applications: The mean-shift algorithm [Comaniciu 99] and its anisotropic extension [Wang 04b] in the work of Wang et al. [Wang 04c]; And the normalized cuts algorithm [Shi 00], used by Kolliopoulos et al. [Kolliopoulos 06]. Both methods handle multi-valued pixels (e.g., embedding colors, but also depth for instance).

Another technique is anisotropic diffusion [Perona 90, Black 98], which has been used recently in the work of Winnemöller et al. [Winnemöller 06]. They use a technique called bilateral filtering [Tomasi 98], which can be considered as a fast approximation of anisotropic diffusion methods [Elad 02]. In particular, both approaches have the advantage of smoothing an image while keeping its discontinuities, hence removing the inconveniences of a classic Gaussian blur.

Finally, mathematical morphology has been recently incorporated in expressive rendering approaches, like the watercolor renderings of Bousseau et al. [Bousseau 06] and the image abstractions of Bangham et al. [Bangham 03], that use filters by reconstruction. For a pedagogical survey on morphological techniques, the reader is referred to the survey by Kopen et al. [Köppen. 00].

In the context of the work-flow we defined in the previous section, all those algorithms have interesting qualities. However, it is not straightforward how they can be used as models for the *acquisition* of the spatial abstraction of an image. Moreover, we must also devise a way to incorporate both spatial and colorimetric abstraction in the same workflow, and it is not clear for now how it could be done.

10.4 Contributions

In this thesis, we mainly concentrated on the acquisition of a color palette from an image, which is part of the first stage of our work-flow. This is a complex problem, since it involves the perception of colors, which is not well understood. That might explain why little work has been done in the direction of category based color transfer.

We present in Chapter 11 a new color palette extraction technique that we plan to use with color transfer and spatial simplification techniques inspired from previous work. Moreover, we analyse the color perception issues involved in this process and propose solutions as future work.

Chapter 11

Color palette extraction

Figure 11.1: Left: a fine painting from "The Castle in the Sky" ©. Right: its associated palette extracted with a user-defined tolerance.

Our main motivation for this work is to describe an input image by a set of color prototypes: a color palette (see Figure 11.1). As discussed previously, many applications in graphics, especially in expressive rendering, work by manipulating the color distribution of an input image or video. Using an accurately extracted color palette as an intermediate tool for color processing would allow a higher-level control of color composition. Our second interest is to study the efficiency of commonly used color spaces and color difference measures. This way, we gain insights into the color-related problems raised by a supra-threshold process like the extraction of a color palette, i.e., a process using differences above a near-noticeable threshold.

Understanding color is a complex problem, which is reflected by the numerous available color spaces: RGB, CMYB, HSV, Lab, Luv, L$\alpha\beta$, etc. They rely on different color models: additive colors, subtractive colors, opponent colors, etc. The reason for this diversity is that they are often used in specific applications. For instance, RGB is used for television and computer displays, and thus most of the digital image formats describe a pixel using an RGB triplet. RGB colors are additive because the contribution of each color emitted from a screen (Red, Green and Blue) is combined optically by the human visual system of an observer located at a sufficiently far distance. CMYB, on the other hand, is tailored to printing, and thus colors are subtractive: the superposition of a layer of Cyan, Magenta, Yellow or Black on a white

paper alters its reflective properties. HSV is usually found in graphics softwares because it intuitively matches the manipulations artists are used to do: by controlling the Hue, he or she can select a base color, that is then desaturated and darkened using Saturation and Value coefficients.

Those color spaces are thus mainly focused on a specific body of applications. On the other hand, color spaces like Lab, Luv and L$\alpha\beta$ have been designed to reflect more or less accurately our perception of colors. They are based on an opponent color model, which have been observed physiologically in low-level vision: yellow-blue and red-green are mutually exclusive, while achromatic color is determined by relative activity of the black and white channels with the provision that black is solely a contrast color. Lab is nearly always used when an application needs to compute "perceptually uniform" color differences. However, it is only tailored to small color differences, and medium and big differences encountered in supra-threshold computations are badly evaluated using this color space. We will nonetheless use Lab for our color palette extraction, because there is no better choice; But we will also test its effectiveness, along with alternative color difference formula.

11.1 Definitions

The goal of our study is to find a color descriptor that we call a palette and that is able to represent faithfully the distribution of colors in a picture up to a user-defined tolerance parameter.

In this work, we restrict ourselves to a simple model of color distribution in a picture: we consider the input picture to be composed of a set of contiguous regions, each one exhibiting low color variations. This formulation is well adapted to many drawings and paintings where those regions correspond to brush strokes, wash tints or pen and ink lines; We also found that it worked well with many photographs, where regions might correspond to diffuse surfaces. Moreover, due to noise in the capture process (e.g. with a camera or scanner), we consider only a subset of those picture regions, that we call regions of interest.

Definition. *A region of interest R_i is a connected set of pixels characterised by:*

- *a minimum, user-specified width σ_d:*

$$\exists D_{\sigma_d} \subset R_i$$

  where D_{σ_d} is a disc of diameter σ_d.

- *no visible color discontinuity, w.r.t a color threshold σ_r:*

$$\forall p \in R_i,\ \nexists q \in N_p \ s.t.\ |col(q) - col(p)| > \sigma_r$$

  where N_p is the $8-$neighborhood of pixel p

- *no overlapping:*

$$\forall i \neq j, R_i \cap R_j = \emptyset$$

The value of σ_d thus corresponds to the smallest width of a significant region in the picture: small isolated clusters of pixels and thin stripes in-between region boundaries are not considered as regions of interest and will be ignored during the palette extraction.

Having defined the set of regions of interest in a picture, we then want to extract the minimum number of color prototypes necessary to describe them w.r.t a user-defined tolerance parameter. As the tolerance value increases, each prototype represents a broader range of colors and the number of prototypes thus decreases. Therefore, instead of using single colors as prototypes, we use clusters in the chosen color space in order to represent the range of colors carried under a given prototype.

Definition. *A color palette C is a set of color clusters called prototypes in a chosen color space that describes the color distribution of regions of interest in an image up to a tolerance parameter t and we note $C \overset{t}{\sim} \{R_i\}$.*

Unfortunately, the meaning of the perceptual color similarity $\overset{t}{\sim}$ is not straightforward, because it depends on the chosen color space and color difference formula employed. As we've seen previously, color perception is a delicate problem, and we discuss possible alternatives in the next Section.

11.2 Color difference formula

The perception of colors is such a complex mechanism that the definition of a color difference is ambiguous as soon as it operates at a high level. A good example is the turquoise color, where different observers usually do not agree on whether it is a blue or a green color. However, for some colors, such as deep red for example, there is not such an ambiguity. Although each individual has a personal interpretation of color, there is also a consensus, even if it seems to be only for certain "pure colors". In our goal to create a color descriptor, we should integrate this consensus among observers. To answer partly this demand, we decide to represent colors in the Lab color space.

11.2.1 The Lab color space

This color space has been created by the CIE (Comission Internationale de l'Eclairage in French) in 1976 in order to fulfill the needs of a perceptually uniform color space. However, it has been designed to fit only small color difference observations, and thus only allows to model a low-level consensus among observers' color perception.

The axis of the Lab color space correspond to a luminance (L), a red-green (a) and a blue-yellow (b) axis, that are in accordance with the opponent model of color vision. The luminance axis is also called the achromatic axis, and the distance between a color and the achromatic axis corresponds to chroma. The hue is related to the angle around the achromatic axis (see Figure 11.2).

To evaluate the difference between two colors c_1 and c_2 in Lab, the euclidean distance given below is used, taking advantage of its locally, perceptually uniform property.

$$\Delta E_{Lab} = \sqrt{\Delta L^2_{Lab} + \Delta a^2_{Lab} + \Delta b^2_{Lab}}$$

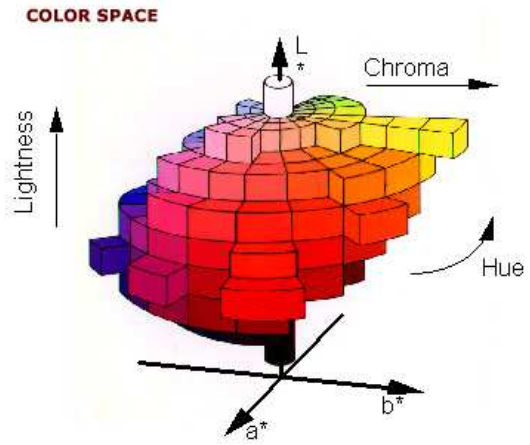


Figure 11.2: The Lab color space

where $\Delta L_{Lab} = L_{c_1} - L_{c_2}$, $\Delta a_{Lab} = a_{c_1} - a_{c_2}$ and $\Delta b_{Lab} = b_{c_1} - b_{c_2}$.

Since it is the most widely used color space as soon as it comes to perceptual color uniformity, we use it for our study. However, we propose alternate color differences in the following section and we will compare the effectiveness of each of them in the context of our supra-threshold application.

11.2.2 Alternatives to the Lab Color Difference

It has been reported [Luo 00] that using a euclidean distance in Lab to compute color differences results in small perceptual imprecisions, especially in the blue region. Recently, the CIE proposed a more accurate color difference formula working in Lab, called CIEDE2000 [Luo 00], fit from an experiment on an vast number of observers in different color scnenarii. It not only addresses the blue region imprecision, but also some other subtle problems of Lab, for instance along the achromatic axis. The general CIEDE2000 formula is written below.

$$\Delta E_{00} = \sqrt{(\frac{\Delta L'}{k_L S_L})^2 + (\frac{\Delta C'}{k_C S_C})^2 + (\frac{\Delta H'}{k_H S_H})^2 + R_T (\frac{\Delta C'}{k_C S_C})(\frac{\Delta H'}{k_H S_H})}$$

The detail for each of the variables can be found in [Luo 00].

While it accounts for all the observed imprecisions of Lab, the formula is quite complex, and it is not a distance. Another, simpler formula, called DIN99, has been proposed in [Cui 01]. When compared to CIEDE2000, DIN99 has the advantage of being a deformation of Lab: equations used to obtain L, a and b coefficients have been modified to solve Lab imprecisions. DIN99 is thus a real color space, and a color difference is computed using an euclidean distance.

$$\Delta E_{99} = \frac{1}{k_e}\sqrt{\Delta L_{99}^2 + \Delta a_{99}^2 + \Delta b_{99}^2}$$

The detail for each of the variables can be found in [Cui 01].

Note that there is no need to explicitly convert a color to DIN99 in order to compute a color difference: since it is a deformation of Lab, the conversion is done implicitly during the difference computation. The DIN99 formula performs less accurately than

the CIEDE2000 formula, but it is still far better than the Lab formula, and has the advantage of being a distance computation.

However, even if those formula are supposed to improve Lab for small color differences, they do not help for supra-threshold ones, as with any euclidean distance in any color space [S.M. 95].

11.2.3 Color similarity through categorisation

Recall the ambiguity of the turquoise color: depending on the observer, it will be categorised as a blue or a green. It appears that we have a natural tendency to classify colors using a restricted number of categories, which can be perceived as high-level descriptions. There is here a deep link with linguistics, because of the use of specific terms to describe those categories. Berlin and Kay [Berlin 69] studied this natural tendency towards categorisation. They performed naming experiments in 98 different languages and examined the consensus between different cultures. These experiments were done using color chips coming from the Munsell book of colors, so that they could analyze the results in the perceptually uniform Munsell color space (similar to Lab). They found that in languages coming from so-called "developed countries", one can distinguish, at maximum, eleven basic color terms: black, white, grey, red, green, blue, orange, yellow, brown, purple and pink. Moreover, they found that there were prototypical colors for which a full consensus was reached among observers, while the consensus decreased away from the prototypes and became unreliable near category boundaries. Figure 11.3 illustrates those categories, along with their prototypes, plotted on the outer boundary of Munsell's color space. Note in particular the lack of consensus in the gap in-between the blue and green regions that explain how a color such as turquoise is hard to categorize.

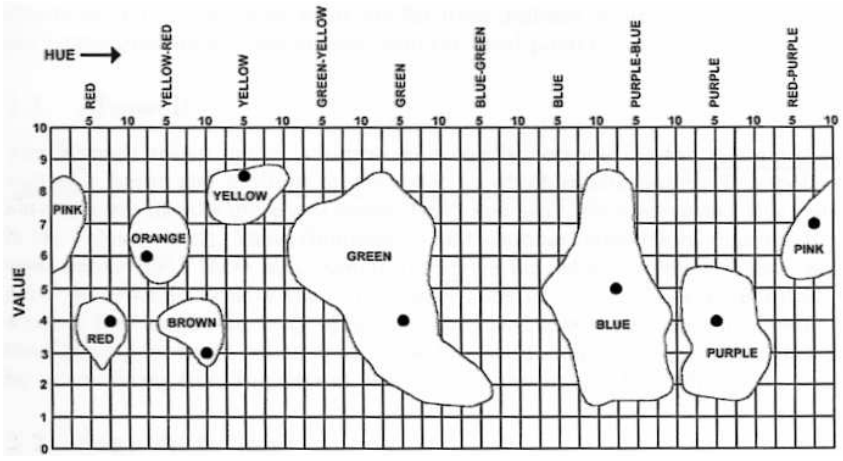


Figure 11.3: The discrete consensus regions corresponding to each one of the eight chromatic basic color terms plotted on the outer envelope of Munsell's color space (black, white and grey are not plotted here).

To our knowledge, very few color applications made use of this experiment, with the notable exception of the color transfer applications of Chang et al. [Chang 03]. Unfortunately, they relied on categories with sharp boundaries in their work, even if Berlin and Kay's experiment reported their poor reliability. Nonetheless, they obtained interesting results that motivate further research.

Another work who did use color categories is the color image database retrieval algorithm from Seaborn et al. [Seaborn 99]. But instead of using the discrete categories of Berlin and Kay, they performed additional experiments in order to get a fuzzy categorization, where this time each color is given a distribution of membership values in each of the eleven basic color categories. This fuzzy categorization is illustrated in Figure 11.4 in Munsell's color space again, where the grey level represents the membership in the closest category.

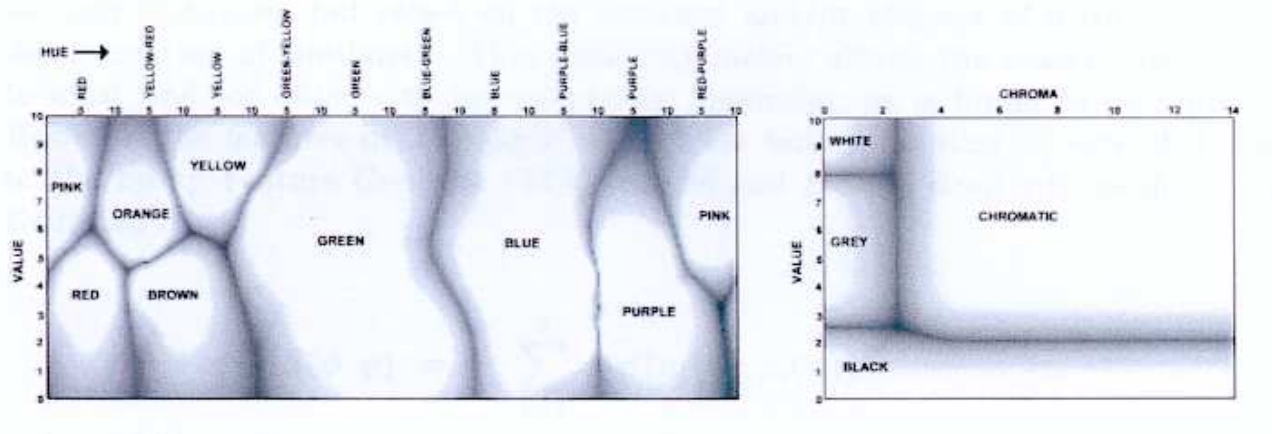


Figure 11.4: The fuzzy color categorization for chromatic and achromatic colors.

Intuitively, each membership value represents the probability to belong to the corresponding category for an arbitrary observer. Once this fuzzy categorization is built, Seaborn et al. define a similarity measure $S \in [0,1]$ between two colors c_1 and c_2, based on Fuzzy C-means, where the similarity is computed as the intersection of the fuzzy sets of each category:

$$S(c_1,c_2) = \sum_{i=1}^{11} \min\{\mu_i(c_1), \mu_i(c_2)\}$$

where $\mu_i(c)$ is the membership value of color c in the i-th category.

When the similarity between two colors is equal to 1, they use the euclidean distance in the Munsell space (similar to the euclidean distance in Lab) to distinguish them. Even if it seems to be sufficient for their image database retrieval application, the question of how high-level category similarity and low-level color difference can be combined is still unanswered.

11.3 Our approach

For the purpose of our color palette extraction, we organise our approach in four stages:

1. Identify candidate regions of interest via an image space segmentation;
2. Keep only valid regions of interest by removing candidate regions that do not fulfill the minimum size requirement;
3. Group regions of interest of similar color via a color space segmentation;

4. Extract color prototypes by means of a principal component analysis (PCA).

11.3.1 Stage 1: Image segmentation

The first stage consists in a segmentation of the input image in a set of regions exhibiting no colour discontinuity. To this end, we use the *Mean Shift Segmentation* algorithm that takes as input two parameters: a domain threshold σ_d that corresponds to a characteristic size in image space, and a range threshold σ_r that corresponds to a characteristic size in color space.

The algorithm then works in a five dimensional space (2 dimensions for the picture and 3 dimensions for color): For each pixel in the image, it finds neighbors in both picture and color based on σ_d and σ_r; It computes a vector (the mean shift) from the current 5D location and the center of gravity of the selected neighbors and moves the pixel along it; It then continues this way iteratively until convergence, i.e. when the mean shift is close to a null vector.

This algorithm is guaranteed to converge to the local mode of the 5D density function underlying the picture, as illustrated in Figure 11.5 for a 2D case. In order to perform the segmentation, all the pixels that converged to a same mode are clustered into a single region (the red spheres in Figure 11.5).

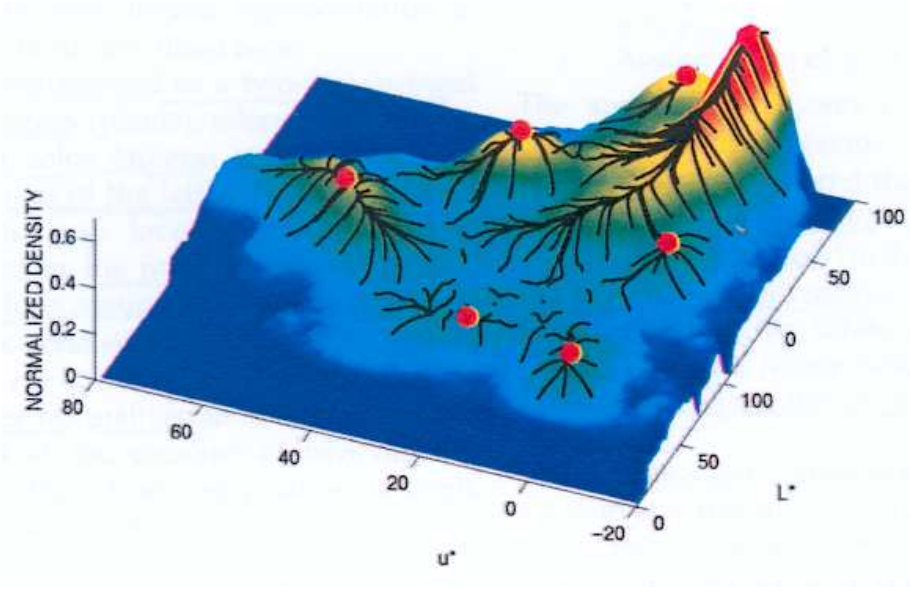


Figure 11.5: Illustration of the mean-shift algorithm in 2D: each pixel converges to a local mode of the density function.

In our application, we set σ_d to the desired minimum size of a valid region of interest, with typical values between 1 and 5 pixels. We set $\sigma_r = 3$ which is known to be the near-noticeable distance using the euclidean metric in the Lab color space. We show the result of a segmentation in Figure 11.6.

11.3.2 Stage 2: De-noising

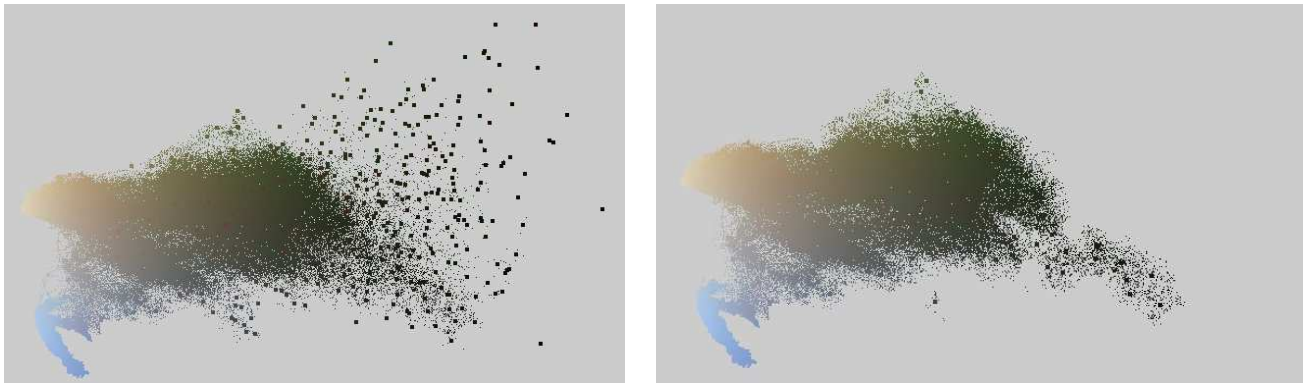
The mean shift segmentation gives us candidate regions of interest in the picture, along with the mode of each region. We now apply a denoising operation where regions that are "thinner" than the user defined minimum size, i.e. σ_d, are removed.

(a) (b)

Figure 11.6: (a) Source image (b) Each pixel is given the color of the mode it converged to during the mean shift segmentation.

To this end, we directly apply the definition of a region of interest: For each region of the segmentation, it is considered a valid region of interest if a disc of diameter σ_d can be completely contained in the region.

The effect on the segmentation is better visualized in the Lab color space, as shown in Figure 11.7, where isolated regions due to noise are removed by the process. In those figures, points correspond to pixels of the input picture and squares to modes of the segmentation; Notice how modes having a small number of pixels attached to them tend to be removed by the denoising process.



(a) (b)

Figure 11.7: (a) Lab distribution after segmentation (b) After denoising.

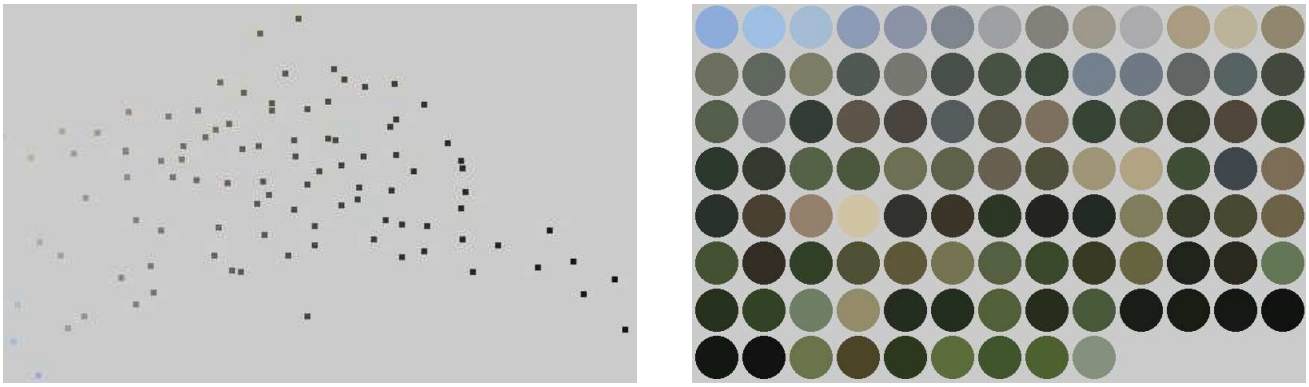
11.3.3 Stage 3: Color segmentation

Stages 1 and 2 together allow us to retrieve the regions of interest in the input picture. But they also give us a set of modes that represent prototypical colors of the regions. Unfortunately, modes color cannot be taken directly as prototypes of our palette for an intuitive reason: two regions might be located away from each other in the picture and

their colors be very similar, so that taking their modes color will result in prototype redundancy.

We thus have to cluster modes colors regardless of their position; this is equivalent to consider that regions at various locations in the picture, but with a similar color, will be represented by a single prototype in the final palette. To perform clustering, we again apply the mean-shift algorithm, this time on the modes of regions of interest, and only in Lab in order to ignore regions location.

For this color-space mean-shift, a single threshold has to be chosen, which correspond to the user-specified tolerance parameter t that will control the number of prototypes in the palette: Indeed, the bigger the tolerance is, the more clustering is performed, yielding a small number of prototypes, and thus reducing color complexity. We show in Figure 11.8 the result of the mean-shift in Lab with $t = 3$ and the corresponding colors of the palette.



(a) (b)

Figure 11.8: (a) The modes obtained with a mean-shift in Lab color space with $t = 3$ (b) Colors corresponding to each mode.

11.3.4 Stage 4: PCA

In order to better analyze our results, the range of colors represented by a single prototype should be visualized intuitively. To this end, we first perform a principal component analysis on the set of points associated to each prototype. The resulting collection of Gaussian distributions, shown in Figure 11.9(a), is what we call a palette, and we will also refer to a prototype as one of the Gaussian distributions from now on.

Then, in order to visualize each prototype, we take its two most elongated axis and scale them to fit a disc, as shown in Figure 11.9(b). This way, it is easier to grasp the range of colors represented by a given prototype.

11.4 Results

We show now palette extraction results using four categories of images: color gradients, paintings & drawings, computer-generated images and photographs. We used for these images the DIN99b formula for color differences because, as discussed in Section 11.5, it gave better palette extractions on average, especially for bluish colors.

(a) (b)

Figure 11.9: (a) The principal component analysis for each prototype (b) A more intuitive visualization of the palette, using prototypes elongation.

We set the tolerance parameter to low values, typically $t \in [3,5]$, because we found that increasing tolerance generated prototypes with too broad color ranges. Thus, using tolerance values $t > 5$ will perceptually merge different colors into a single prototype; while the small tolerance values we use in the following examples are often too low to be able to merge similar colors into a single prototype, leading to redundant prototypes (especially visible with dark colors). It is an important limitation of our approach, and we discuss avenues for ameliorations in Section 11.5.

Figures 11.10 and 11.11 show results using a blue-to-green gradient, and a more complex "metallic" gradient. Note how the extracted prototypes correspond to regions of various width in the gradient, since the palette is only a color descriptor. With the more complex gradient of Figure 11.11, we get a more complex color palette, as expected. Setting tolerance to $t = 5$ however is not sufficient to merge similar colors into a single prototype, which leads to a set of redundant prototypes in the dark tones for instance.

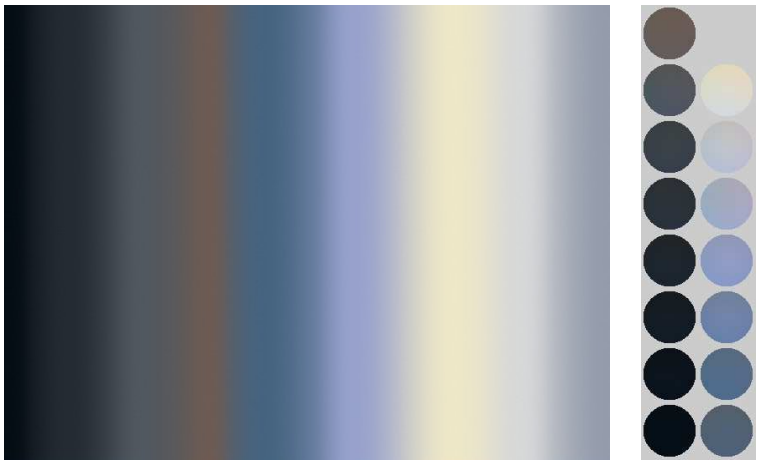
Figures 11.1, 11.12, 11.13 and 11.14 show results on paintings & drawings. A single glance at the palettes allows to recognize the "color ambiance": a use of pastel colors in Figure 11.13; darker, more profound colors in Figure 11.14. Here, the complexity of the picture color composition, as well as the use of a lower tolerance, result in greater numbers of prototypes. Again, this has the disadvantage of creating redundent prototypes, in dark or desaturated colors especially.

Figures 11.15, 11.16, and 11.17 show results from computer-generated images. We intentionally selected images with complex gradients of colors that, together with a low tolerance value, result in a great number of prototypes for Figures 11.15 and 11.16. The later is a particular bad case for our current approach, which creates an important number of dark tones in the palette. However, the monochromatic nature of Figure 11.17 gives a very low number of prototypes: indeed, it is very similar to a gradient image in its color composition.

Finally, Figures 11.18, 11.19 show results from photographs. While Figure 11.18 contains mainly whites, browns and blues, Figure 11.19 is much more complex in its color composition, which results in a very high number of prototypes.

(a) (b)

Figure 11.10: (a) The input blue-to-green gradient image. (b) Its palette extracted with $t = 5$.



(a) (b)

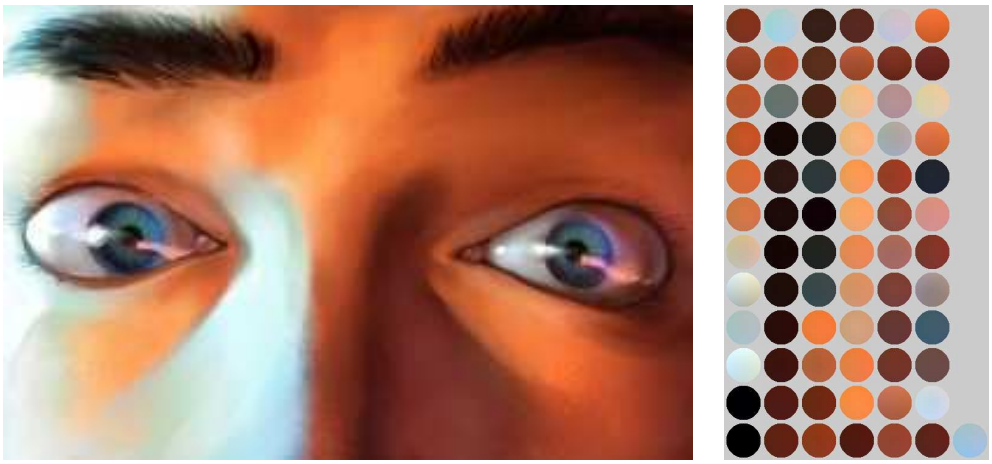
Figure 11.11: (a) The input “metallic” gradient image. (b) Its palette extracted with $t = 5$.

(a) (b)

Figure 11.12: (a) The input image. (b) Its palette extracted with $t = 3$.

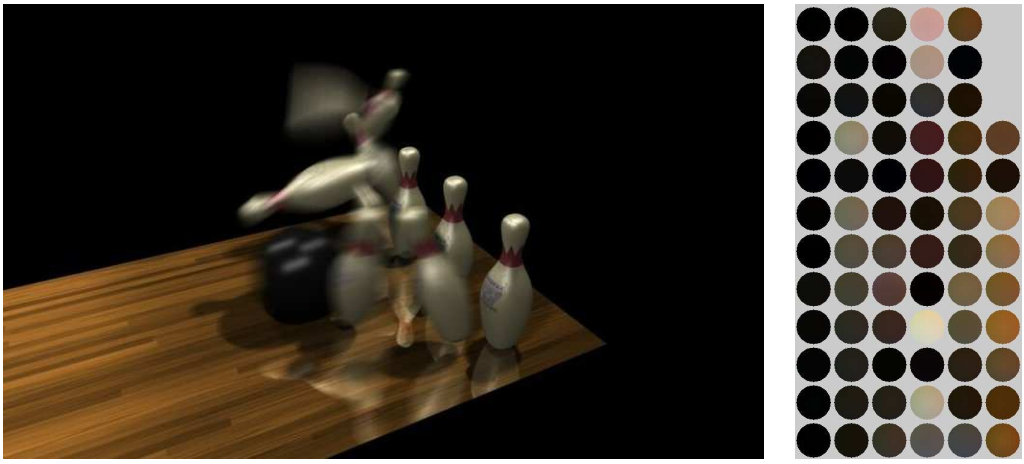
(a) (b)

Figure 11.13: (a) The input image. (b) Its palette extracted with $t = 3$.



(a) (b)

Figure 11.14: (a) The input image. (b) Its palette extracted with $t = 3$.



(a) (b)

Figure 11.15: (a) The input image. (b) Its palette extracted with $t = 3$.

(a) (b)

Figure 11.16: (a) The input image. (b) Its palette extracted with $t = 3$.

(a) (b)

Figure 11.17: (a) The input image. (b) Its palette extracted with $t = 3$.

(a) (b)

Figure 11.18: (a) The input image. (b) Its palette extracted with $t = 3$.

(a) (b)

Figure 11.19: (a) The input image. (b) Its palette extracted with $t = 3$.

11.5 Discussion

On tolerance control vs number of prototypes

We think that using a tolerance parameter to control the color complexity of a palette is the most intuitive control: it adapts the complexity of the palette to the complexity of the picture. Take for instance a tricolored flag; Since it is only composed of three colors, the color complexity is already so low that varying tolerance will not change the result.

However, one could prefer to ask for a desired number of prototypes in the palette. A way to do that would be to run iteratively stage 3 of our algorithm with increasing tolerance values until the number of prototypes is close to the expected number. However, like with the flag example, this expected number might never be reached; And asking for a same number of prototypes on different images will yield different amounts of abstraction.

On Lab, DIN99 and CIEDE2000 color difference measures.

We implemented the DIN99 formula [Cui 01] and used it throughout our algorithm. We now compare it to the euclidean distance in Lab. We plan to do similar comparisons with the CIEDE2000 formula [Luo 00] in a near future.

In Figure 11.20, we show a close-up of the desert image segmented using the colors of its palette computed, on the left with a Lab euclidean distance, on the right with the DIN99 formula. Using the later segments better the sky into two different blue colors, and detects grey colors more finely when compared to Lab that mistakes desaturated greens and beiges with greys (e.g. on the bluish distant mountain).

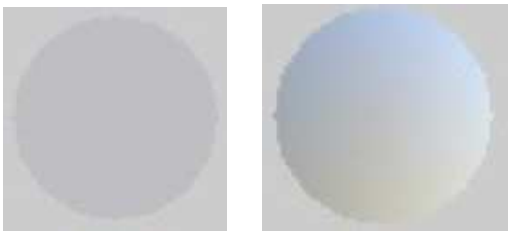
(a) (b)

Figure 11.20: (a) Segmentation using Lab color difference formula. (b) Segmentation using DIN99 color difference formula.

On the need of color categories

Analyzing the palettes we extracted in the last Section, we make two observations: First, some colors seem to be redundant, especially in the dark and desaturated tones

(brown colors are particularly problematic); Second, some prototypes hold different colors, even with the small tolerance values used in our examples (See Figure 11.21).



(a) (b)

Figure 11.21: (a) A prototype's center color, compared to (b) its full color range, showing that unwanted color variations can be found even using a small tolerance value.

Concerning color redundancy, one explanation would be that dark or desaturated colors are perceived differently depending on the surrounding color. We chose to use a neutral grey as the background color for our palette, but this could have a non-negligible impact on the perception of dark; desaturated colors that might be hard to differentiate in the palette, but would be easily identified in the picture because of a different surrounding color. The spatial aspects of color perception are discussed further in the next Chapter.

Another explanation would be that some perceptual variations depending on saturation and lightness are not taken into account in any color difference formula. Indeed, those formula are tailored to perceptually uniform, but small color differences. In particular, it seems that the colors that are hard to differentiate lie in-between high-level color categories: dark colors are a blend of a color with black, desaturated colors are a blend of a color with grey; and brown is hard to perceive in general. There is thus a possibility that color categories play an important role in suppressing this redundancy.

Concerning color variations inside a single prototype, it seems, again, that categorization plays a major role. When looking at Figure 11.21-right, our first observation is that the prototype should not contain shades of blue, nor yellow. In fact, our confusion is that we can **name** different color categories inside a single prototype. Basic color category similarity measures like the one of Seaborn et al. [Seaborn 99] might thus be crucial to constrain the clustering of prototypes, in order to ensure that a "single" category is represented. This will be even more important with bigger tolerance values, that did not give satisfying results in the present study because of the categorization problem.

On the evaluation of results

Apart from the purely suggestive observations that we made above, there is no simple method to evaluate the quality of a color palette. Our method compares favorably to previous work, but the goal of previous approaches is not really the extraction of a color palette.

Actually, even with very naive color palette extractions using, for example, quantization, only visual examination can perform the comparison, since the mechanisms underlying color perception are not well understood. Therefore, a user study might be the only way to compare two palette extraction methods.

On the use of a palette in color image processing

Once a palette is "properly" extracted, it only gives a description of the input image. In order to use it to process color images, we first need to **associate** each prototype to a region of the picture. Then we need to provide **color editing** tools, like manual edition or color transfer techniques, to change the input palette into a new target one that have its own harmony. Finally, color layers have to be composited, with an optional **spatial simplification** step in order for instance to mimic the use of a brush.

For the purpose of analyzing the extracted palettes, we computed a PCA for each prototype. The resulting set of Gaussian distributions might be used for all the processes described above. Here, we just give some ideas that we had during the course of this study and that we keep for further investigations.

Association might be done by segmentation, by giving each pixel the color of its closest prototype. But alternative strategies are possible, like building a fuzzy categorization: in this case, each pixel has a (possibly zero) membership in each prototype, depending on the distance to its Gaussian distribution. This way, we keep a color layer per prototype in the picture that we will composite later on, using for example the technique of Grundland et al. [Grundland 05b].

Color editing is also made simpler with a Gaussian distribution, since its center can be translated to change the focal color, its axes rotated and scaled to vary saturation or lightness, etc. For color transfer, it opens the question of how to map a source palette to a reference one, i.e. mapping Gaussian distributions. Again, the work by Grundland et al. [Grundland 05b] gives interesting ideas.

Finally, once color composition has been extracted by means of a palette, modified through editing and mapped back to regions in the picture, one might also want to control the spatial configuration of those regions in the picture before compositing them to give the final image. We plan to build onto previous work, that used a combination of scale-space filtering and morphological operators [Bangham 03].

Chapter 12

Remarks on color categorisation

The previous chapter explored a practical approach to perform the first step of our color composition work-flow, namely the extraction of a color palette. However, it raised some perception-related issues since color perception is not yet well understood. Our observations oriented us towards the importance of the categorical perception of colors, and we discuss here the properties of the previously mentioned basic color terms (see Section 12.1).

However, categories are very high-level descriptors of color, and the human visual system exhibits other important lower-level behaviors that should be taken into account prior to any categorization. We discuss these in turn in Section 12.2.

12.1 Color categorization and linguistics

Color categories detailed Let us first more formally define the basic color terms (BCTs). As specified by Hardin and Maffi [Hardin 00]:

> "Basic terms must be those which are general and salient. A term is general if it applies to diverse classes of objects and its meaning is not subsumable under the meaning of another term. A term is salient if it is readily elicitable, occurs in the idiolects of many speakers and is used consistently by individuals and with a high degree of consensus among individuals."

In their experiments, Berlin & Kay [Berlin 69] found eleven basic color terms that correspond to categories in the Munsell color space. The category focii reliably tend to cluster in relatively narrow regions of the Munsell space, whereas category boundaries are drawn unreliably, with low consistency and consensus for any language.

Moreover, the BCTs can be organized into progressive divisions of the color space, yielding three types of basic categories that have strong connexions with Hering's opponent process theory of color: composite, fundamental and derived. Lightness and darkness are the most salient visual experiences and thus correspond to the composite stage (black, grey and white colors). Unique hues correspond to the fundamental step (red, green, yellow and blue colors). And binary (or blended) hues correspond to the final derived step (purple, pink, orange and brown colors). This decomposition seems to have a direct influence on how BCTs are linked to each other (Figure 12.1). We speak of two colors as being linked or unlinked, depending upon whether or not a bridge color is necessary to move between them.

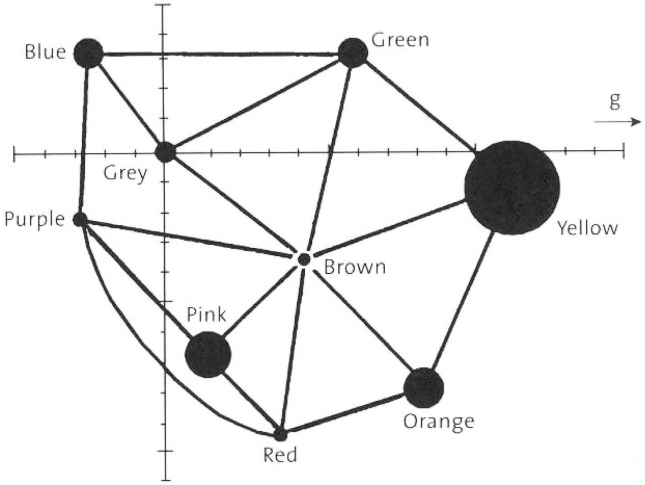


Figure 12.1: Links between basic color terms. Dot sizes correspond to the mean luminance of the category.

Some aspects of the categorization are still ambiguous though. For instance, there is room for a twelfth BCT, “peach”, that has been named in experiments with varying terms (like “beige”, or “tan”). It also seems to be the most difficult color to name, maybe because of this linguistic ambiguity. Also, brown, as well as black, are very specific colors, since they require a lighter surround for their very existence. Notice that a categorization based on color naming is not necessarily the best one, since the role of language related to perception is not clear. We discuss this point in the following.

Connexions between BCTs and linguistics Categorizing a color without taking its context into account is equivalent to categorizing a color space. Regardless of the chosen color space, the results vary with the task assigned to the user: whether the categorization is free, constrained by a fixed number of categories, or constrained by a naming task. The more the task is constrained, the more consensus are found [Bonnardel 06].

Choosing a naming task precisely leads to the eleven consensus regions corresponding to Berlin and Kay ’s BCTs. Determining whether language influences perception (relativistic view) or whether perception guides language (structuralistic view) is not clear. But there is undoubtedly a significant link in-between the two. Even in a free categorization task, there is no way to evaluate clearly the role played by language in the categorization process.

Apart from maximum consensus, a naming-based categorization has another important benefit: it does not depend on the set of colors to be categorized, whereas other categorizations will give different results for different color sets [Bonnardel 06]. Thus it can be considered as a “context-free” categorization.

However, as already noted, consensus regions found in this categorization are narrow and boundaries between regions are not well-defined because their identification depends on the observer and may even vary for a given observer in two experiments. Therefore, this categorization can only give, for a color taken out of its context, a probability distribution in each of the eleven BCTs. These unclear boundaries are no exceptions in language [Hardin 00]:

> "Typological features are not established in an absolute fashion, but in term of a *sliding scale* of cases, from best cases satisfying all criteria to a grey zone of unclear cases, to cases that do not satisfy any of the criteria."
>
> "It is in the nature of the dynamics of language, as a mutable system, for there to be grey zones, and for boundaries to be flexible."

Finally, it can be demonstrated [Bonnardel 06] that color categorization is not only spatial, but also temporal, allowing its use in a dynamic context (i.e., videos).

12.2 Lower-level properties of color perception

Unfortunately, a context-free categorization like the BCTs will not be efficient in practice. Indeed, vision science tells us that color cannot be taken out of context, and that lower levels of visual perception are expected to have a strong influence on categorization. In particular, there are three important properties to take into account before any categorization can take place: color constancy, transparency and simultaneous contrast.

Color constancy corresponds to the ability of the human visual system to perceive the reflectance color of an object and use this knowledge in the perception of its color. The most striking example is when one reads the pages of a book: independently of the environmental illumination (an out-door setting or near a bed-side lamp), characters will always be perceived as black ink onto a white page. Note, however, that this is a conscious perception, so that we are also able to abstract ourselves from this effect (painters are especially trained for this task).

Moreover, the transparent property of an object can lead to the perception of two or more nested or overlapping color regions in the image. Perception of transparency can be linked to color constancy since we are able to infer (again, rather consciously) the color of an object appearing behind a transparent one.

Finally, simultaneous contrast describes the tendency of the human visual system to relate colors spatially: we do not perceive colors in an image isolated from each other. At a low level, the surrounding of a retinal point must be taken into account in a color opponent way. At a higher level of perception, colors are perceived relatively to each other. This has an important consequence on discrimination tasks: the only class of judgements that leads to approximate consensus ratings are categorical ones, which is in accordance with a fuzzy color categorization of the image. Simultaneous contrast is also significant in a dynamic context.

Of the three perceptual properties presented above, only latter corresponds to very low levels of visual perception, since it operates unconsciously. Color constancy and transparency can be ignored if one is interested only in the distribution of colors *in the picture plane*: only the colors reflected by objects is then taken into account, which is the main concern when trying to extract a color palette for instance. Simultaneous contrast, however, has to be dealt with because it occurs in the front-end of the human visual system, before any categorization can occur. It thus should be studied in connection with multi-scale image analysis theories [Romeny 94, Alvarez 94], which are considered in the vision community as a biologically-plausible model of front-end vision. This is in accordance with the spatial analysis stage of our color distribution workflow, and is thus the main direction we will take in future work.

Part IV

Closing remarks

Discussion on the role of vision in expressive rendering

Through the chapters of this Thesis, while introducing both representation metaphors for the production of expressive animations and acquisition methods in line drawings and color images, I kept referring to the underlying theme of the intricate connections of expressive rendering with vision. I would like now to discuss in a more principled way and in more details what I believe is the role of vision science in the young field of expressive rendering. I hope this modest discussion will further motivate practical systems that truly enhance the way digital images are built by artists.

The dimensions of representation

Before going into the details of the human visual system (HVS) properties, I'd like to come back on an implicit consideration we've made in the context of animation (Part I): the dimensions of representation. These dimensions have first been described by Willats [Willats 97] where he makes a distinction between *object-centered* and *view-centered* representations. He gives a convincing example with projection systems. Perspective projections, as well as orthogonal and oblique projections, all lead to view-centered representations: the objects are projected solely based on viewpoint properties, not object properties. However, fold-out geometry found, for instance, in cubist paintings, is hard to describe derived from views; but it can be explained as derived from internal object-centered descriptions. This time, the object is more projected based on its semantic properties and the view plays a minor role, making the interpretation of a possible view harder (which was one of the goals of cubist painters).

This object-centered versus view-centered distinction can be found at any stage of the representation model. In the attributes system, the color of an object can depend on its material properties uniquely, as with a classic toon shader (object-centered); or can be affected by a view-dependent criterion such as depth, as with aerial perspective (view-centered). We have seen that primitives can either come from the projection of a 3D primitive onto the picture plane, which usually reinforce the shape properties of objects in the representation (object-centered); or they can be extracted in the picture plane, in practice flattening the representation (view-centered). And the mark system obeys the same classification: stroke properties (length, curvature or thickness for instance) can either come from some attribute mapping from the scene (object-centered), or be specified by a style and thus have no correspondence with any information in the scene (view-centered).

Of course, a representation does not have to be purely object- or view-centered, and most of them are a blend of the two. An interesting way to visualize this grading

has been proposed by McCloud [McCloud 94], who uses a two-dimensional diagram in order to classify previous work in comics history. His diagram represents a triangle as in Figure 2, where one vertex corresponds to a *realistic* representation, another one to a pure emphasis of the *picture plane*, and the third one to a pure abstraction only focused on the communication of *meaning*. Hence, the *Realism* → *Picture plane* axis roughly corresponds to the view-centered dimension, and the *Realism* → *Meaning* axis to the object-centered dimension. This visualization gives a way to define the style of a representation at a very high level, by locating it inside the triangle.

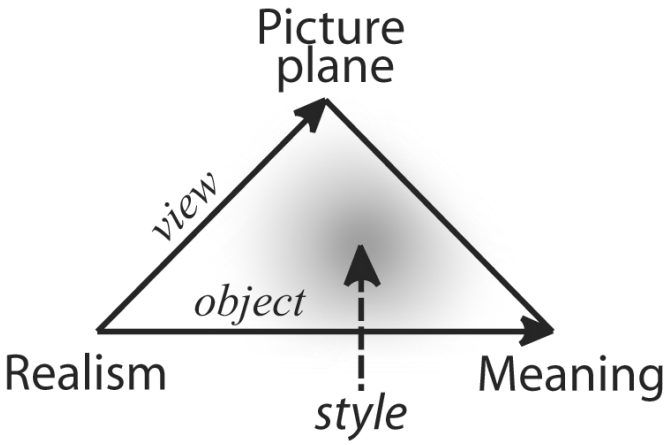


Figure 2: The triangle visualization proposed by McCloud [McCloud 94] to classify pictorial representations. The high-level style of any representation can be broadly located inside the triangle.

The role of vision

Now that we have organized a representation into two meaningful dimensions, we can look forward to describe the role played by human vision in the creation, as well as the interpretation of a representation. McCloud [McCloud 94] again evokes these tasks in a pedagogical manner:

> "Media convert thoughts into forms that can traverse the physical world and be reconverted by one or more senses back into thoughts."
>
> "The mastery of one's medium is the degree to which a project represents what the artist truly envisioned to be, the degree to which his ideas survive the journey."

The design of expressive rendering techniques should always keep this process in mind, i.e. new techniques should help the artist in expressing in the most straightforward way his idea or concept, through the representation of a scene. But when his thoughts are converted into form and "traverse the physical world", back into thoughts, they pass by a vision process: the artist "makes use" of his visual system to *create* the representation on one side; while the observer does the same to *interpret* it via his own perception on the other side.

If one wants to use computers in order to automate parts of the creative process, then the role of visual perception, both on the creative and interpretative sides, cannot be ignored. In the following I give a brief overview of some important properties of the human visual system, how they can influence the creation and the interpretation of a pictural representation, and their potential benefits in expressive rendering applications. For a more complete overview of vision models, the interested reader is referred to the book by Palmer [Palmer 99].

Marr's four stages of Vision Vision is a heuristic process in which inferences are made about the most likely environmental condition that could have produced a given image. According to Marr, there are four stages of visual perception. They are named for the kind of information they represent explicitly; they are the image-based, surface-based, object-based and category-based stages of perception. At each level, current processing is influenced by prior higher-level interpretations, and this combination of top-down and bottom-up operations is what allows inferences of the visual system to be made (see Figure 3).

However, the information captured by the eyes is not processed as a whole by the visual system. Very quickly, various kinds of information take different paths and are treated rather in parallel. Indeed, there seems to be four different pathways in the visual system: color, form, stereo (depth) and motion. The separation is not complete, and it is only a vast simplification, but it motivates, at least at a computational level, the parallel processing of these four different kinds of information. Note that the separations adopted in this Thesis follow this property: Part I deals with motion, Part II is more about form, and Part III is interested in color. I did not really work on depth in my Thesis, but it is an interesting area of future work.

Another important property of the visual system is that perception might be conscious at some stage, and unconscious at others. In particular, we are able to perceive consciously two different stimulii: the *proximal stimulus*, which corresponds to features in the picture plane; and the *distal stimulus*, corresponding to features of objects in the scene. The proximal stimulus hence is analyzed by the image-based stage, i.e. the perception of low-level information like the color reflected from an object, a texture pattern, or the contrast along a silhouette; while the distal stimulus is analyzed by the subsequent three stages, and gives the properties of surfaces (sharp angles, orientation, depth, etc), objects (shape, material properties, etc) and their category (is it a chair, a face ?). This is illustrated in Figure 3 by the dashed boxes.

The distinction between proximal and distal stimulii is very important, since it is this ability that allows, for instance, an observer to see in a painting a collection of paint strokes on a canvas, as well as the representation of a possible scene. Proximal and distal stimulii are directly linked to the previously mentioned notions of view- and object-centered representations respectively: indeed, a representation can trigger more or less each of the stimulii in order to orient it towards the picture plane or scene features. The study of the perception of proximal and distal stimulii is thus of primary interest to expressive rendering.

Therefore, I will briefly present in the following some of the most important properties of the proximal and distal stimulii, and discuss the practical benefits of their incorporation into expressive rendering systems.

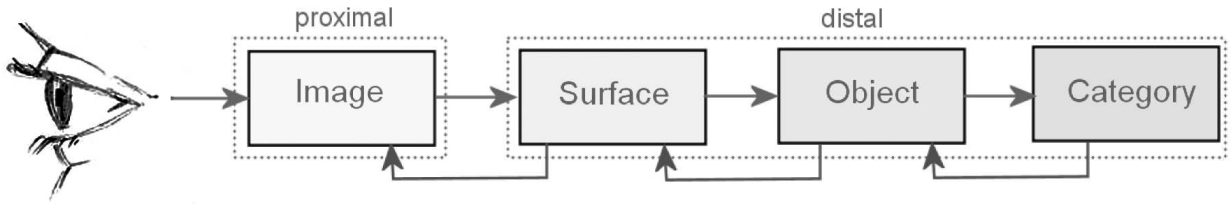


Figure 3: The four stages vision model proposed by David Marr.

Perception of the proximal stimulus According to Marr's theory, the first analysis the HVS performs is image-based, hence it only deals with information found in the retinal image. However, objects move rigidly in 3D, illumination varies, the size of objects in the retina changes with the depth from the eye, view directions may change, etc. It is thus natural to require early visual operations to be unaffected by certain primitive transformations (e.g. translations, rotations, and grey-scale transformations). In other words, the visual system should extract properties that are *invariant* with respect to these transformations. A common trend, which has been shown to be biologically plausible, is to represent the visual signal at multiple scales that hold these invariant properties (see [Romeny 94] and [Alvarez 94] for a deeper treatment). These scale-space systems deal primarily with the spatial integration of low-level visual information in the front-end of the HVS.

We already evoked the potential advantages of using multi-scale representations in Part III. In the first stage of our color composition workflow, i.e. the extraction of a color palette, a multi-scale analysis would allow us to take into account simultaneous contrast effects occuring at different scales in the image. It also makes a well-suited representation to extract color regions in the second step of the workflow, spatial distribution, since regions of various sizes could be created by the user through a direct manipulation of the multi-scale representation. This approach has already been investigated in [Bangham 03]: while they only deal with regions having hard boundaries, their approach motivates future work in the field.

Another advantage of using a multi-scale representation is in the detection of discontinuities. Edge detection is commonly linked to the contrast sensitivity of the HVS: center-surround inhibitory operations occur at the early stages of vision, and allow to detect luminance gradients; Then, edges are inferred from contrast information, and discriminated between reflectance edges (e.g. made by a hard shadow) and discontinuity edges (e.g. along a silhouette). Line junctions play an important role in this discrimination and are usually classified based on a junction catalog (T-, L-, Y- and end-junctions are the most frequently encountered). Extracting edges and junctions at multiple scales gives even more information: it allows to recover fine-to-coarse structures in an image, and would be of practical use, for instance, in the automatic analysis and manipulation of line drawings as presented in Chapter 7.

To summarize, scale-space theory seems to account for the perception of regions of various shape and color, as well as the perception of edges at various scales in the retinal image. These cues constitute the first available information extracted from the proximal stimulus.

Other properties are extracted at this point, see [Palmer 99] for a detailed presentation. For instance, an important mechanism is the figure/ground separation: when

it considers two regions, the HVS has a strong preference to ascribe the contour of just one of its bordering regions (the figure) and to perceive the other side as part of a surface extending behind it (the ground). The factors that help the HVS to discriminate figure from ground are many: surroundedness, size, orientation, contrast, symmetry, convexity and parallelism are the most important. The remaining figure regions may be related together by the laws of perceptual organization: regions are grouped based on proximity, similarity of color, size, orientation, similarity of velocity (also called common fate), symmetry, parallelism, continuity and closure for the most important criteria. In this process, past experience is also an important factor: if elements have been previously associated in prior viewings, they will tend to be seen as grouped in present situations.

The combined effects of figure/ground and perceptual organization are particularly important in the perception of textures, which are sets of figure regions defined by locality and stationarity properties. Textures have been extensively studied in the vision community and a lot of models exist to describe the HVS behavior in perceiving textural images. In the context of expressive rendering, the study of texture perception is of particular interest when analyzing stroke patterns, as presented in Chapter 8. In particular, figure/ground models would allow us to coherently discriminate between the strokes and the background, while perceptual organization and statistical texture models could relate strokes together, and thus extract the distribution properties that are essential to reproduce in order to synthesize perceptually similar patterns.

Finally, optic flow is the dominant proximal motion stimulus for normal everyday vision, and here again, some interesting properties of the HVS have been observed. Paradoxical motion, for instance, occurs when there is a clean perception of motion but no global change in the perceived position of the moving objects. It can partly be explained by the presence of motion contrast operators that act on the optic flow. Motion transparency corresponds to the perception of two or more overlapping optic flows. It occurs not only with transparent objects such as water, but also with fast moving objects and/or observer, like with a tree foliage in the wind. It is however limited to a small number of layers (between three and four).

Motion contrast and transparency models are needed when rendering expressive animations. For instance, the 2D stroke patterns presented in Chapter 4 would benefit from a practical motion transparency model: it would allow to determine whether the overlapping of two patterns moving in 2D produce a motion transparency percept (i.e. a disturbing sliding effect), and thus whether we should use an additional pattern to faithfully represent the 3D motion.

Perception of the distal stimulus Once elements of the proximal stimulus have been extracted, they are further processed by the HVS to progressively give distal information. However, one should not forget that distal information can influence the perception of the proximal stimulus via a top-down mechanism, as shown in Figure 3.

The first information that is looked for is surface information (in the surface-based stage). According to J. J. Gibson, surface perception can be conceived as determined by its distance and orientation, the latter being defined by the slant and tilt of the surface with respect to the viewer's line of sight. In the vision literature, approaches

that try to model how the HVS infers surface are called shape-from-X models, even if they should be termed surface-from-X because they only retrieve surface information.

Examples of models include shape-from-shading, shape-from-textures and shape-from-reflections models. These are numerous and I only present some simple ideas of their functioning here. To retrieve orientation, some models measure a deformation of the observed proximal property (luminance, texture or reflections) in a preferred orientation and use it to find the orientation of the underlying surface. While for depth, many measured proximal properties are gathered (like binocular disparity, relative size, position relative to the horizon, occlusion and motion parallax) and fused in sometimes complex ways (modified weak fusion for instance, where some properties are either "promoted" by others, or independently combined).

Such models are very useful in expressive rendering when the goal is precisely to depict depth or orientation, i.e. surface cues. For instance, in Chapter 5, we conjectured that surface perception models are in close relation to contour rendering of a 3D object. I.e., studying the way surface depth, orientation and curvature are reconstructed from shading, textures or reflections might give better insights into how the same surface information is retrieved from line drawings.

Surface information, while it is part of the distal stimulus (an information about objects in the scene), corresponds to an extrinsic description: it is only valid from the current point of view. The next stage for the HVS is to use this information in object-based processes to infer intrinsic information about the objects, i.e. their view-independent characteristics. This corresponds to a higher level of vision, as compared to low-level vision (i.e. the image-based stage) and mid-level vision (i.e. the surface-based stage). It is thus harder to find models of the HVS at this stage.

Existing approaches describe for instance how the HVS perceives object materials or transparency (e.g., [Fleming 03]), or how it retrieves the color of an object, i.e. color and lightness constancy. They also describe how 3D motion can be inferred from optical flow and other informations, see [Palmer 99] for more details. Among all these properties, the ability of the HVS to perceive the shape of an object and its parts is maybe the more complex to understand. There are many models of shape perception (Palmer presents four different models in his book); All of them posess great advantages in describing this shape perception ability of the HVS, but they all have drawbacks that make them inapplicable to describe shape perception in its entirety. The use of such models in expressive rendering might then be a little prematurate, but they promise to allow even more powerful tools, especially for sketch-based modeling applications.

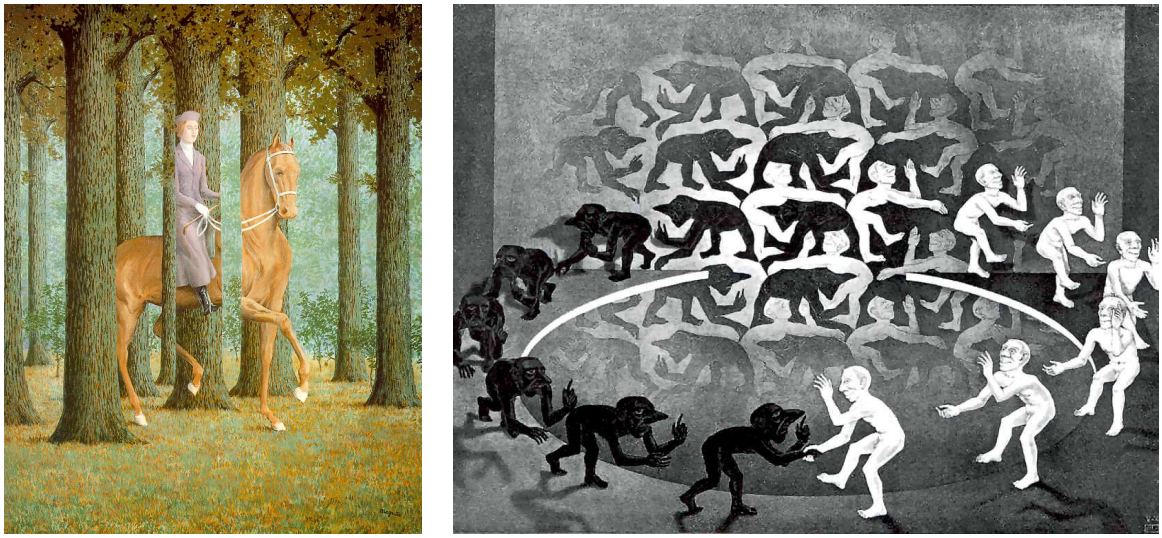
Finally, moving to the category-based stage, models are even more complex to find. Color categorization is an exception, and it has been thoroughly presented in Chapters 11 and 12. Object perception is way more complex, probably because of the greater number of possible categories. An important remark that I already made in Chapter 12 is that categorization is tied to language, and it makes it difficult to decide whether language influence perception, or perception influence language.

Perception and schema Recall that schema, defined by Gombrich [Gombrich 61] as "mere illusions that lead to different styles", correspond to artistic techniques that are invented and then learnt by successive generations of artists. An interesting

hypothesis, formulated by Durand [Durand 00], is that these schema are hints about the way the HVS works.

In Chapter 2 of this Thesis, we proposed an intuitive technique to create aerial perspective effects. Aerial perspective is an example of a schema used by artists in order to give perceptual cues of depth, which is absent from a canvas. It uses the commonly encountered effects of haze in an outdoor environment in an artistic way, so that the visual system is able to retrieve depth information, a distal property, from tonal modifications, a proximal property. Another example is the use of silhouette lines: most of the time they correspond to a contrast perception in the proximal stimulus, but correspond to more complex properties of the distal stimulus, i.e. depth discontinuities.

These observations suggest that visual perception might be an important factor of the evolution of schema through art history: those schema that can efficiently convey perceptual cues, invented by inspired artists, are taught to the next generation of artists that combine them in new ways, or may even improve them. Some of the works of Magritte (Figure 4(a)) or Escher (Figure 4(b)) are good illustrations of the connections between schema and perception: they systematically violate schema rules in order to set ambiguous compositions where a valid perceptual interpretation cannot be performed (here they play with figure/ground perception).



(a) (b)

Figure 4: (a) “Le blanc-seing” by R. Magritte. (b) “Encounter” by M. C. Escher.

Putting the artist into the loop

From the last section we recall that view-centered representations tend to stimulate mainly the proximal mode, by emphasizing properties of the picture plane, and the relations amongst the 2D primitives that are part of it; while object-centered representations tend to stimulate mainly the distal mode, by trying to show object parts, properties and functions.

I thus propose to view expressive rendering techniques as tools that perform synthesis and analysis steps in-between the creation of a representation via the systems

of Willats and Durand, and its inspection via the vision model of Marr (this is illustrated in Figure 5). The user can be part of this back-and-forth system in two ways: he can act as a designer by guiding the synthesis processes, i.e. by taking decisions concerning the representation systems; and he can refine the analysis processes so that the final rendering comes the closest to his goals. This definition bears many resemblances with the way artists work with traditional media: they alternate creation and inspection steps that are similar to these synthesis and analysis processes.

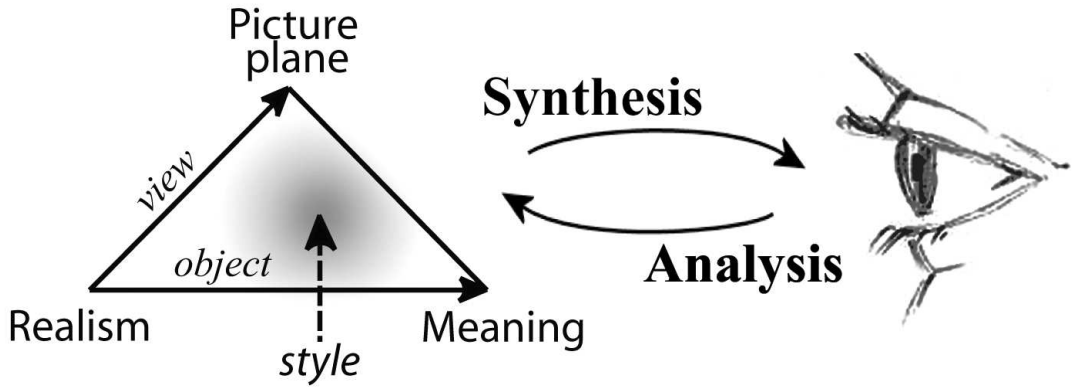


Figure 5: Expressive rendering techniques can be designed as alternations between analysis and synthesis steps.

The extent to which the user interacts with such a system will depend of course on the application. In the following, I propose three classes of applications, that differ from each other in the degree of implication of the user, as well as the amount of analysis that has to be performed by the system to infer a correct representation. I called these classes refinement-based, example-based and sketch-based applications and they are detailed below. Note that I do not claim this classification to be complete; it is rather a starting point which, I hope, will help relate various expressive rendering applications and give them common evaluation methods.

Refinement-based applications In such applications, comprising expressive and photorealistic renderings, the user has to rely on his own interpretation of the scene to choose amongst the set of parameters to get the expected result. Then, he can refine those parameters in order to converge to a better solution. For example, in order to assign a material to an object in a classic computer graphics system, the artist uses an a priori on its properties to choose the right set of material parameters, and then modifies their values until a satisfying result is obtained. With such an approach, as we already said in Chapter 5, feedback is of course very important, but also the use of a perceptually uniform parameter space so that a small change in a parameter value will impart a similarly small change in the visual result.

In some cases, this approach can be tedious, or even intractable, in particular with expressive renderings. For instance, in Chapter 5, we envisioned the use of multiple overlap 2D stroke patterns to represent the shading and motion of a 3D object. Letting to the user the task of setting the number of patterns and their overlapping in order to reduce their apparent sliding, for each frame of an animation, sounds like a particularly tedious task. Therefore, in future work, we plan to implement an automated analysis of the representation, through the use of motion transparency models;

this way, the system will be able to refine automatically the number of patterns and their overlap, and hence improve the representation at the same time ensuring a minimum sliding.

In the case above, "sliding" refers to a perceptual property of the proximal stimulus, i.e. overlapping motion flows. Refining a representation based on the perceptual properties of the distal stimulus is more complex, since it involves a deeper analysis of the representation. We already talked about such a task, in Chapter 5, when we evoked the yet unsatisfying results of current line rendering techniques. Many of the lines extracted with the available methods at the time of writing this Thesis would still need manual editing, but this is, again, a tedious task to perform. In order to create better line renderings, we must first understand more precisely what object features (distal stimulus) they represent. This way, we will be able to let the computer automatically perform the refinement process, using the lines generated and what they are supposed to represent.

Example-based applications Example-based applications allow the artist to give a sample of the style he wants to create (e.g., by drawing it), or maybe only a reference (e.g., via a reference painting). They can be generalized using analogies: knowing examples A, A', and B and a relation $A : A'$, it consists in creating B' so that $A : A' :: B : B'$. Two specific analogies are of interest: synthesis, where an example A' is given and we want to synthesize more in B' ($\emptyset : A' :: \emptyset : B'$); and transfer, where a source image B is changed into a target image B' to match the colors of a reference image A' ($\emptyset : A' :: A : B'$).

For example, with stroke pattern synthesis, the user gives a set of strokes A' that are analyzed and used to create a similar set B' of different size, but with similar properties. This kind of application typically involves the analysis of a proximal stimulus, since a pattern can be considered as a specific class of texture. Thus, models of texture perception might be used in order to synthesize patterns that are perceptually similar to the provided example. In Chapter 8, we used perceptual organization models in order to perform the synthesis. In the future, as proposed in Chapter 9, we plan to investigate more deeply the parametric texture models found in vision science.

Another example is color transfer, which consists in modifying a source image B into a target one B' so that it matches the colors of a reference image A'. The relation between B and A' is here of primary interest, and to retrieve such a relation, we proposed in Chapter 11 to match the color palettes of B and A'. While we only concentrated on the extraction of these palettes, we found that it is a complex problem since it involves to go from a proximal stimulus (pixel colors) to a high-level distal stimulus (color categories), without any a priori information on the representation. In a future work we plan to investigate more intermediate stages of the distal stimulus perception as discussed in Chapter 12, using for example the scale-space theory to get a more progressive analysis.

Finally, silhouette annotation is a good example of analogy. Here, the user shows how to modify a silhouette A to give a new one A'. He can then apply this "style" to another silhouette B to get a stylized silhouette B'. While this task would be complex without any information about the scene, as with color transfer, it most of the time comes with a lot of informations about the distal stimulus. Indeed, in practice, silhouette primitives (the distal stimulus) can be selected from a 3D model and projected in the picture plane. Then, the user draws on top of that silhouette and the same stylization is "easily" applied to any other silhouette, since they are readily available

in the scene. However, when such a distal information is not available, or the analogy is more complex, then a more sophisticated analysis of the relation between A and A' has to be performed.

Sketch-based applications Sketch-based applications encompass all techniques where the user gives a sample of the results he expects, and the system tries to infer the higher-level data that could have given such a sample. Those models thus make use of distal and proximal analysis to help the user recover data at higher levels of the representation model. It can happen at any level: from marks to 2D primitives, from 2D primitives to 3D primitives or from 2D primitives to attributes.

For example, over-sketching tools or segmentation algorithms are 2D sketch tools where marks (strokes or pixels) are organized into more meaningful 2D primitives (contours or color regions). We proposed an over-sketching tool in Chapter 7.

Sketch-based modelling applications are 3D sketch tools, since a 3D shape is inferred from simple line drawings. Similarly, relighting applications are attribute sketch tools, where properties of materials and lights are retrieved from sparse color samples input by the user. These applications are very complex in nature because they are typically under-constrained; they thus need additional sets of constraints that are usually provided via domain-specific problems, but may also be given by the use of sophisticated vision models (see for instance [Karpenko 06]).

Evaluation methods

Now that we have roughly organized applications into three classes, we can devise specific evaluation methods that will allow to compare and resituate expressive rendering techniques. We thus turn to some propositions for each class of applications.

Refinement-based applications sometimes have the advantage that they can be evaluated by comparing the result of a rendering with a ground-truth image. Similar methods can be used to evaluate both Photorealistic renderings and media simulations, by comparing their results to real-life measurements. However, for representations departing from simulation, the evaluation is much more arbitrary (e.g. toon-shading): it can only rely on more subjective comparisons with paintings and drawings. A better way to evaluate such systems is to first give desirable perceptual properties of the final result (as with the “no-sliding” constraint for stroke patterns motion), and then evaluate a method’s results on this basis.

Example-based models could be evaluated through the use of rating experiments: since the goal is to match an analogy, two different methods can be compared based on perceptual evaluation. A real observer could be prompted to pick a most similar candidate out of many method results and reaction times could be used to quantify this rating.

Finally, sketch-based models could be evaluated by model comparison: by drawing or painting on top of an existing model, and comparing the inferred and existing model. The closer the sketch depicts the synthesized shape or material, the better the method is. The problem of defining distance metrics holds, and one could use perceptual properties to this end.

Conclusions

I presented in this thesis new representation and acquisition models that augment the possibilities in expressive rendering. Thanks to the work made with my collaborators, new representations have been created that take inspiration from traditional artistic techniques and generalize them under a common logic, this way giving intuitive tools to the user. The acquisition of low-level stylistic information from vector line drawings and bitmap color images has also progressed in the direction of a "hands-on" interface for the user.

Not only do these contributions have potential practical applications, but they have also been put in a broader context where human visual perception models are used to predict, refine, extract, compare and evaluate expressive rendering methods. While we only discussed the potentials of vision knowledge in the creation of expressive representations, we actually made a direct use of perceptual mechanisms in the acquisition of style from user-provided examples. We only scratched the surface in the use of human vision models, but we also motivated their potential benefits in future work by discussing interesting practical couplings between perception and expressive rendering applications.

Expressive rendering has still many lessons to learn from traditional painting, drawing, photography and cinematography techniques, and I proposed in this thesis to do so in a principled way: by carefully studying existing artistic techniques, or schema, and understanding to what aspects of the visual experience they correspond. This is especially interesting, not only because it links expressive rendering to the analysis of existing pieces of artwork, but also because it allows to explore new aspects of human visual perception that can only be triggered by expressive representations.

Bibliography

[Agrawala 00] Maneesh Agrawala, Denis Zorin & Tamara Munzner. *Artistic Multiprojection Rendering*. In Proceedings of the Eurographics Workshop on Rendering Techniques 2000, pages 125–136, London, UK, 2000. Springer-Verlag. 12

[Agrawala 01] Maneesh Agrawala & Chris Stolte. *Rendering Effective Route Maps: Improving Usability Through Generalization*. In Eugene Fiume, editeur, SIGGRAPH 2001, Computer Graphics Proceedings, pages 241–250. ACM Press / ACM SIGGRAPH, 2001. 12

[Agrawala 03] Maneesh Agrawala, Doantam Phan, Julie Heiser, John Haymaker, Jeff Klingner, Pat Hanrahan & Barbara Tversky. *Designing effective step-by-step assembly instructions*. ACM Trans. Graph., vol. 22, no. 3, pages 828–837, 2003. 12

[Alvarez 94] L. Alvarez & J-M. Morel. *Formalization and computational aspects of image analysis*. Acta Numerica, vol. 3, pages 1–59, 1994. 109, 139, 146

[Andersen 89] G. J. Andersen. *Perception of three-dimensional structure from optic flow without locally smooth velocity*. Journal on Experimental Psychology and Human Perception Performance, vol. 15, no. 2, pages 363–371, 1989. 63

[Anjyo 03] Ken-ichi Anjyo & Katsuaki Hiramitsu. *Stylized Highlights for Cartoon Rendering and Animation*. IEEE Comput. Graph. Appl., vol. 23, no. 4, pages 54–61, 2003. 13, 23, 28

[Anjyo 06] Ken-ichi Anjyo, Shuhei Wemler & William Baxter. *Tweakable light and shade for cartoon animation*. In NPAR '06: Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering, pages 133–139, New York, NY, USA, 2006. ACM Press. 13, 23, 28

[Ashikhmin 01] Michael Ashikhmin. *Synthesizing natural textures*. In SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics, pages 217–226, New York, NY, USA, 2001. ACM Press. 75

[Bangham 03] J.A. Bangham, S.E. Gibson & R.W. Harvey. *The Art of Scale-Space*. In British Machine Vision Conference, 2003. 118, 135, 146

[Barla 05] Pascal Barla, Joëlle Thollot & François Sillion. *Geometric Clustering for Line Drawing Simplification*. In Proceedings of the Eurographics Symposium on Rendering, 2005. 77

[Barla 06a] Pascal Barla, Simon Breslav, Joëlle Thollot & Lee Markosian. *Interactive Hatching and Stippling by Example*. In INRIA Technical report, 2006. 74, 94, 100

[Barla 06b] Pascal Barla, Simon Breslav, Joëlle Thollot, François Sillion & Lee Markosian. *Stroke Pattern Analysis and Synthesis*. In Computer Graphics Forum (Proc. of Eurographics 2006), volume 25, 2006. 93

[Barla 06c] Pascal Barla, Lee Markosian & Joëlle Thollot. *X-Toon: An extended toon shader*. In International Symposium on Non-photorealistic Animation and Rendering (NPAR), 2006. 21

[Barrow 81] Harry G. Barrow & Jay M. Tenenbaum. *Interpreting Line Drawings as Three-Dimensional Surfaces.* Artif. Intell., vol. 17, no. 1-3, pages 75–116, 1981. 108

[Baudel 94] Thomas Baudel. *A mark-based interaction paradigm for free-hand drawing*. In UIST '94: Proceedings of the 7th annual ACM symposium on User interface software and technology, pages 185–192, New York, NY, USA, 1994. ACM Press. 73, 78

[Baxter 04] William Baxter, Jeremy Wendt & Ming C. Lin. *IMPaSTo - A Realistic, Interactive Model for Paint*. In 3rd International Symposium on Non-Photorealistic Animation and Rendering (NPAR'04), pages 45–56, June 2004. 19

[Berlin 69] Brent Berlin & Paul Kay. Basic color terms: Their universality and evolution. University of California Press, 1969. 123, 137

[Beusmans 87] J. Beusmans, D.D. Hoffman & B.M. Bennett. *Description of solid shape and its inference from occluding contours*. journal of the Optical Society of America, vol. 4, pages 1155–1167, 1987. 108

[Black 98] M. J. Black, G. Sapiro, D. Marimont & D. Heeger. *Robust anisotropic diffusion*. IEEE Trans. on Image Processing, vol. 7, no. 3, pages 421–432, 1998. 118

[Bonnardel 06] V. Bonnardel. *Color naming and categorization in inherited color vision deficiencies*. Visual Neuroscience, vol. 23, no. 3, 2006. 138, 139

[Bousseau 06] Adrien Bousseau, Matt Kaplan, Joëlle Thollot & François X. Sillion. *Interactive watercolor rendering with temporal coherence and abstraction*. In International Symposium on Non-Photorealistic Animation and Rendering (NPAR), 2006. 15, 18, 51, 118

[Chang 03] Youngha Chang, Suguru Saito & Masayuki Nakajima. *A Framework for Transfer Colors Based on the Basic Color Categories.* In Computer Graphics International, pages 176–183, 2003. 117, 123

[Chi 06] Ming-Te Chi & Tong-Yee Lee. *Stylized and Abstract Painterly Rendering System Using a Multiscale Segmented Sphere Hierarchy*. IEEE Transactions on Visualization and Computer Graphics, vol. 12, no. 1, pages 61–72, January/February 2006. 18, 37

[Chu 05] Nelson S.-H. Chu & Chiew-Lan Tai. *MoXi: real-time ink dispersion in absorbent paper*. ACM Trans. Graph., vol. 24, no. 3, pages 504–511, 2005. 19

[Coleman 04] Patrick Coleman & Karan Singh. *RYAN : Rendering your animation nonlinearly projected*. In 3rd International Symposium on Non-Photorealistic Animation and Rendering (NPAR'04), 2004. 12

[Collomosse 04] J. P. Collomosse & P. M. Hall. *A Mid-level Description of Video, with Application to Non-photorealistic Animation*. In 15th British Machine Vision Conference (BMVC), 2004. 15

[Comaniciu 99] Dorin Comaniciu & Peter Meer. *Mean Shift Analysis and Applications*. In ICCV (2), pages 1197–1203, 1999. 118

[Cornish 01] Derek Cornish, Andrea Rowan & David Luebke. *View-dependent particles for interactive non-photorealistic rendering*. In GRIN'01: No description on Graphics interface 2001, pages 151–158, Toronto, Ont., Canada, Canada, 2001. Canadian Information Processing Society. 18, 41

[Cui 01] G. Cui, M. R. Luo, B. Rigg, G. Roesler & K. Witt. *Uniform Colour Spaces Based on the DIN99 Colour-Difference Formula*. Colour and Imaging Inst., Univ. of Derby, 2001. 122, 133

[Cunzi 03] Matthieu Cunzi, Joëlle Thollot, Sylvain Paris, Gilles Debunne, Jean-Dominique Gascuel & Frédo Durand. *Dynamic Canvas for Immersive Non-Photorealistic Walkthroughs*. In Proc. Graphics Interface. A K Peters, LTD., june 2003. 51

[Curtis 97] Cassidy J. Curtis, Sean E. Anderson, Joshua E. Seims, Kurt W. Fleischer & David H. Salesin. *Computer-generated watercolor*. In Siggraph 97, Proceedings of the 24th annual conference on Computer graphics and interactive techniques, pages 421–430. ACM Press/Addison-Wesley Publishing Co., 1997. 19

[Daniels 99] Eric Daniels. *Deep canvas in Disney's Tarzan*. In SIGGRAPH '99: ACM SIGGRAPH 99 Conference abstracts and applications, page 200, New York, NY, USA, 1999. ACM Press. 16

[DeCarlo 03] Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz & Anthony Santella. *Suggestive Contours for Conveying Shape*. ACM Transactions on Graphics, SIGGRAPH, vol. 22, no. 3, pages 848–855, July 2003. 17, 86, 108

[Deussen 00a] Oliver Deussen, Stefan Hiller, Cornelius Overveld & Thomas Strothotte. *Floating Points: A Method for Computing Stipple Drawings*. Computer Graphics Forum (EG'00), vol. 19, no. 3, pages 40–51, 2000. 72

[Deussen 00b] Oliver Deussen & Thomas Strothotte. *Computer-Generated Pen-and-Ink Illustration of Trees*. In Proceedings of ACM SIGGRAPH 2000, Computer Graphics Proceedings, Annual Conference Series, pages 13–18, 2000. 18, 78

[Durand 00] Frédo Durand. *The art and science of Depiction*. In Unpublished manuscript, 2000. x, 8, 59, 149

[Durand 02] Frédo Durand. *An Invitation to Discuss Computer Depiction*. In 2nd International Symposium on Non-Photorealistic Animation and Rendering (NPAR'02), Annecy, France, June 3-5 2002. vi, vii, x, 8, 9, 36, 60

[Efros 99] Alexei A. Efros & Thomas K. Leung. *Texture Synthesis by Non-parametric Sampling*. In IEEE International Conference on Computer Vision, pages 1033–1038, 1999. 75, 97, 98

[Eissele 04] Mike Eissele, Daniel Weiskopf & Thomas Ertl. *Frame-to-Frame Coherent Halftoning in Image Space*. In Proceedings of Theory and Practice of Computer Graphics 2004, pages 188–195, 2004. 51

[Elad 02] Michael Elad. *On the origin of the bilateral filter and ways to improve it*. IEEE Transactions on Image Processing, vol. 11, no. 10, pages 1141–1151, 2002. 118

[Elder 96] James H. Elder & Steven W. Zucker. *Computing Contour Closure*. In ECCV (1), pages 399–412, 1996. 78

[Etemadi 91] A. Etemadi, J.P. Schmidt, G. Matas, J. Illingworth & J.V. Kittler. *Low-Level Grouping of Straight Line Segments*. In British Machine Vision Conf., pages 119–126, 1991. 95

[F. Pitié 05] A. Kokaram F. Pitié & R. Dahyot. *Towards Automated Colour Grading*. In 2nd IEE European Conference on Visual Media Production, 2005. 117

[Fleming 03] Roland W. Fleming, Ron O. Dror & Edward H. Adelson. *Real-world illumination and the perception of surface reflectance properties*. J. Vis., vol. 3, no. 5, pages 347–368, 7 2003. 108, 148

[Fleming 04] Roland W. Fleming, Antonio Torralba & Edward H. Adelson. *Specular reflections and the perception of shape*. J. Vis., vol. 4, no. 9, pages 798–820, 2004. 108

[Freeman 03] William T. Freeman, Joshua B. Tenenbaum & Egon Pasztor. *Learning Style Translation for the Lines of a Drawing*. ACM Transactions on Graphics, vol. 22, no. 1, pages 1–14, January 2003. 73, 74

[Freudenberg 01] Bert Freudenberg, Maic Masuch & Thomas Strothotte. *Walk-Through Illustrations: Frame-Coherent Pen-and-Ink Style in a Game Engine*. In A. Chalmers & T.-M. Rhyne, editeurs, In Proceedings of Eurographics 2001, volume 20(3) of *Computer Graphics Forum*, pages 184–191, 2001. 49, 54

[Girshick 00] Ahna Girshick, Victoria Interrante, Steven Haker & Todd Lemoine. *Line direction matters: an argument for the use of principal directions in 3D line drawings*. In NPAR 00, Proceedings of the 1st international symposium on Non-photorealistic animation and rendering, pages 43–52. ACM Press, 2000. 40

[Gombrich 61] Ernst Hans Gombrich. Art and illusion: A study in the psychology of pictorial representation. Princeton University press, 1961. viii, 148

[Gooch 99] Bruce Gooch, Peter-Pike J. Sloan, Amy Gooch, Peter Shirley & Richard Riesenfeld. *Interactive technical illustration*. In Proceedings of the 1999 symposium on Interactive 3D graphics, pages 31–38. ACM Press, 1999. 13

[Gooch 01] Bruce Gooch & Amy Gooch. Non-photorealistic rendering. A. K. Peters, Ltd., 2001. vi, 78

[Grabli 04a] Stephane Grabli, Frédo Durand & Francois X. Sillion. *Density Measure for Line-Drawing Simplification*. In 12th Pacific Conference on Computer Graphics and Applications (PG'04), pages 309–318, Seoul, Korea, October 06 - 08 2004. 72, 73, 78

[Grabli 04b] Stéphane Grabli, Emmanuel Turquin, Frédo Durand & François Sillion. *Programmable Style for NPR Line Drawing*. In Rendering Techniques 2004 (Eurographics Symposium on Rendering). ACM Press, June 2004. 48, 78

[Grundland 05a] Mark Grundland & Neil A. Dodgson. *Color Histogram Specification by Histogram Warping*. In Proceedings of SPIE, volume 5667, 2005. 117

[Grundland 05b] Mark Grundland & Neil A. Dodgson. *Color Search and Replace*. In Computational Aesthetics, 2005. 116, 135

[Haeberli 90] Paul Haeberli. *Paint By Numbers: Abstract Image Representations*. ACM SIGGRAPH Computer Graphics, vol. 24, no. 4, pages 207–214, August 1990. 14

[Hardin 00] C. L. Hardin & Luisa Maffi. *Color Categories in Thought and Language*. Minds Mach., vol. 10, no. 3, 2000. 137, 138

[Hays 04] James Hays & Irfan Essa. *Image and Video Based Painterly Animation*. In 3rd International Symposium on Non-Photorealistic Animation and Rendering (NPAR'04), pages 113–120, Annecy, France, June 07-09 2004. 14

[Hertzmann 98] Aaron Hertzmann. *Painterly rendering with curved brush strokes of multiple sizes*. In Siggraph 98, Proceedings of the 25th annual conference on Computer graphics and interactive techniques, pages 453–460. ACM Press, 1998. 14

[Hertzmann 00] Aaron Hertzmann & Denis Zorin. *Illustrating smooth surfaces*. In Siggraph 00, Proceedings of the 27th annual conference on Computer graphics and interactive techniques, pages 517–526. ACM Press/Addison-Wesley Publishing Co., 2000. 49

[Hertzmann 02a] Aaron Hertzmann. *Fast paint texture*. In NPAR '02: Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering, pages 91–ff, New York, NY, USA, 2002. ACM Press. 19, 48

[Hertzmann 02b] Aaron Hertzmann, Nuria Oliver, Brian Curless & Steven M. Seitz. *Curve Analogies*. In Rendering Techniques 2002: 13th Eurographics Workshop on Rendering, pages 233–246, June 2002. 74

[Hertzmann 03] Aaron Hertzmann. *A Survey of Stroke-Based Rendering*. IEEE Computer Graphics and Applications, vol. 23, no. 4, pages 70–81, July/August 2003. Special Issue on Non-Photorealistic Rendering. 19, 93

[Horn 87] Berthold K. P. Horn. *Closed-form solution of absolute orientation using unit quaternions*. Journal of the Optical Society of America, vol. 4, no. 4, pages 629–642, April 1987. 52

[Hsu 93] S. C. Hsu, I. H. H. Lee & N. E. Wiseman. *Skeletal strokes*. In UIST '93: Proceedings of the 6th annual ACM symposium on User interface software and technology, pages 197–206, New York, NY, USA, 1993. ACM Press. 19

[Igarashi 97] Takeo Igarashi, Satoshi Matsuoka, Sachiko Kawachiya & Hidehiko Tanaka. *Interactive Beautification: A Technique for Rapid Geometric Design*. In UIST (ACM Annual Symposium on User Interface Software and Technology), pages 105–114, 1997. 73, 78

[Interrante 97] Victoria Interrante. *Illustrating surface shape in volume data via principal direction-driven 3D line integral convolution*. In Siggraph 97, Proceedings of the 24th annual conference on Computer graphics and interactive techniques, pages 109–116. ACM Press/Addison-Wesley Publishing Co., 1997. 40, 49

[Isenberg 03] Tobias Isenberg, Bert Freudenberg, Nick Halper, Stefan Schlechtweg & Thomas Strothotte. *A Developer's Guide to Silhouette Algorithms for Polygonal Models*. IEEE Computer Graphics and Applications, vol. 23, no. 4, pages 28–37, July/August 2003. 16

[Jodoin 02] Pierre-Marc Jodoin, Emric Epstein, Martin Granger-Piche & Victor Ostromoukhov. *Hatching by Example: a Statistical Approach*. In 2nd International Symposium on Non-Photorealistic Animation and Rendering (NPAR'02), Annecy, France, June 3-5 2002. 74

[Kalnins 02] Robert D. Kalnins, Lee Markosian, Barbara J. Meier, Michael A. Kowalski, Joseph C. Lee, Philip L. Davidson, Matthew Webb, John F. Hughes & Adam Finkelstein. *WYSIWYG NPR: drawing strokes directly on 3D models*. In SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques, pages 755–762, New York, NY, USA, 2002. ACM Press. 16, 49, 54, 73, 74

[Kalnins 03] Robert D. Kalnins, Philip L. Davidson, Lee Markosian & Adam Finkelstein. *Coherent stylized silhouettes*. ACM Transactions on Graphics, vol. 22, no. 3, pages 856–861, July 2003. 17

[Kaplan 05] Matthew Kaplan & Elaine Cohen. *A Generative Model for Dynamic Canvas Motion*. In Computational Aesthetics, 2005. 18, 51

[Karpenko 06] Olga A. Karpenko & John F. Hughes. *SmoothSketch: 3D free-form shapes from complex sketches*. ACM Transactions on Graphics, vol. 25/3, pages 589–598, 2006. 152

[Klein 00] Allison W. Klein, Wilmot W. Li, Michael M. Kazhdan, Wagner T. Correa, Adam Finkelstein & Thomas A. Funkhouser. *Non-Photorealistic Virtual Environments*. In Proceedings of ACM SIGGRAPH 2000, Computer Graphics Proceedings, Annual Conference Series, pages 527–534, July 2000. 22

[Knill 92] D. C. Knill. *The perception of surface contours and surface shape: from computation to psychophysics.* Journal of the Optical Society of America, vol. 9, no. 9, pages 1449–1464, 1992. 109

[Koenderink 82] J.J. Koenderink & A.J. van Doorn. *The Shape of Smooth Objects and the Way Contours End*. Perception, vol. 11, pages 129–137, 1982. 108

[Koenderink 84] J.J. Koenderink. *What does the occluding contour tell us about solid shape?* Perception, vol. 13(3), pages 321–330, 1984. 108

[Koenderink 96] Jan J. Koenderink, Andrea J. van Doorn, Chris Christou & Joseph S. Lappin. *Shape Constancy in Pictorial Relief.* In Object Representation in Computer Vision, pages 151–164, 1996. 108

[Koenderink 01] J. J. Koenderink, A. J. van Doorn, A. M. L. Kappers & J. T. Todd. *Ambiguity and the 'mental eye' in pictorial relief*. Perception, vol. 30, pages 431–448, 2001. 108

[Kolliopoulos 06] A. Kolliopoulos, J. M. Wang & A. Hertzmann. *Segmentation-Based 3D Artistic Rendering*. In Eurographics Symposium on Rendering 2006, 2006. 15, 118

[Kurlander 88] David Kurlander & Eric A. Bier. *Graphical search and replace*. In SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques, pages 113–120, New York, NY, USA, 1988. ACM Press. 74, 98

[Köppen. 00] M. Köppen., K. Franke & O. Unold. *A Survey on Fuzzy Morphology*. In Proc. 5th Intl. Conf. on Pattern Recognition and Image Analysis, pages 424–427, 2000. 118

[Lake 00] Adam Lake, Carl Marshall, Mark Harris & Marc Blackstein. *Stylized rendering techniques for scalable real-time 3D animation*. In Proceedings of the 1st international symposium on Non-photorealistic animation and rendering, pages 13–20. ACM Press, 2000. 13

[Levene 98] J. Levene. A framework for non-realistic projections. Master's thesis, MIT, 1998. 12

[Litwinowicz 97] Peter C. Litwinowicz. *Processing images and video for an impressionist effect*. In SIGGRAPH 97, pages 407–414, 1997. 14

[Lloyd 82] S. P. Lloyd. *Least squares quantization in PCM*. IEEE Trans. on Information Theory, vol. 28, no. 2, pages 129–137, 1982. 100

[Luft 05] T. Luft & O. Deussen. *Interactive Watercolor Animations*. In Pacific Graphics, 2005. 14

[Luft 06] Thomas Luft & Oliver Deussen. *Real-Time Watercolor Illustrations of Plants Using A Blurred Depth Test*. In NPAR 2006: Fourth International Symposium on Non Photorealistic Animation and Rendering, jun 2006. to appear. 14, 18, 45, 48

[Luo 00] M. R. Luo, G. Cui & B. Rigg. *The Development of the CIE 2000 Colour Difference Formula: CIEDE2000*. Colour and Imaging Inst., Univ. of Derby, 2000. 122, 133

[McCloud 94] Scott McCloud. Understanding comics. Harper, 1994. vi, vii, 22, 109, 144

[Meier 96] Barbara J. Meier. *Painterly rendering for animation*. In Siggraph'96, Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, pages 477–484. ACM Press, 1996. 17, 35, 36, 38, 40, 48, 60

[Nehab 04] Diego Nehab & Philip Shilane. *Stratified Point Sampling of 3D Models*. In Eurographics Symposium on Point-Based Graphics, pages 49–56, June 2004. 52

[Ni 06] Alex Ni, Kyuman Jeong, Seungyong Lee & Lee Markosian. *Multi-scale Line Drawings from 3D Meshes*. In 2006 ACM Symposium on Interactive 3D Graphics and Games, March 2006. 28

[Northrup 00] J. D. Northrup & Lee Markosian. *Artistic silhouettes: a hybrid approach*. In Proceedings of the 1st international symposium on Non-photorealistic animation and rendering, pages 31–37. ACM Press, 2000. 16, 17

[Ostromoukhov 99] Victor Ostromoukhov. *Digital facial engraving*. In Siggraph 99, Proceedings of the 26th annual conference on Computer graphics and interactive techniques, pages 417–424. ACM Press/Addison-Wesley Publishing Co., 1999. 50, 72

[Palmer 99] S. Palmer. Vision science : Photons to phenomenology. MIT Press, 1999. 94, 145, 146, 148

[Pastor 03] Oscar E. Meruvia Pastor, Bert Freudenberg & Thomas Strothotte. *Real-Time Animated Stippling*. IEEE Computer Graphics and Applications, vol. 23, no. 4, pages 62–68, 2003. 18, 41

[Perona 90] P. Perona & J. Malik. *Scale-space and edge detection using anisotropic diffusion*. Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 12, no. 7, pages 629–639, 1990. 118

[Portilla 00] Javier Portilla & Eero P. Simoncelli. *A Parametric Texture Model Based on Joint Statistics of Complex Wavelet Coefficients*. Int. J. Comp. Vision, vol. 40, no. 1, pages 49–70, 2000. 74, 110

[Praun 01] Emil Praun, Hugues Hoppe, Matthew Webb & Adam Finkelstein. *Real-time hatching*. In Siggraph 01, Proceedings of the 28th annual conference on Computer graphics and interactive techniques, page 581. ACM Press, 2001. 18, 23, 37, 49, 72, 103

[Preim 95] Bernhard Preim & Thomas Strothotte. *Tuning rendered line-drawings*. In WSCG'95, pages 228–238, February 1995. 73, 78

[Qian 94] Ning Qian, R. C. Anderson & Edward H. Adelson. *Transparent Motion Perception as Detection of Unbalanced Motion Signals III: Modeling*. The Journal of Neuroscience, vol. 14, no. 12, pages 7381–7392, 1994. 63

[Reinhard 01] Erik Reinhard, Michael Ashikhmin, Bruce Gooch & Peter Shirley. *Color Transfer between Images*. IEEE Comput. Graph. Appl., vol. 21, no. 5, pages 34–41, 2001. 117

[Romeny 94] Bart M. Romeny. Geometry-driven diffusion in computer vision. Kluwer Academic Publishers, Norwell, MA, USA, 1994. 110, 139, 146

[Rosin 94] P.L. Rosin. *Grouping curved lines*. In 5th British Machine Vision Conf, pages pp. 265–274, York, 1994. 78

[Rosin 98] Paul L. Rosin. *Determining local natural scales of curves.* Pattern Recognition Letters, vol. 19, no. 1, pages 63–75, 1998. 88

[Rost 04] Randi J. Rost. Opengl(r) shading language. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004. 54

[Rusinkiewicz 00] Szymon Rusinkiewicz & Marc Levoy. *QSplat: A Multiresolution Point Rendering System for Large Meshes*. In Proceedings of ACM SIGGRAPH 2000, pages 343–352, July 2000. 47

[Saito 90] Takafumi Saito & Tokiichiro Takahashi. *Comprehensible rendering of 3-D shapes*. In Siggraph 90, Proceedings of the 17th annual conference on Computer graphics and interactive techniques, pages 197–206. ACM Press, 1990. 38

[Salisbury 94] Michael P. Salisbury, Sean E. Anderson, Ronen Barzel & David H. Salesin. *Interactive pen-and-ink illustration*. In Siggraph 94, Proceedings of the 21st annual conference on Computer graphics and interactive techniques, pages 101–108. ACM Press, 1994. 72

[Salisbury 96] Mike Salisbury, Corin Anderson, Dani Lischinski & David H. Salesin. *Scale-dependent reproduction of pen-and-ink illustrations*. In Siggraph 96, Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, pages 461–468. ACM Press, 1996. 73

[Saund 94] Eric Saund & Thomas P. Moran. *A perceptually-supported sketch editor*. In UIST '94: Proceedings of the 7th annual ACM symposium on User interface software and technology, pages 175–184, New York, NY, USA, 1994. ACM Press. 73, 94

[Saund 03] Eric Saund. *Finding Perceptually Closed Paths in Sketches and Drawings*. IEEE Trans. Pattern Anal. Mach. Intell., vol. 25, no. 4, pages 475–491, 2003. 73, 78

[Schofield 93] Simon Schofield. *Non-photorealistic Rendering: A critical examination and proposed system*. PhD thesis, School of Art and Design, Middlesex University, 1993. vi, vii, 7, 14

[School 95] DK Art School. An introduction to art techniques. Dorling Kinsley limited, 1995. 68

[Seaborn 99] Matthew Seaborn, Lee Hepplewhite & John Stonham. *Fuzzy colour category map for the measurement of colour similarity and*

dissimilarity. Pattern Recognition, vol. 38, no. 2, pages 165–177, February 1999. 124, 134

[Shi 00] Jianbo Shi & Jitendra Malik. *Normalized Cuts and Image Segmentation*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, no. 8, pages 888–905, 2000. 118

[Sloan 01] Peter-Pike Sloan, William Martin, Amy Gooch & Bruce Gooch. *The Lit Sphere: A Model for Capturing NPR Shading from Art*. In Graphics Interface 2001, pages 143–150, 2001. 13

[S.M. 95] Wuerger S.M., Maloney L.T. & Krauskopf J. *Proximity Judgments in Color Space: Tests of a Euclidean Color Geometry*. Vision Research, vol. 35, no. 6, pages 827–835, 1995. 123

[Solso 94] Robert L. Solso. Cognition and the visual arts. MIT Press, Cambridge, MA, USA, 1994. 108, 115

[Sperl 04] Daniel Sperl. *Realtime Painterly Rendering for Animation*, 2004. 37

[Stevens 81] Kent A. Stevens. *The Visual Interpretation of Surface Contours*. Artif. Intell., vol. 17, no. 1-3, pages 47–73, 1981. 109

[Stoner 90] G. R. Stoner, T. D. Albright, & V. S. Ramachandran. *Transparency and coherence in human motion perception*. Nature, vol. 344, pages 153–155, 1990. 63

[Strothotte 02] Thomas Strothotte & Stefan Schlechtweg. Non-photorealistic computer graphics: Modeling, rendering and animation. Morgan Kaufmann, 2002. vi, 78

[Todd 90] J. T. Todd & F. D. Reichel. *Visual perception of smoothly curved surfaces from double-projected contour patterns*. J Exp Psychol Hum Percept Perform., vol. 16, no. 3, pages 665–674, 1990. 109

[Tomasi 98] C. Tomasi & R. Manduchi. *Bilateral Filtering for Gray and Color Images*. In ICCV '98: Proceedings of the Sixth International Conference on Computer Vision, page 839, Washington, DC, USA, 1998. IEEE Computer Society. 118

[Turk 01] Greg Turk. *Texture Synthesis on Surfaces*. In Proceedings of ACM SIGGRAPH 2001, pages 347–354, 2001. 75

[Vanderhaeghe 06] David Vanderhaeghe, Pascal Barla, Joëlle Thollot & François Sillion. *A dynamic drawing algorithm for interactive painterly rendering*. In Siggraph technical sketch: SIGGRAPH'2006. ACM, aug 2006. 35

[Wang 04a] Chung-Ming Wang & Yao-Hsien Huang. *A Novel Color Transfer Algorithm for Image Sequences*. J. Inf. Sci. Eng., vol. 20, no. 6, pages 1039–1056, 2004. 118

[Wang 04b] Jue Wang, Bo Thiesson, Yingqing Xu & Michael Cohen. *Image and Video Segmentation by Anisotropic Kernel Mean Shift*. In ECCV. Springer-Verlag, 2004. 118

[Wang 04c] Jue Wang, Yingqing Xu, Heung-Yeung Shum & Michael Cohen. *Video Tooning*. In ACM Transactions on Graphics (proc. of SIGGRAPH 04), 2004. 15, 118

[Watson 94] A.B. Watson & M. P. Eckert. *Motion-Contrast Sensitivity: Visibility of Motion Gradients of Various Spatial Frequencies.* Journal of the Optical Society of America, vol. 11, no. 2, pages 496–505, 1994. 63

[Wei 00] Li-Yi Wei & Marc Levoy. *Fast texture synthesis using tree-structured vector quantization*. In SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, pages 479–488, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co. 75

[Wei 01] Li-Yi Wei & Marc Levoy. *Texture Synthesis Over Arbitrary Manifold Surfaces*. In Proceedings of ACM SIGGRAPH 2001, pages 355–360, 2001. 75

[Willats 97] John Willats. Art and representation: New principles in the analysis of pictures. Princeton University Press, Princeton, NJ, USA, 1997. 394 pages. x, 8, 60, 143

[Willats 05] John Willats & Frédo Durand. *Defining Pictorial Style: Lessons from Linguistics and Computer Graphics*. Axiomathes, vol. 15, no. 2, 2005. x, 8, 60

[Wilson 04] Brett Wilson & Kwan-Liu Ma. *Representing Complexity in Computer-Generated Pen-and-Ink Illustrations*. In NPAR, 2004. 73, 78

[Winkenbach 94] Georges Winkenbach & David H. Salesin. *Siggraph 94, Computer-generated pen-and-ink illustration*. In Proceedings of the 21st annual conference on Computer graphics and interactive techniques, pages 91–100. ACM Press, 1994. 49, 72

[Winnemöller 06] Holger Winnemöller, Sven Olsen & Bruce Gooch. *Real-Time Video Abstraction*. In Proceedings of SIGGRAPH'06, 2006. 15, 118

[Wood 94] Phyllis Wood. Scientific illustration. Wiley, 1994. vi, vii, 3, 4, 22, 68, 69, 114

[Wood 97] Daniel N. Wood, Adam Finkelstein, John F. Hughes, Craig E. Thayer & David H. Salesin. *Multiperspective Panoramas for Cel Animation*. Proceedings of SIGGRAPH 97, pages 243–250, August 1997. ISBN 0-89791-896-7. Held in Los Angeles, California. 52

[Yoshizawa 05] Shin Yoshizawa, Alexander Belyaev & Hans-Peter Seidel. *Fast and robust detection of crest lines on meshes*. In SPM '05: Proceedings of the 2005 ACM symposium on Solid and physical modeling, 2005. 17

[Yu 04] Jingyi Yu & Leonard McMillan. *A Framework for Multiperspective Rendering*. In Eurographics Symposium on Rendering, Norrkoping, Sweden, 2004. 12

[Zander 04] Johannes Zander, Tobias Isenberg, Stefan Schlechtweg & Thomas Strothotte. *High Quality Hatching*. Computer Graphics Forum (Proceedings of Eurographics), vol. 23, no. 3, September 2004. 73

[Ziou 98] Djemel Ziou & Salvatore Tabbone. *Edge Detection Techniques - An Overview*. International Journal of Pattern Recognition and Image Analysis, vol. 8, pages 537–559, 1998. 78