CONTOUR-BASED IMAGES:
Representation, Creation and Manipulation

Alexandrina ORZAN

October 16, 2009

Abstract

This thesis proposes a novel image primitive — the diffusion curve. This primitive relies on the principle that images can be defined via their discontinuities, and concentrates image features along contours. The diffusion curve can be defined in vector graphics, as well as in raster graphics, to increase user control during the process of art creation.

The *vectorial diffusion curve* primitive augments the expressive powers of vector images by capturing complex spatial appearance behaviors. Diffusion curves represent a simple and easy-to-manipulate support for complex content representation and edition.

In *raster* images, diffusion curves define a higher level structural organization of the pixel image. This structure is used to create simplified or exaggerated representations of photographs in a way consistent with the original image content. Finally, a fully automatic vectorization method is presented, that converts raster diffusion curve to vector diffusion curve.

Contents

chapter 1

Introduction

The computer, as an art tool, has transformed traditional activities like painting, drawing and design, and has made possible new forms of art creation, such as algorithmic art[1] and net art[2]. As Anne Morgan Spalter remarks in the opening of her book "The Computer in the Visual Arts" [Spa98]:

> "with the advent of personal computer and the commodification of interactive graphics software [...] the computer became a valuable Postmodern art tool."

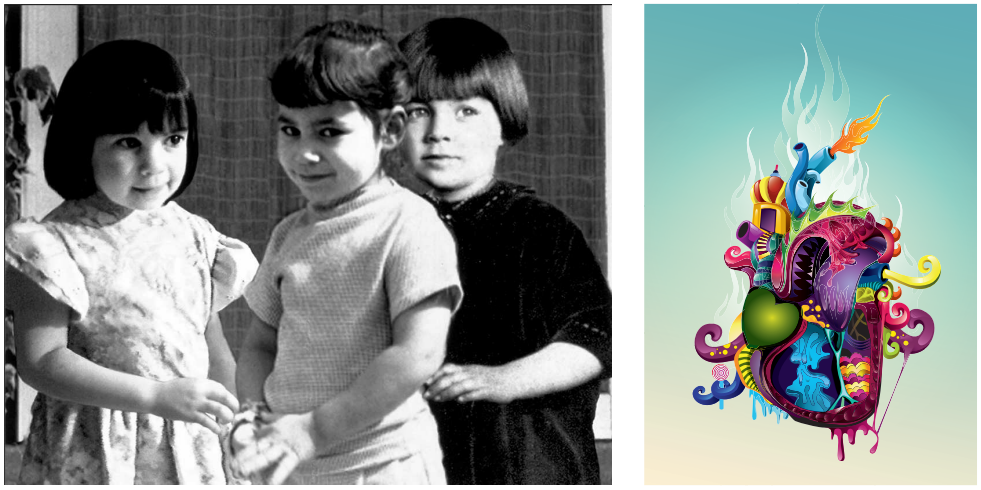
The computer has become a standard tool in many artistic processes because it introduces extraordinary flexibility to the act of creation. Two examples of art imagery made possible by the computer are shown in Figure 1.1. First is a creative photo-painting where the artist has brought together three generations of women in a single photograph. In the second image (Figure 1.1 (b)), the artist uses geometrically defined shapes to obtain exact lines and flawless curves for his design of a fantasy heart. These two images also illustrate the two distinct categories that can be used to represent a computer-generated artwork: raster graphics and vector graphics.

1 Digital images

Vector graphics is the creation of digital images through a sequence of geometrical primitives such as points, lines, curves, and polygons, all based on mathematical equations. *Raster graphics* provide an alternative representation for describing images. Rather than geometry, raster

[1] For example, San Base's dynamic painting uses computer algorithms to continuously "reinvent" itself: `http://www.sanbase.com/`

[2] Net art uses the internet as its medium and cannot be experienced in any other way. One example of net art is "My boyfriend came back from the war" by Olia Lialina:
`http://www.teleportacia.org/war/`



(a) (b)

Figure 1.1: *(a) Michele Turre,"Me, My Mom & My Girl at Three", 1992. This art work showcases four distinct phases of image making history: The 1920s studio portrait of her mother has the soft lighting, grainy film base, and soft printing style of that era. The picture of herself has the sharp focus look typical of the 1950s. The picture of her daughter was captured from video and still has scan lines running through it. She then combined these three photographic technologies in an image that speaks clearly of the computer age. (b) Dhanank Pambayun, "Living on a Heart Grunge", 2008. Here, the artist used the abstract geometric mark-making and the sharp, clean-cut look typical to vector graphics, to create a vintage looking fantasy.*

graphics use a grid of individual pixels to define images, where each pixel can store a different color.

Both representations have advantages and limitations. One important benefit of vector graphics is that images can be scaled to any size without any loss of quality, by simply multiplying their analytic description by a constant factor. As shown in Figure 1.2, vector pictures retain sharp features when magnified — a property called **resolution-independence**. In contrast, raster graphics are resolution dependent; they cannot scale to an arbitrary resolution without loss of apparent quality. When magnified, a raster image becomes grainy, and the eye can pick out individual pixels of uniform color (as shown in the zoomed-in version of the raster image in Figure 1.2). From an artist point of view, this means that a raster art can only be created, modified, and displayed at a single scale, while a vector art can be manipulated and exhibited at any scale.

Vector graphics are also easily **editable**. A vector object is a continuous mathematical description, and changes are made by modifying the mathematical formulae. Intuitive tools can be used to stretch, twist, and color objects in the picture. In raster images, on the other hand, there is no inherent relationship between any parts of the image; they are all just pixel values. Editing a raster image is not straightforward — changes of an object shape or color can only be made

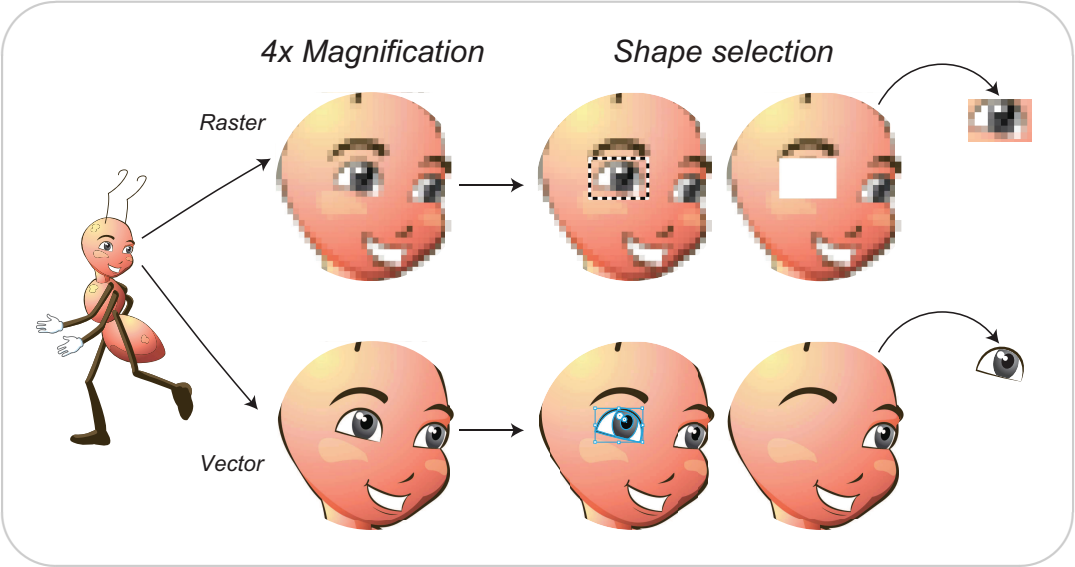


Figure 1.2: ***Raster graphics vs. Vector graphics:*** *This image demonstrates how vector (bottom row) and raster images (top row) behave when re-scaled (first column). It additionally illustrates the difference in shape selection for the two image formats (last column).*

by changing individual pixels. This is illustrated by the shape selection done in Figure 1.2. While raster selection simply moves a rectangle of color samples, in the vector selection the entire eye is dragged to a new location, and the face remains untouched.

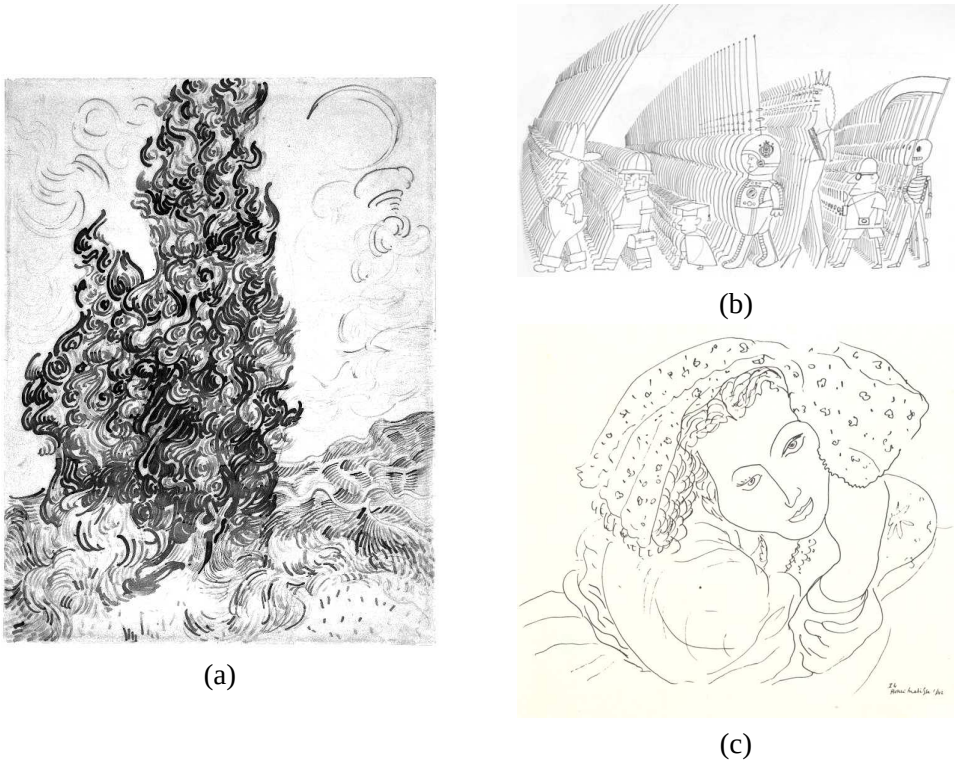
Vector-based images are also more easily **animated** than raster images, through keyframe animation of their underlying geometric primitives. Vector graphics thus provide the artists with an infinitely malleable and flexible image making tool.

However, for all of their benefits, vector-based drawing tools offer only limited support for representing complex image content. The complexity of object shapes and colors is limited to the vocabulary available for describing them and precludes many paint-type touches, such as creating complex shading by smudging colors. On the contrary, raster images can capture and represent **complex images**, and are typically used for the representation of photographs and photo-realistic images.

Raster and vector are thus complementary representations. While vector graphics have the *descriptive* power to specify semantically important image features, they cannot represent complex imagery. Raster graphics have the *expressive* power to depict photo-quality pictures, but image manipulation is extremely difficult.

2 Thesis

The thesis presented in this manuscript is that contour drawings are a useful tool for the depiction of image features, in particular color, shading and texture. This dissertation explains how a contour-based representation can be used in vector and raster graphics, and provides examples of powerful and intuitive user controls enabled by the use of contours. The proposed approach is inspired by the art of contour drawing and its capacity of expressing a subject in a few sketched lines.



(a) (b) (c)

Figure 1.3: ***The expressive power of contour drawing****: (a) Vincent van Gogh, "Cypresses", 1889. Circular, energetic, emotionally charged lines are used to convey the dramatic mood. (b) Saul Steinberg, "The Discovery of America", 1992 [Ste92]. The regular repetition of straight lines emphasizes firmness of purpose and dynamic motion. (c) Henri Matisse, "Variation 1", 1942. The curves in this drawing incorporate qualities like economy, sensuality, and elegance.*

2.1 Contour drawing

Drawing is the starting point of all visual art creations (Encyclopædia Britannica [EB09]). Painters, architects, sculptors, scientists, and film-makers alike rely on drawing to express their initial thoughts and to explore new possibilities for their designs [Kov06]. Every painting is

built essentially on its pre-sketched main contours, consolidated into colored surfaces [EB09].

A contour drawing is composed of lines and the empty space between the lines.

The principal element of drawing is the line, appearing as a border setting of bodies, colors and planes. Within the line composition, the space left blank fulfills an essential role: it conveys the uniform surface, the borders and nuances of which are indicated by the lines. The flat planes of a building, the unlined appearance of a cheek, the smooth width of a garment, the glassy surface of a lake, can all be encompassed by the empty spaces in the drawing [Kov06, EB09].

With the aid of this simple vocabulary, the viewer can be made to effortlessly identify the object of a drawing. The angular meeting of two lines, for example, may be considered as representing the borders of a plane; the addition of a third line can suggest the idea of a cubic body. Vaulting lines stand for arches, convergent lines for depth. The form of a line is enough for the human mind to call forth associations and "read" a complete scene, because "the visual world is made of contours, creases, scratches, marks, shadows, and shading" (Marr [Mar82]). Lines, therefore, can represent the way we perceive and understand the surrounding world [Mar82].

Everything real or imaginary can be represented through line drawing.

Line drawings, by their very simplicity, also offer a broad scope for the expression of artistic intention. Anything in the visible or imagined universe may be the theme of a drawing; bodies, space, depth, and even motion can be made visible through lines. In her book "Drawing for Dummies", Brenda Hoddinott [Hod03a] explains:

> "You can draw any object when you see its edges as simple lines. [...] Even complex subjects, such as people, can be rendered using only lines."

Furthermore, because of the rapidity with which it can be created, drawing captures in the flow of its line the personality of the artist, much as handwriting represents the writer's individual traits (Figure 1.3). Michael Craig-Martin [Kov06] describes the power of drawing thus:

> "Its characteristics include spontaneity, creative speculation, experimentation, directness, simplicity, abbreviation, expressiveness, immediacy, personal vision, technical diversity, modesty of means, rawness, fragmentation, discontinuity, and open-endedness. These have always been the characteristics of drawing."

Thesis: A digital image can be represented, created and edited via its contours.

Guided by the properties of line drawing and by the utilization of contours as a support for painting, this thesis exposes the possibility of using contour drawing to represent a colored digital image. Considering the role played by empty spaces — that of the smooth mass of a figure — we express the space between contours as smooth variations of color, shading and texture. Such a representation inherits the characteristics of drawing; it is effortlessly comprehended and easily created, for example, and it preserves the individuality of the artist's line style.

2.2 Contours as basic primitives for digital image creation

The following approach relies on the observation that traditional color painting starts with contour lines and proceeds by filling the outlined spaces with color and texture. Our work reposes on this classical framework to propose a general digital model, that is subsequently adapted to raster and vector imagery. This representation, called the *diffusion curve*, uses **contours** to define the shape of the depicted scene, and attaches **attributes** — arbitrary image features — to the contours. The space between the lines is filled with smoothly varying attributes by *diffusing* and mixing the values fixed along contours. To adapt this abstract model to the two digital image formats, we rely on the inherent characteristics of each representation.

2.2.1 Vector graphics

Vector graphics have an extraordinary *descriptive* power, offering fine-tuned control over form, color and placement. The diffusion curve vector primitives take advantage of this descriptive capacity to experiment with possible image attributes and design methods of creating a vector image from scratch. An important part of this dissertation is dedicated to the diffusion curve vector primitive and throughout this manuscript the generic term “diffusion curve” refers to the vector diffusion curve.

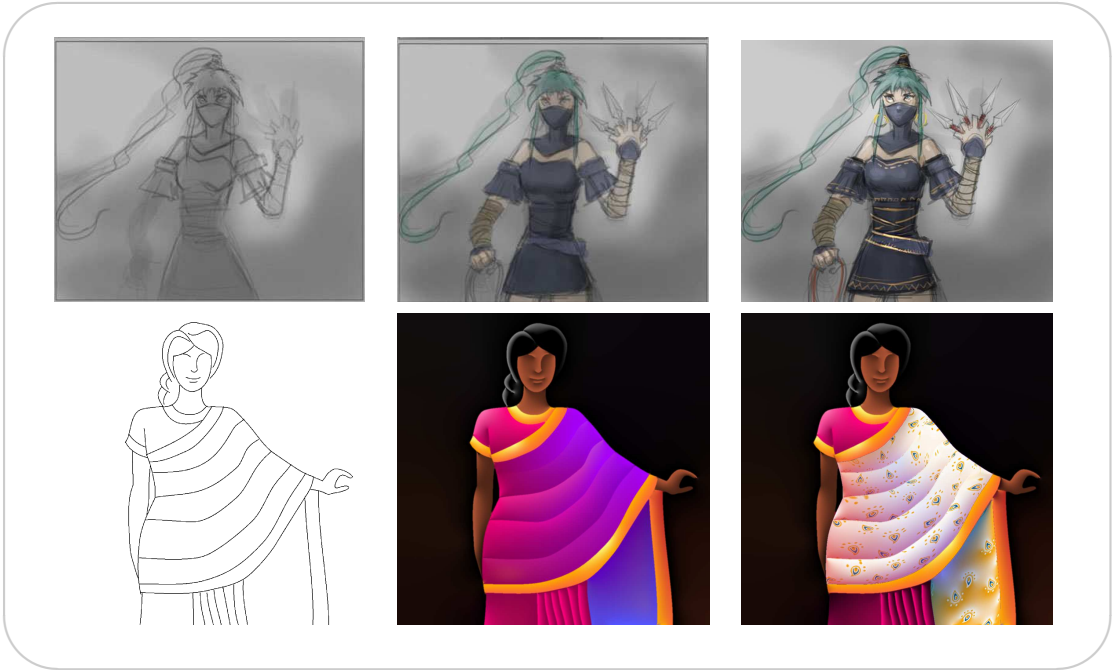


Figure 1.4: *Parallel between traditional media creation and the proposed vector graphics creation.* (Top) *Watercolor creation steps: contour drawing, color filling, and shading ©Xia Taptara.* (Bottom) *The diffusion curves steps: contour drawing, color setting, and finally texture and shading definition ©Laurence Boissieux.*

Creation with diffusion curves is easy to master by artists, because it allows for workflows close to the traditional "on-paper" art creation process [Hod03a]. Diffusion curvesenable the artist to start with a "blank paper", define contours and use them to develop color variations, shading effects and textured surfaces. A parallel between traditional media creation and the proposed vector graphics representation is shown in Figure 1.4.

2.2.2 Raster graphics

Raster graphics are especially notable for their power to faithfully represent photographs. In photographs, the entire scene is already "created", but lacks semantic information. The raster diffusion curve strives to represent raster images as a higher-level structure organization, that the user can manipulate more easily than pixels. In this structure, contours are made by edges — points in a raster image at which the image brightness changes sharply — with a color attribute.

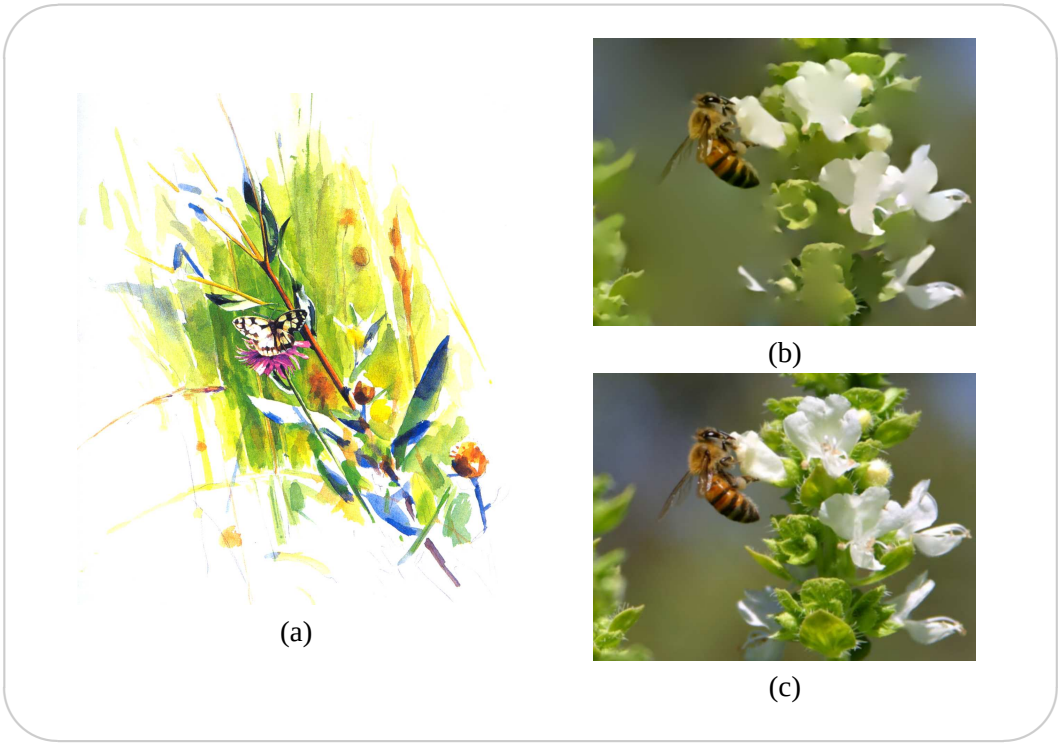


Figure 1.5: *(a) Example of hand-made image abstraction: "Le Papillon" (The Butterfly), watercolor by Eric Alibert. From "Leman, mon ile", © 2000 by Editions Slatkin. As seen in the guidebook of scientific illustration [Hod03b]. It is clear that focus of the image is the butterfly: it is depicted with many details, while plants around have little detail and their shape is less precise. Our raster diffusion curves allow the user to obtain similar effects (b) from a photograph (c).*

One example of raster manipulation is given in Figure 1.5. Figure 1.5 (a) represents a hand-made scientific illustration, from which it is clear that the main subject of the image is the

butterfly: it is depicted with many details, while plants around are only suggested. However, while abstracted, secondary elements of the image retain their look and are easily identified; in other words, their relevant structural components are preserved through the abstraction process. Similarly, our diffusion curve structural information guides user image manipulations, but preserves the relevant data (Figure 1.5 (b)).

2.3 Contributions

We propose a novel image primitive — the diffusion curve. This primitive relies on the principle that images can be defined via their discontinuities, and concentrates image features along contours. The diffusion curve can be defined in vector graphics, as well as in raster graphics, to increase user control during the process of art creation.

1. In particular, the *vectorial diffusion curve* primitive augments the expressive powers of vector images by capturing complex spatial appearance behaviors. Diffusion curves represent a simple and easy-to-manipulate support for complex content representation and edition.

   – First, the proposed vector primitive can depict complex color variations; represent lighting effects via user-defined normals; and natively handle textures.

   – Second, we provide powerful, high-level tools to intuitively manipulate the vector parameters. Using the diffusion curve principle that attributes vary smoothly everywhere except on contours, we allow the user to sparsely define image parameters, including colors, normals and texture coordinates, along curves of discontinuity. We additionally design editing methods that support common artistic workflows. Particularly, we describe methods for applying textures directly to a 2D image, without requiring full 3D information, a process we call *texture-draping*.

   – Third, based on a GPU-accelerated rendering we provide instant visual feedback and allow unhindered artistic manipulations.

2. In *raster* images, the present approach relies on edges to define diffusion curves as a higher-level structural organization of the pixel image. This structure is used to create simplified or exaggerated representations of photographs in a way consistent with the original image content.

3. Finally, a fully automatic vectorization method is presented, that converts raster diffusion curve to vector diffusion curve.

2.4 Organization

This manuscript explores primarily the capabilities of diffusion curves for vector graphics. Chapter 2 discusses the existent vector representations and their expressive capacity.

Chapter 3 introduces the diffusion curve vector primitive and describes how to efficiently render an image from such primitives.

Chapter 4 presents various options for creating and editing a vector diffusion curves image. It explains the process of creating and manipulating three image features: colors, shading and texture.

In Chapter 5, a raster-to-vector process is discussed, where image edges are transformed into raster diffusion curves and used to vectorize the image.

Chapter 6 revisits the raster diffusion curve primitive and explores how this structure can be used to enhance photographic representations.

Finally, the conclusion (Chapter 7) gathers our thoughts relative to the choices we have made and discusses future work possibilities.

chapter 2

Vector Graphics

This chapter presents a background on vector graphics, and details the distinguishing characteristics of different vector representations. Section 1 outlines the two vector system categories: stacking and planar maps. Vector primitives that fall into the stacking category are organized in Section 1.1. Each primitive is presented as a geometric shape onto which varied attributes can be attached; the shape definition, the attribute placement on the shape, and the types of possible attributes, are discussed in turn. Section 1.2 details the planar map representation and gives an overview on what constitutes a shape in planar maps and how attributes can be attached to each shape.

Possible attributes for vector primitives are color, shading and texture. Section 2 details each attribute and explains how manipulating the geometric shape influences the attribute values and positioning.

And finally, Section 3 considers automatic and semi-automatic methods of extracting vector primitives from raster images.

1 Representation

Vector primitives are geometric shapes with attached attributes of *color*, *shading* and *texture*. Depending on the way vector primitives interact with one another, vector illustrations fall in two categories: stacking and planar maps. The following sections present, for stacking and planar maps in turn, the possible primitives. For each primitive, three elements are discussed: the shape the primitive can have, the accepted attributes, and how these attributes can be attached to the shape.

1.1 Stacking

In a system that uses the stacking metaphor [Sut80], all shapes have a stacking order. A shape higher in the stack occludes the objects below it in regions of overlap. Since the objects do not interact other than to hide each other, each shape can be edited independently from the others. A simple example of this behavior is illustrated in Figure 2.1 (a) and (b).

Paths and geometric objects

Classical shapes in a stacking metaphor are either free-form *paths* or predefined geometric *objects*. A path is a sequence of Bézier curves, and is mostly used to create freehand art. Objects, on the other hand, rely on specific geometric definitions and can be created and edited in ways unique to their specified type. For example, in Adobe Illustrator© *CS4* and Inkscape© 0.45, the predefined objects are straight line segments, rectangles, ellipses, polygons and stars. An object is less "free" than a path, but has the advantage of dedicated tools based on geometric properties. An ellipse, for example, will thus be manipulated by increasing and decreasing its major and minor radius, and not by deforming individual Bézier curves.

Complex shapes can be obtained by combining two or more shapes using boolean operations, such as unions, intersections or differences. The Inkscape documentation [Ink08] presents a very good description of the different possibilities of creating paths and objects.

For both paths and objects, *attributes* can be attached to closed regions or to borders (as in Figure 2.1, where regions are filled with uniform color and borders are colored in a different hue). Open shapes are usually "closed" automatically by an invisible line segment uniting the extremities. The attributes defined for paths and objects are flat colors, linear and radial gradients and texture.

Gradient meshes

Recently, a different vector primitive was proposed for stacking systems: the gradient mesh (Adobe Illustrator©, Corel CorelDraw©).

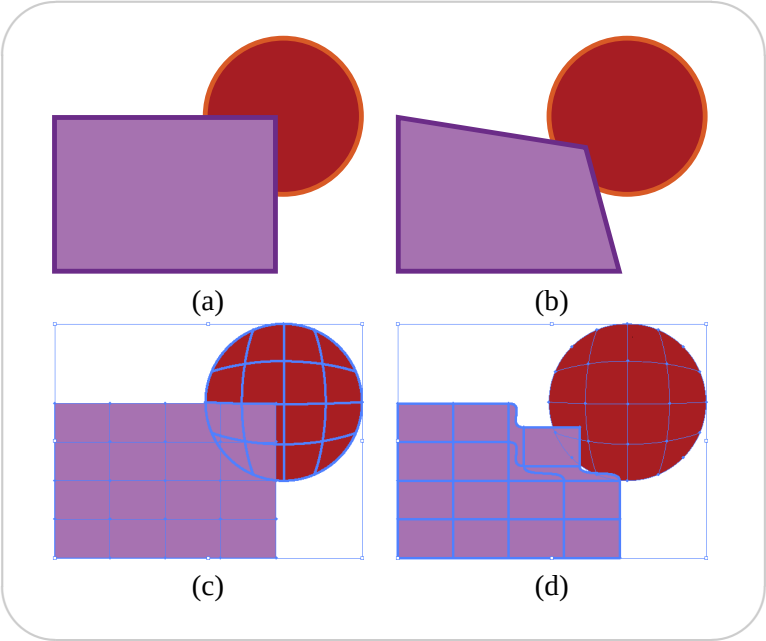


Figure 2.1: ***Vector shapes****: (a) A vector illustration in classical stacking shapes; (b) shows what happens when the foremost rectangle is modified. In (c), the same drawing is done with* gradient meshes. *The rectangle and the circle form each a gradient mesh object. (d) illustrates how the rectangle deforms when performing the same operation as in (b). Note that the mesh "knots" anchor the form, and thus only the top right patch deforms.*

A gradient mesh can be seen as a net placed over the object, made up of intersecting horizontal and vertical curves. Each 'knot" in the net — that is each point where the mesh intersects — anchors the object in place.

These *anchor points* can be pulled or adjusted to control the shape of the mesh, and the two intersecting curves are deformed accordingly. A comparison between the classical stacked shapes and gradient meshes is shown in Figure 2.1, where Figure 2.1 (c) illustrates the gradient mesh structures corresponding to the stacked shapes in Image 2.1 (a). Figure 2.1 (d) is the result of a gradient mesh deformation under the same user interaction as in Figure 2.1 (b).

Currently, the gradient mesh accepts only one *attribute* type: a color value. But a different value of color can be placed at each anchor point, and smoothly interpolated across mesh faces. Complex color variations can be created this way (Figure 2.2).

3D shapes

Yet another stacking primitive is the 3D shape, that can be created automatically from 2D artwork (Adobe Illustrator *CS*2). There are three ways to create a 3D object: extrude, bevel and revolve. In addition, a 2D or 3D object can also be rotated in three dimensions. Figure 2.3 illustrates these options.

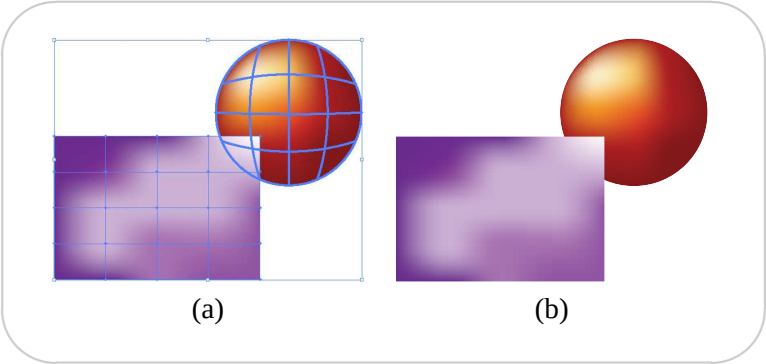


Figure 2.2: ***Complex color variations with gradient mesh:*** *(a) The mesh superposed onto the color filling. (b) The final vector drawing, with mesh nodes having varying colors.*

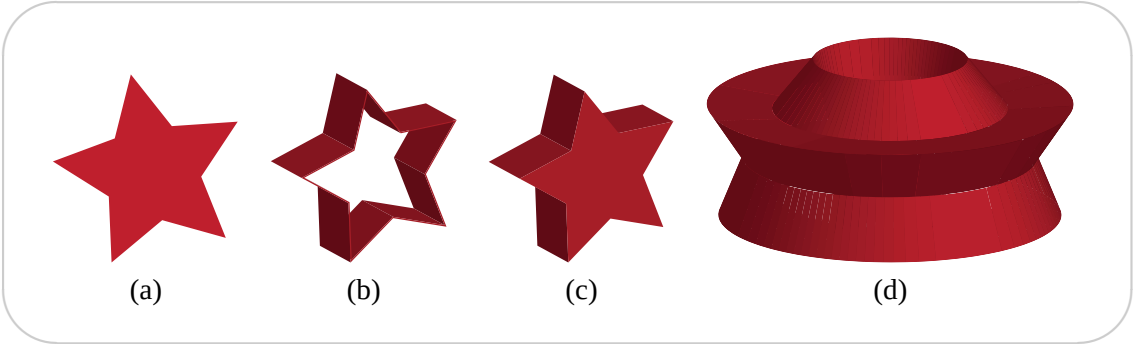


Figure 2.3: ***Possible 3D shapes from a 2D form:*** *(a) The original 2D object. (b) Extruded shape. (c) Beveled shape. (d) Revolved shape.*

For this vector primitive, *attributes* are placed on the 3D shape, and are automatically deformed by the 3D. Possible attributes are *shading* and *texture* (arbitrary 2D artwork). Note that in the term *shading* we include all possible illumination effects.

1.2 Planar maps

In a system that uses the planar map metaphor [BG89], all shapes are treated as though they are on the same flat surface. That is, none of the shapes is behind or in front of any other. Instead, the outlines divide the drawing surface into areas, derived from the intersection points of a line drawing.

For creating planar map drawings, the usual working process is to first create the line drawing. A planar map graph — the subdivision of the plane into nonoverlapping regions bounded by simple closed curves — is then automatically computed from the line drawing.

Attributes — of uniform colors, linear and radial color gradients, and texture — can be attached to any of the planar map regions, regardless of whether the area is bounded by a single shape

outline or by segments of multiple shapes. The result is that object painting is like filling in a coloring book or using watercolors to paint a pencil sketch. This important difference between planar maps and stacking is illustrated in Figure 2.4.

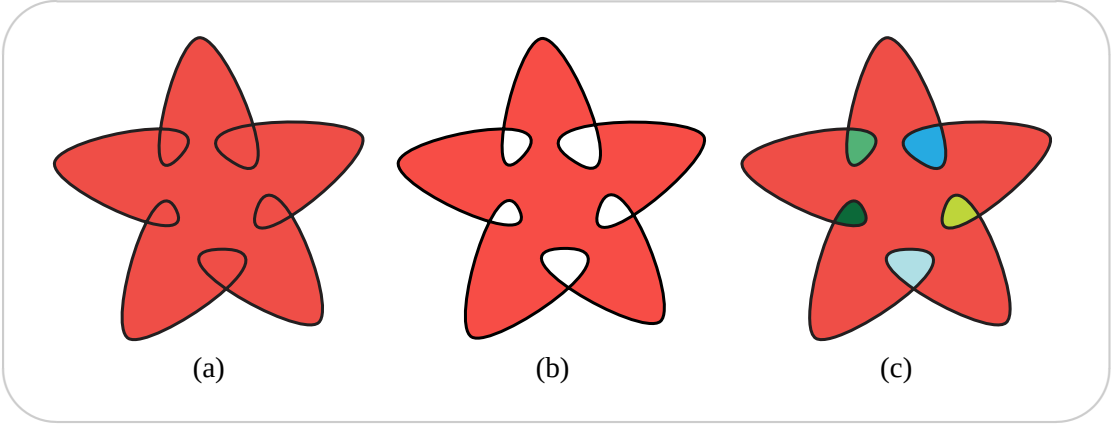


Figure 2.4: ***Color filling****: Example of different color filling behaviors for a closed self-intersecting path. (a) The stacking system in Adobe Illustrator© fills the entire region bounded by the path with a single primitive. (b) The Inkscape© stacking system detects holes in the path, and so removes them from region to be colored. (c) The planar map system proposed by Adobe Illustrator© can attribute one color primitive to each closed region, even if they are bounded by a single path.*

Planar map systems are especially useful in illustrations where the elements interact in a non-stacked way, such as weaves, knotwork, or linked rings. Planar maps are equally needed in illustrations that do not have any underlying structure, like hand-drawn cartoons (ToonBoom©, Adobe Flash©), and in illustrations where the regions to be filled are bounded by several unrelated paths.

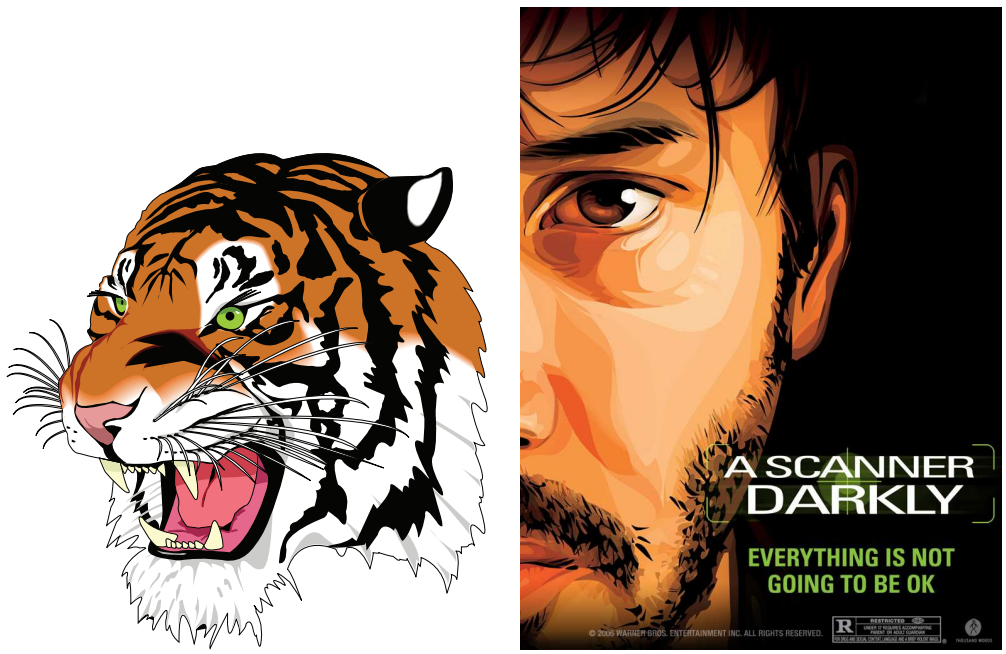
2 Creation and Manipulation

This section discusses in detail each vector attribute: color, shading and texture. For each attribute type, the creation tools that allow the user to specify various effects are presented. Also discussed is the influence shape manipulations have over the attribute values and positioning.

2.1 Shape and color

Stacking

The simplest way of coloring a vector drawing in stacking systems is by assigning one uniform color per region. This creates **flat-colored** results like those in Figure 2.5.



(a) (b)

Figure 2.5: ***Flat color in vector graphics****: Two examples of vector graphics realized with flat filling. (a) The impression of color variation around the nose is given by multiple superposed objects of increasingly lighter color. Image taken from the Inkscape© examples. (b) Flat colors can be used to obtain a strong visual effect. ©Warner Bros. Entertainment, Inc.*

While flat colors can create a very strong and suggestive effect in some cases (Figure 2.5 (b)), they are generally not sufficient to depict more natural-looking images, with highlights and smooth varying shadows. Figure 2.5 (a) shows how flat color regions are used to imitate a smooth varying color.

Gradient color primitives allow an artist to directly fill a region with gradations of color that blend smoothly into each other.

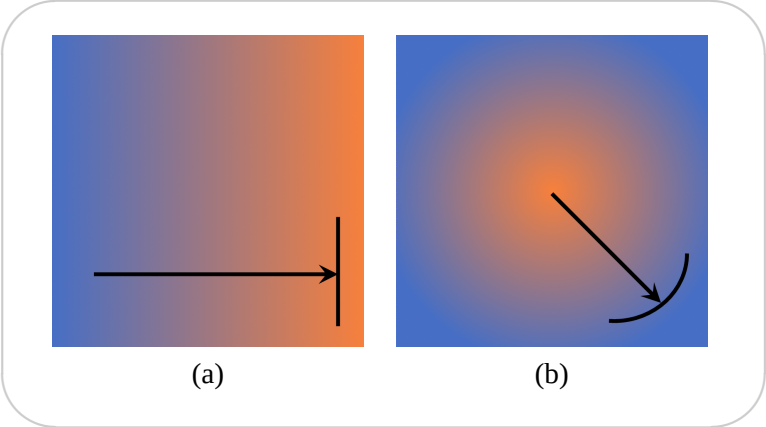


Figure 2.6: ***Simple gradients****: (a) Linear gradient. (b) Radial gradient.*

The most commonly used gradients are linear and radial (depicted in Figure 2.6). These gradients are generally suitable for traditional illustration and design work. In Figure 2.7, for example, a convincing effect of light coming from the upper right side is realized through the use of linear and radial gradients.

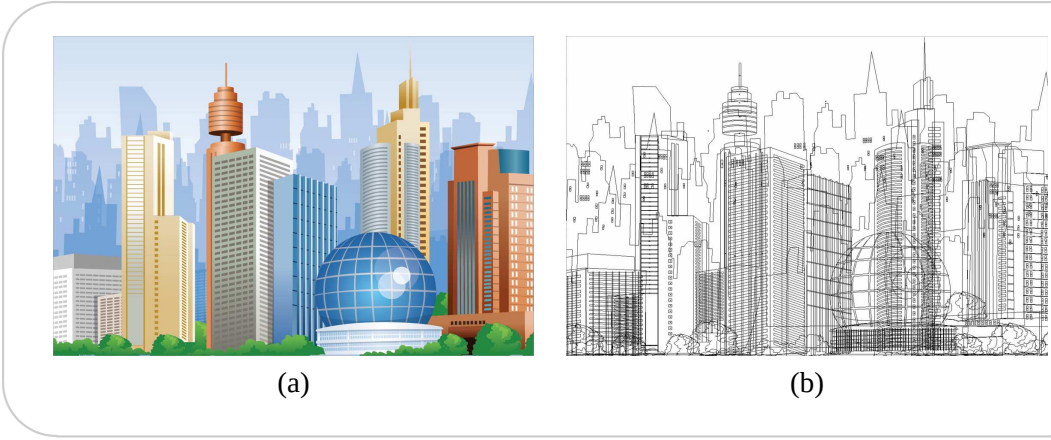


Figure 2.7: ***Light effect with linear and radial gradients****: (a) The final vector drawing. Image from* `www.freevectors.net`*. (b) shows the outlines of shapes used.*

But these simple gradients are limited from a creative standpoint, as it is difficult to create realistic images, paint-like styles, or complicated optical effects using only these types of color gradient. This is illustrated in Figure 2.8, an image taken from the Inkscape tutorial. Here, complex color variations are realized by the use of radial gradients alone; but this necessitated a great amount of primitives, making the result difficult and time consuming to create and edit.

Even with color variations inside closed regions, vector graphics are generally characterized by sharp color transitions between one colored region and another. The numerous cases where

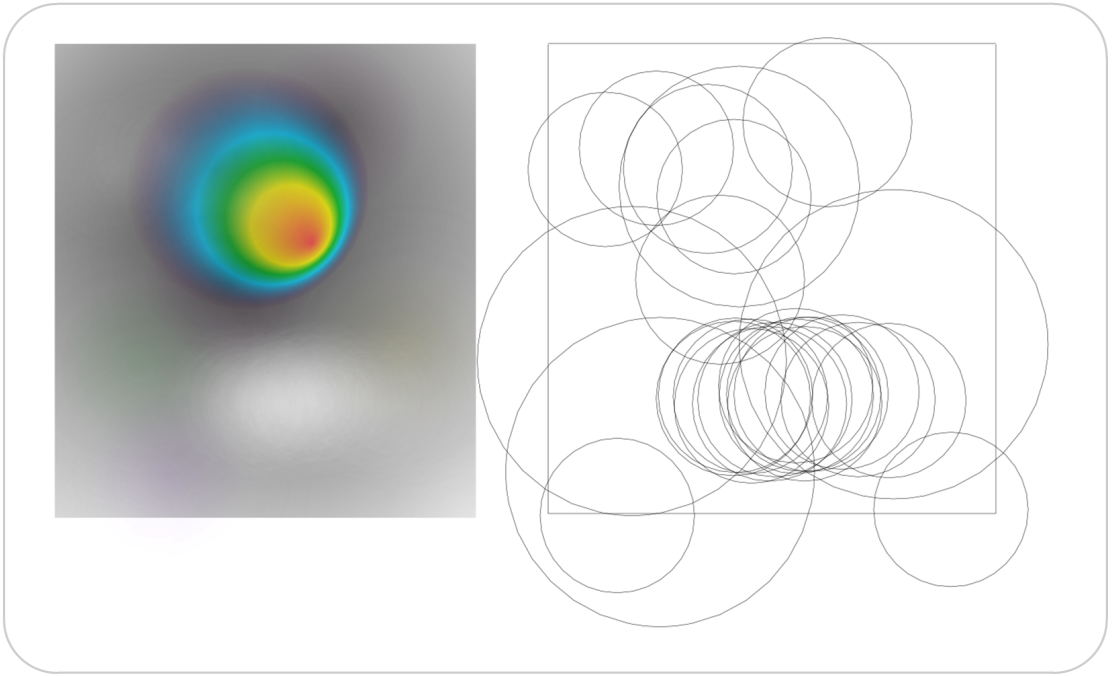


Figure 2.8: ***Complex color variations with simple gradients****: (a) The final vector drawing. (b) shows the outlines of shapes used. Note the complex arrangement and the numerosity of the employed primitives.*

region borders are not obvious — of which fog, smoke, and out of focus objects are just a few examples — are difficult to represent just with in-region gradients alone.

The classical solution for stacking systems is to add a **transparency** value to color attributes. The superposition of multiple regions of varying transparencies and different shapes can create an effect similar to brush strokes on canvas. An exquisite example of this is shown in Figure 2.9. However creating such complex images by blending region colors necessitates a great number of regions, and complicated, un-intuitive shapes.

Another solution to avoid sharp borders, adopted by the SVG format and modern tools (Adobe Illustrator©, Corel CorelDraw©, Inkscape©), is to reblur the image once vector primitives have been rasterized. However, they only allow for a uniform blur for each given primitive, which, similar to the limitations of flat colors or simple gradients, necessitates an impractically large number of primitives to approximate complex image content.

Gradient meshes have been specifically proposed to address these issues. However, manipulating color with gradient meshes is tightly linked to manipulating the mesh shape, because the only way of adding color variations is by introducing mesh anchor points. Creating a mesh drawing from scratch thus requires much skill and patience, because the artist needs to accurately anticipate the mesh resolution and topology necessary to embed the desired smooth features. This is why most users rely on an example bitmap to drive the design of realistic gradient meshes. The users first decompose an input photograph into several sub-objects and

Figure 2.9: ***Transparency effects:*** *Combining irregular shaped regions with gradients and transparency can create complex shading effects, as in this drawing by Jory Dayne ©Jory Dayne.*

then draw meshes over each sub-object following their topology; finally, they sample colors in the original photograph, assigning them to the mesh vertices. Many tutorials describing this approach and the mesh creation from scratch are available on the Web[1]. Still, drawing effective meshes and performing accurate manual color sampling is very time consuming in practice (several hours or even days for detailed images) and requires a good appreciation of the image complexity to adopt an adequate mesh resolution (Figure 2.10).

[1]Among gradient mesh tutorials available on the Web, these offer detailed descriptions for beginners: `http://www.learnit2.com/tutorial015/`, `http://www.magicalbutterfly.com/tutorials/meshtutorial/meshtutmain.htm`, `http://www.creativebush.com/tutorials/mesh_tutorial.php`

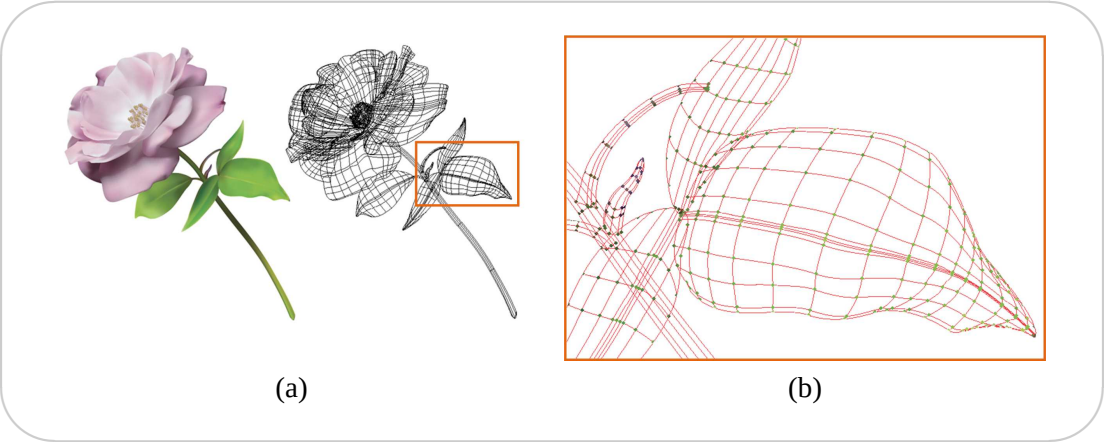


Figure 2.10: ***Gradient meshes complex example****: (a) An example of gradient mesh (©Adobe). (b) Zoom on the mesh, with all the color points marked. Note that to modify the color or the shape of the mesh, each point has to be individually edited.*

Diffusion curves achieve the same level of visual complexity as that reached by gradient meshes, but with a more direct workflow, better suited to artistic endeavors.

Planar maps

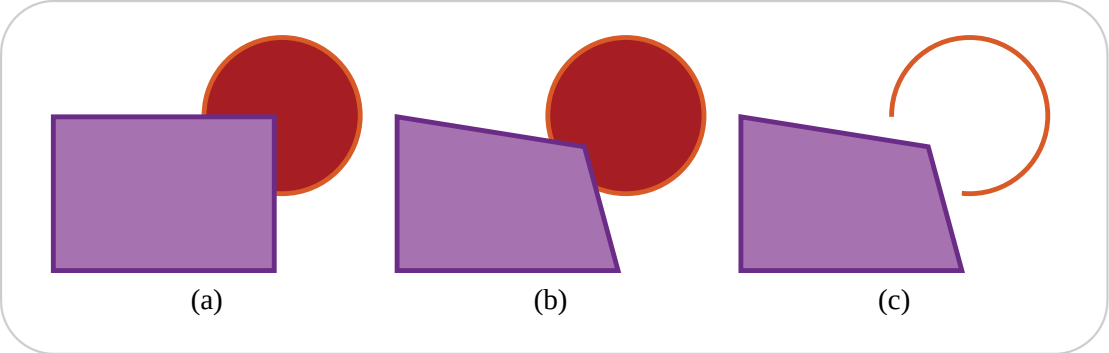


Figure 2.11: ***Planar map behavior*** *: (a) A vector illustration that looks the same in* stacking *and* planar map *systems. But the behavior of these two vector categories is different. This is visible in figures (b) and (c); (b) shows what happens when the foremost rectangle is modified in a* stacking *environment. Note that the background circle becomes more visible. (c) demonstrates the* planar map *behavior under the same operation. The red region, enclosed by the arc of a circle and the sharp corner, has been broken; no color can be attached to the corresponding area in the illustration.*

In planar map systems, free-hand line drawings are easily colored by assigning to each region in the planar map graph a separate color. As in stacking systems, planar map regions can be colored with flat colors, or with linear and radial gradient. However, while the illustrations are easy to color, they are difficult to edit. When lines are moved, planar maps are re-created, and

regions appear, disappear, split up or merge. It is difficult to decide how to transfer the colors from the previous planar map to the newly created planar map (Figure 2.11). Depending on the system, planar map computation can also split shapes at intersections, making them no longer editable as a whole (Figure 2.12 (a)).

Recently, Asente et al. [ASP07] proposed a new metaphor for planar-map editing that allows the artist to easily modify a composition after applying color. This system, called *LivePaint*, was also included in Adobe Illustrator©*CS2*. LivePaint proposes a set of editing rules that users would in most cases consider to be the right answer. However, in some cases the fill assignment is inherently ill posed and there is no obvious answer (an example is shown in Figure 2.12 (b)). Figure 2.12 (a) illustrates the LivePaint behavior compared with several other planar map systems.

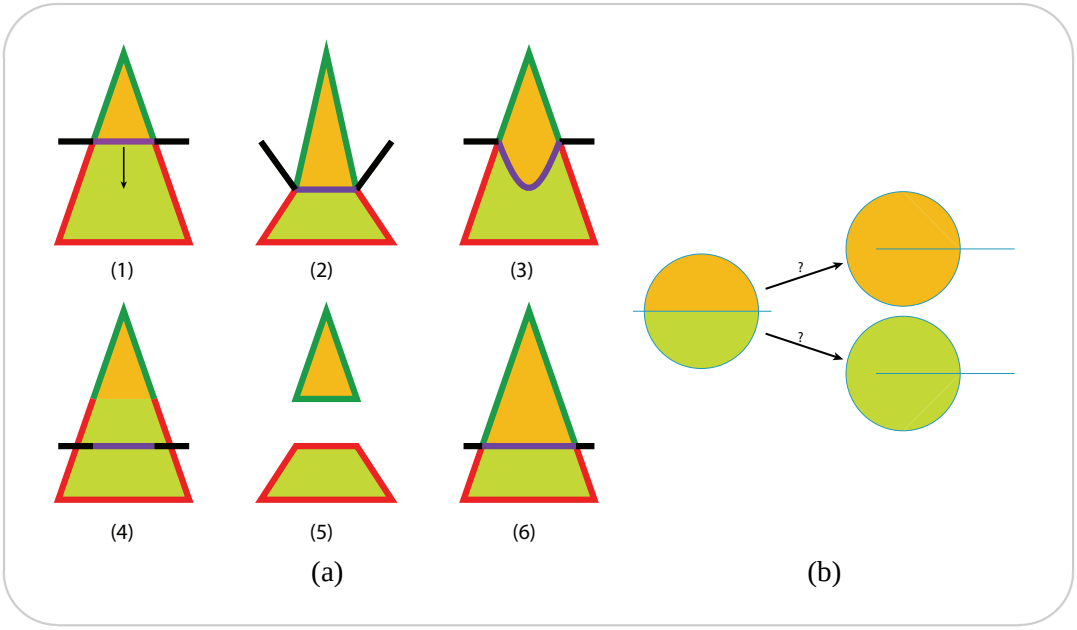


Figure 2.12: ***Planar map editing systems:*** (a) *Various results of editing. (1) Original illustration (2) MapSketch (3) Adobe Flash (4) Adobe Flash - a different behavior (5) Adobe Illustrator Pathfinder (6) Live Paint.* (b) *An ill-posed case for planar map systems. Images taken from Asente et al. [ASP07].*

The diffusion curve vector primitive we propose here is a planar map system in that there is no stacking order in our objects. However, the diffusion curve coloring capabilities surpass the simple linear or radial gradients possible in systems such as Live Trace. As we describe in Chapter 3, gradients in our representation can be arbitrary complex. Additionally, our color attributes, rather than being attached to regions, are attached to lines, thus avoiding the problem of color reassignment and making subsequent manipulation easier.

2.2 Shading

Shading is a very important part of image depiction. Through shadows and highlights, we infer the intended 3D form of the 2D artwork [Hod03a]. Figure 2.13, for example, shows how a flat-colored circle is transformed into a sphere by the judicious positioning of shades.

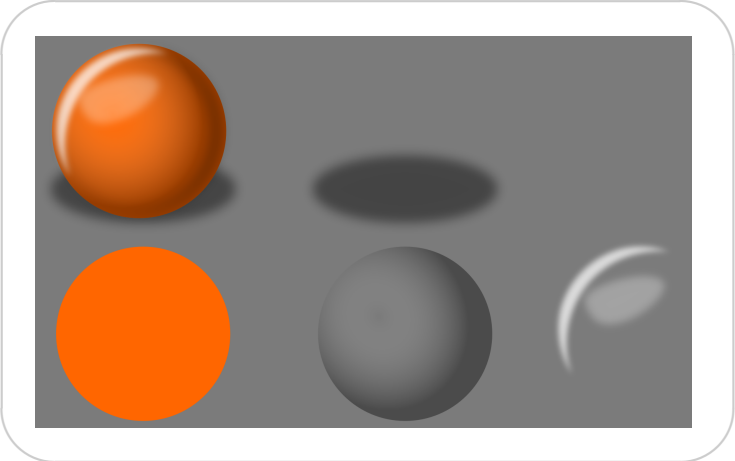


Figure 2.13: ***Sphere with gradient shading and shadow****: Top left: Completed sphere. Top middle: Shadow. Bottom: The sphere with shading and highlight layers. Image taken from the Inkscape tutorial [Ink08]*

In vector graphics, lighting environments and shading effects are usually suggested by color gradients and transparency (an "unwrapped"example is shown in Figure 2.13). As such, the vector graphics capacity of creating shadows and highlights depends of its power of representing color variations.

The *3D shape* stacking primitive provides a different alternative to color gradients: shading effects can then be automatically computed through classical 3D rendering algorithms. An example of these effects is given in Figure 2.14.

For *planar maps*, Johnston [Joh02] proposed a shading method that retains the hand-drawn aspect of the artwork. His approach approximates lighting in 2D drawings by inferring normals from the original line drawing. The key observation was that for curved surfaces, lines form the exterior silhouette and interior folds. Also, on silhouettes and folds, normals are perpendicular to the viewer. Normal vectors are therefore generated along the lines of the drawing by setting the normal value equal to the 2D gradient of the line. Normal values are then diffused in the empty space between lines, to create a sphere-like surface. An approximated shading is then automatically computed from the normals. The results obtained by this method are shown in Figure 2.15.

Diffusion curves follow the inspiration of Johnston, but they allow a higher degree of control on the initial normal values.

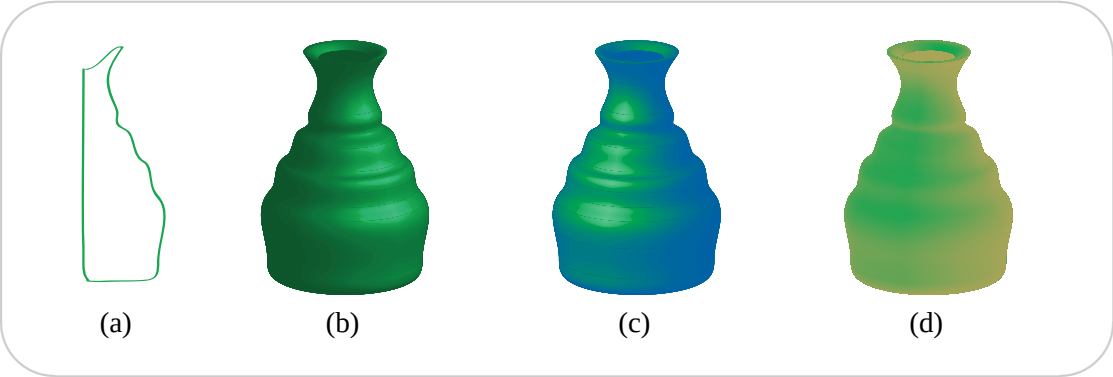


Figure 2.14: ***Shading effects for the 3D shape****: (a) Original 2D drawing. (b) 3D shape obtained by revolving the 2D artwork, with a plastic shading. (c) Plastic shading with different light position and a blue shading color. (d) Diffuse shading with yellow shading color.*

Figure 2.15: ***"Lumo" shading****: Results obtained by the normal-inferring technique presented in Johnston [Joh02]. From left to right: illumination obtained from the normals, original cat drawing, cat with added illumination, and final drawing. Image taken from [Joh02].*

2.3 Texture

Texture creation and manipulation tools originate from the need of controlling how texture is perceived by the viewers. In 2D pictures, textures transmit two important visual cues. The characteristics of a texture define the surface material of an object (wood, stone, etc.). And perceived texture distortions can be used to infer properties of 3D layout of objects and object shape [Jul62, BL91, RL93, LG04].

Considering these attributes of the texture, two different questions arise when creating textured vector drawings:

- How to use vector primitives to generate a texture based on user-specified input?
- How to include that texture pattern inside a vector primitive region, and deform the texture to suggest shape?

Definitions: To facilitate the discussion in the remainder of the section, we make use of the following definitions:

Texel - An atomic texture element, which is distributed repetitively over an image plane.

Texture-map - The planar image representation of a texture. This may be comprised of an arrangement of texels or generated otherwise.

Texture-draping - The process of applying a texture-map to a 2D image. Note, that this is not mere mechanical texture mapping, but rather the tools and techniques allowing the artist to specify texture-coordinates manually, e.g. to direct texture flow or suggest shape.

2.3.1 Creating a vector texture map

While vector images possess semantic capabilities to describe and parameterize textures, there is little work that addresses vectorial textures. Texture support in popular vector formats, such as SVG, CorelDraw©, Flash©, or Adobe Illustrator©, is commonly limited to tiling or pre-defined procedural textures.

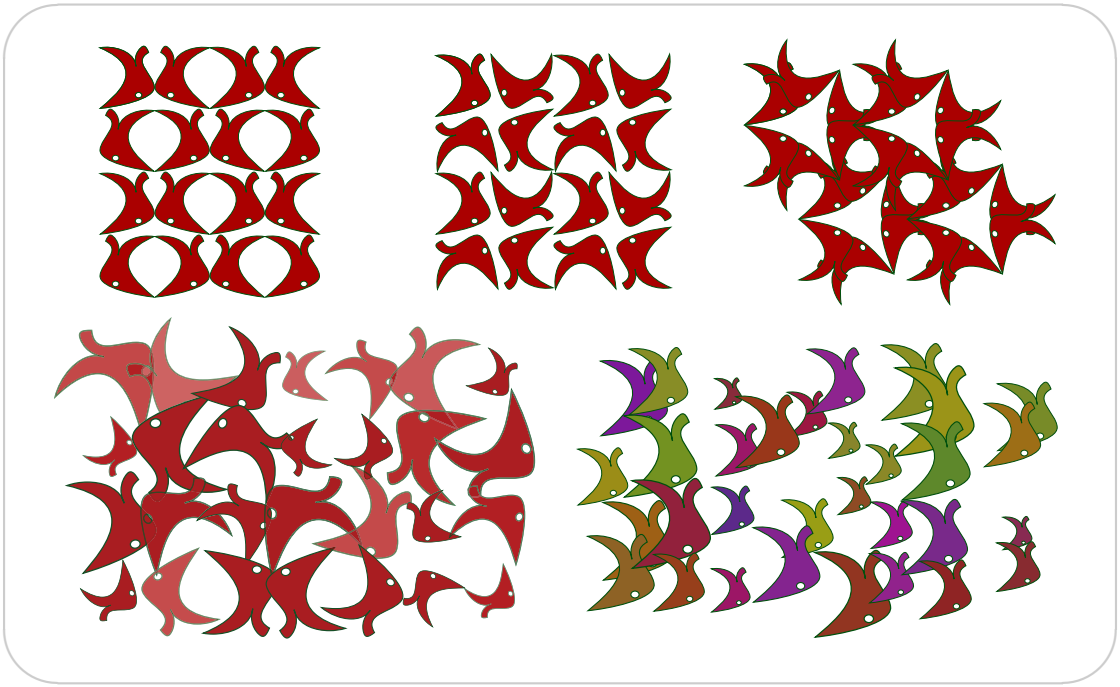


Figure 2.16: *Examples of texture maps produced with Inkscape. (top) Various symmetrical arrangements. (bottom) Randomization of rotation, scale, position and color.*

Tiling means that multiple copies of the sample are simply copied and pasted side by side. The tiling method is generally appropriate for regular, highly repetitive patterns, but doesn't

perform well for irregular patterns or stochastic detail[2]. Adobe Illustrator© proposes a regular grid as the pattern arrangement method, and a special tile element for the corners of the texture. Inkscape© considers all possible two-dimensional repetitive pattern, based on the symmetries in the pattern — the "wallpaper groups". Examples of patterns obtained with Inkscape are shown in the top row of Figure 2.16. For near-regular textures, Inkscape allows variations of position, rotation, size and color on its symmetrical arrangements (see the Tiling chapter of the Inkscape manual [Ink08]). The bottom row textures in Figure 2.16 use the randomization capabilities of Inkscape.

For creating stochastic or near-stochastic textures, **procedural** algorithms are generally used [PV95]. These algorithms generate random variations of a pre-defined type of texture, and rely on grammar definitions and function specifications to do so. Procedural textures create a realistic representation of natural elements such as wood [LP00], marble, granite, metal, stone and others. Unfortunately, they also require programming skills that make the creation of new procedural textures difficult. Procedural textures are also poorly adapted for creative endeavors, because individual artistic styles are not easy to translate into algorithmic representations.

The system of Barla et al. [BBT+06] addresses the learning of user-defined strokes and arrangements, to produce larger texture maps with qualitatively similar textures (Figure 2.17 (a). The proposed method extracts meaningful pattern elements from the user input and imitates their irregular distribution to synthesize new texture elements.

For creating a texture-map, we take advantage of the complex color gradients possible with diffusion curves to propose support for regular and near-regular textures with intricate designs.

2.3.2 Texture draping

In many vector graphics editors (such as Inkscape©), texture maps are directly rendered onto the image plane without much support to direct the arrangement of the texture in the image, for example, to suggest shape.

Tools for *2D scaling and rotation* of individual texels where proposed by Ijiri et al. [IMIM08]. By extending the work of Barla et al., the proposed method is able to synthesize texels along user-constrained paths and within user-defined regions, according to the learned distribution properties (Figure 2.17 (b)). In a similar way, Adobe Illustrator© *symbolism* tool allows the 2D placing of individual texels with the help of different spray tools (that can gather, scatter, shift, and spin texels).

The floral ornament paper by Wong et al. [WZS98] (Figure 2.17 (c)), as well as the Escherization system by Kaplan et al. [KS04a] (Figure 2.17 (d)), and Islamic pattern by Kaplan et al. [KS04b, Kap05] (Figure 2.17 (e)), consider both the shape of texels and their placement as a simultaneous problem.

[2] Structural definitions of textures are presented in Appendix B

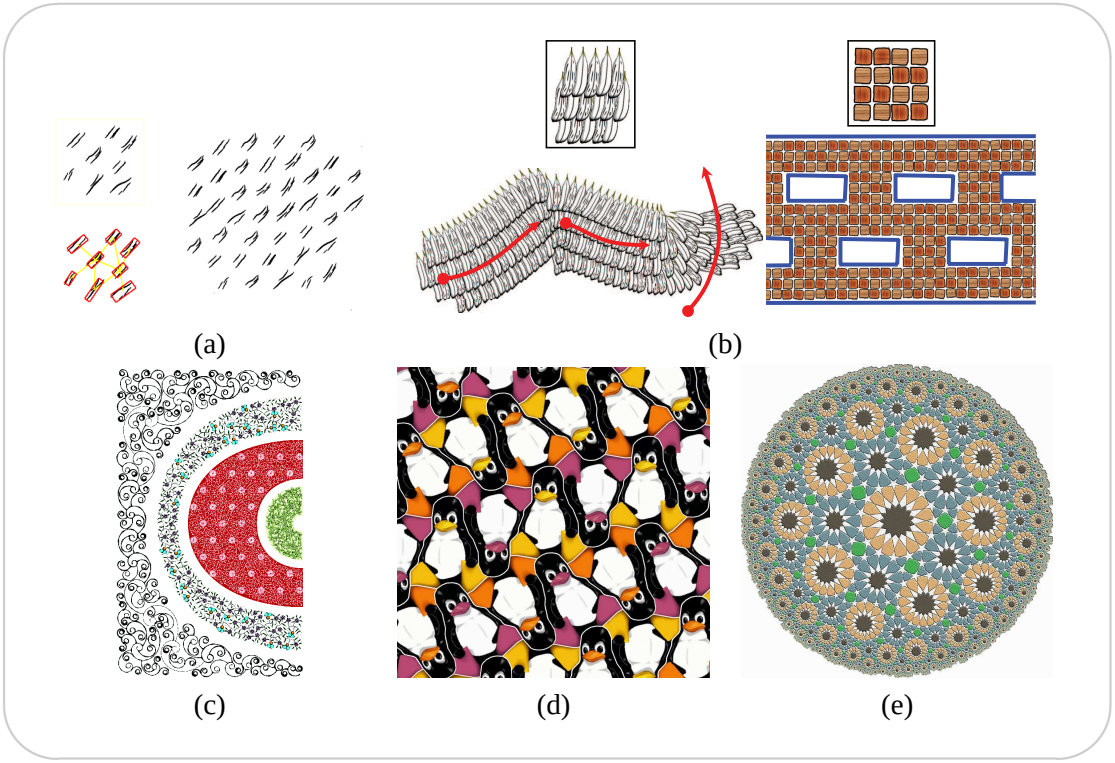


Figure 2.17: ***Vector texture synthesis****: Results from several vector pattern synthesis papers: (a) "Stroke pattern analysis and synthesis" [BBT+06]; (b) "An Example-based Procedural System for Element Arrangement" [IMIM08]; (c) "Computer-generated floral ornament" [WZS98]; (d) "Dihedral Escherization" [KS04a], (e) "Islamic star patterns in absolute geometry" [KS04b].*

Such texture syntheses are well suited to simulate flat texel distributions, but they do not address the problem of planar distribution manipulation to suggest surface shape.

Mesh-based warping can represent perspective deformations, as in the manual texture alignment of Liu et al. [LLH04]. However, the mesh topology tends to be unintuitive (not aligned with visual image features) and difficult to manipulate due to its complexity, despite promising results in partial automation [SLWS07].

The placement of 2D and 3D textures on the surfaces of *3D models* has been extensively studied [Hec86, SKvW+92, Lév01, GDHZ06]. Much of the work addresses atlas construction of 3D models or through-the-lens manipulation of texture coordinates on the projected models. While Illustrator© recently introduced 3D object creation from 2D shapes, a full 3D model that would capture the artist intent remains difficult to create. Folds and asymmetric shapes would be very problematic to create with the tools illustrated in Figure 2.18.

Diffusion curves enable draping texture maps directly onto images, without requiring full 3D information, to suggest shape and flow. The artist can control precisely how texture folds, ripples, and flows over an implied surface.

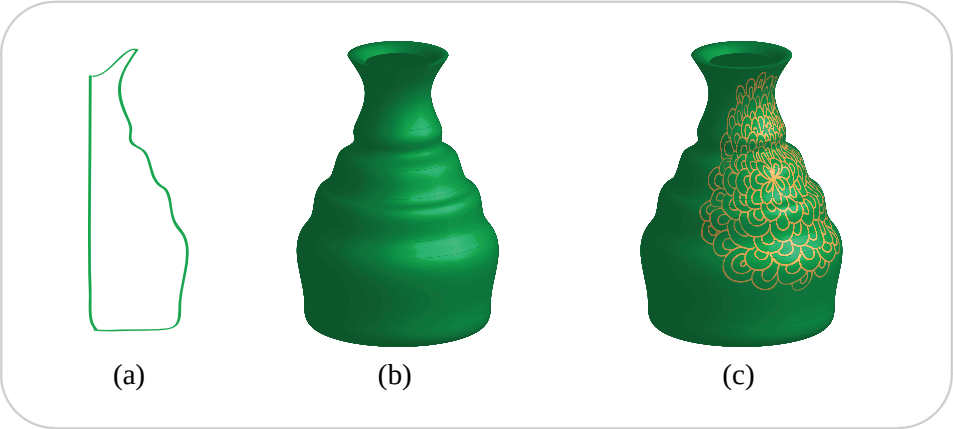


Figure 2.18: ***Adobe Illustrator© 3D texturing****: (a) Original 2D drawing. (b) 3D shape obtained by revolving the 2D artwork, with a plastic shading. (c) Artwork mapped on the revolved 3D shape.*

3 Vectorization

Aside from manually creating vector imagery, an interesting question is the extraction of vector primitives from raster images. Works that transform a bitmap into a vector representation have concentrated on representing color variations, and use for this either color flats, radial gradients, or gradient meshes. This section details the existing vectorization methods, organized according to the vector color primitive they use.

Commercial tools such as Adobe Live Trace© usually operate by segmenting an input image into regions of constant or slowly varying color, and by fitting polygons onto these primitives. Usually, an average *solid color* is assigned to each resulting vector primitive. Although this class of methods produces convincing results in uniform areas, the segmentation typically generates a prohibitive number of primitives in smooth regions.

The ArDeco system of Lecot and Lévy [LL06] allows vectorization of more complex gradients using *linear* or *quadratic gradient* primitives. It is based on a segmentation of the input image into regions of slowly varying color, and an approximation of color variations within each region with linear or quadratic gradients. The resulting primitives are fully compatible with the SVG standard, but the approximation tends to produce sharp color transitions between segmented regions (Figure 2.19).

Recently, the paper of Sun et al [SLWS07] proposed to assist the user by automatically fitting an input *gradient mesh* to an input image. The fitting is achieved by minimizing the reconstruction error between the resulting image and an input photograph. Their semi-automatic method greatly reduces the time required to draw a precise mesh and sampling colors, although the user still has to manually specify the sub-objects of the picture and draw the initial meshes with an adequate resolution. Price and Barrett [PB06] proposed a similar approach for creating a vector graphic image from an raster object, using recursive subdivisions until the reconstruction error falls below a fixed threshold. Their method produces faithful results but also generates

Figure 2.19: ***Ardeco [LL06] results****: (a) original image; (b) the vectorization result. Note that, while the gradient inside regions is well approximated, sharp transitions between regions are noticeable. Image taken from [LL06].*

many small patches in smooth regions. Lai et al. [LHM09] propose a fully automatic method of extracting gradient meshes from an image, that uses surface parametrization and fitting techniques.

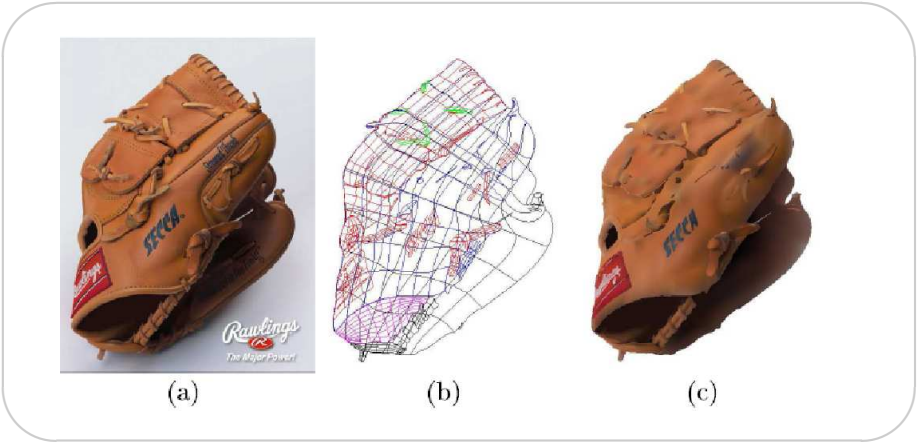


Figure 2.20: ***Price and Barrett [PB06] results****: (a) Raster image. (b) Resolution mesh created from (a). (c) Rendering of (b); colors indicate user-selected sub-objects. Images taken from [PB06] .*

Yet, with all three approaches, it remains unclear how to efficiently manipulate the resulting meshes for further editing. We believe this is due to the extra constraints imposed by the use of a mesh: using a predefined topology, employing specific patch subdivision schemes, and choosing a global orientation. In practice, this translates into a dense net of patches that are not readily connected to the depicted content. Hence, the manipulation of such a set of primitives quickly becomes prohibitive for the non-expert.

Diffusion curves allow a fully automatic extraction of color gradients that is comparable, in quality, with the semi-automatic approach of Sun et al [SLWS07].

chapter 3

Diffusion Curves Representation

This chapter concentrates on how to *represent* an image using diffusion curves. Section 1 discusses the pertinence of representing images by their discontinuities. Influential models of human vision, that identify discontinuities as an important part of the early-stage vision process, are presented. Additionally, a brief overview is given of computer vision findings that demonstrate that contours are relevant as an image representation.

The *diffusion curve* vector primitive is introduced in Section 2. The diffusion curve is defined by its geometric shape — a curve — and a set of attributes attached to the curve. Given a set of diffusion curves, the final image is obtained by solving a Poisson equation. Section 3 first defines the Poisson equation, and then describes the steps needed to transform the diffusion curve structure into the final image. Two methods are proposed: (1) a GPU-based implementation for rendering images defined by a set of diffusion curves in realtime (Section 3.2); (2) a mesh-based solution, that transforms the diffusion curves into sets of triangles, to integrate it into classical vector display systems (Section 3.3).

1 Representing images by their discontinuities

The human visual system is very sensitive to color and contrast variations [Pal99], and the early stages of vision rely on significant changes in the intensity to make explicit the structure of our surroundings [CR68, KD79].

Based on this finding, Marr [MH80] conjectured that an image may be completely represented by zero-crossing data (image edges) on multiple scales of the image. In his seminal book "Vision" [Mar82], he proposed a model of the visual system where three consecutive stages are used to transform the light falling on the retina into awareness of the visual world:

- The early stages of vision act like a *primal sketch*, and extract fundamental components of the scene (edges, regions, etc.), similarly to an artist quickly drawing a pencil sketch as a first impression.
- A second stage — called a 2.5D sketch — makes explicit the surface orientations with respect to the viewer and acknowledges textures. This stage is similar in concept to the stage in drawing where an artist highlights or shades areas of a scene, to provide depth and pattern.
- And lastly, a 3D model stage moves away from the viewer-centered surface description to construct a mental representation of shape and spatial organization in a continuous, 3-dimensional map.

Following Marr's insight, a number of subsequent mathematical models were proposed in order to algorithmically extract a primal sketch from a bitmap image. Lindeberg [Lin91, Lin93, Lin98] studied the notion of scale in image data, and considered image discontinuities at multiple scales for automatic solving of visual tasks (such as computation of surface shape or object recognition).

The works of Carlsson [Car88], Elder and Goldberg [Eld99] have demonstrated that image edges, augmented by color and blur information, constitute a near-complete and natural primitive for encoding images. Elder [EG01a] also suggested the possibility of using edges to efficiently manipulate images with basic operations (edge delete, copy and paste).

Recently, the algorithm proposed by Guo et al. [GZW03, GZW07] automatically separates textures (the "non-sketchable" part) from the "sketchable" part of a primal sketch (the structures). While structures are individually identified by a dictionary of "visual primitives" (such as edges), textured regions are automatically generated from a descriptive model (Markov Random Field model) [Jul62].

The diffusion curves representation is motivated by the primal sketch model, and by the recognized fact that most of the important features in the image are caused by, or can be modeled with edges; and that (possibly open) regions or patches are implicitly defined in between. But by a *vector representation* of edges and their attributes, diffusion curves greatly increase the manipulation capabilities suggested by Elder [EG01a], to include shape, color, contrast, and blur operations.

The diffusion curves are also not limited to color and blur, but can be leveraged to represent and edit other image properties as piece-wise-smooth data. Considering the model of Marr's 2.5D sketch, shading can be easily indicated through diffusion curves attributes, and textures can be arranged and manipulated in 2D images.

This approach provides the user with more intuitive and precise editing tools, and also supports resolution-independence, stylization and key-frame animation.

2 Data structure

The basic element of a diffusion curve is a geometric curve defined as a cubic Bézier spline (Figure 3.1 (a)) specified by a set of control points P. The geometry is augmented with additional attributes:

1. Two sets of color control points C_l and C_r (Figure 3.1(b)), corresponding to color constraints on the *right* and *left* half space of the curve;
2. A set of *blur* control points (Σ) that defines the smoothness of the color transition between the two halves (Figure 3.1(c)).
3. Two sets of *normal* control points N_l and N_r (Figure 3.1(e)), corresponding to normal constraints on each side of the curve;
4. Two sets of *(u,v) texture coordinates* control points U_l and U_r (Figure 3.1(g)).

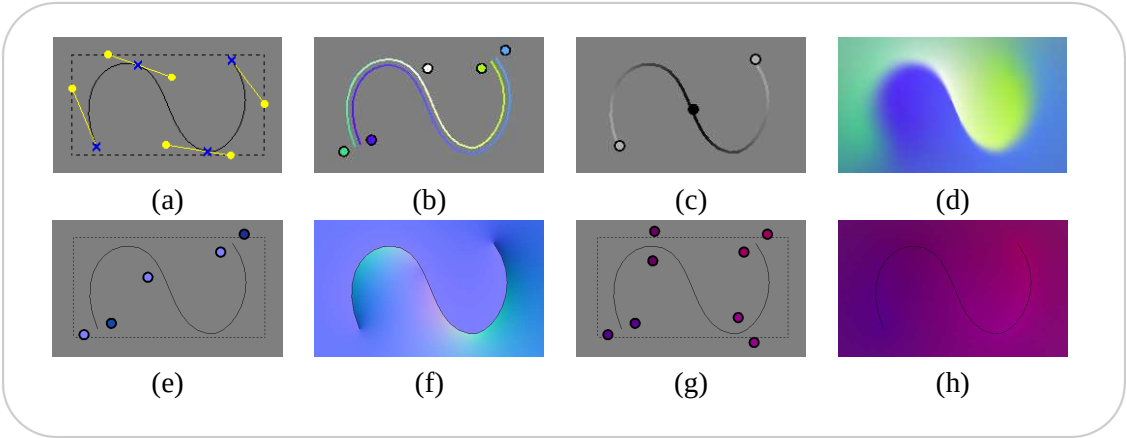


Figure 3.1: *A Diffusion curve: (a) A geometric curve described by a Bézier spline. (b) Arbitrary colors on either side, linearly interpolated along the curve. (c) A blur amount linearly interpolated along the curve. The final* color *image (d) is obtained by diffusion and reblurring. Note the complex color distribution and blur variation defined with a handful of controls. (e) Arbitrary normal attributes on either side of the curve are diffused to obtain a complete* normal map *(f). (g) Left- and right-side* (u,v) *coordinates create a* (u,v) map.

These attributes model three distinct steps of image creation. (1) The curves diffuse color on each side with a soft transition across the curve given by its blur (Figure 3.1(d)) to create the color image. (2) Normals are interpolated and diffused to create a normal map (Figure 3.1(f)), used to integrate shading into the vector drawing. (3) (u,v) coordinates are equally diffused to create a (u,v) map (Figure 3.1(g)); together with the normals, this map is used to drape the texture-maps in the image.

Color and blur attributes can vary along a curve to create rich color transitions. This variation is guided by an interpolation between the attribute control points in attribute space. In practice, we use linear interpolation and consider colors in RGB space throughout the rendering process (Section 3), because they map more easily onto an efficient GPU implementation and proved to be sufficient for the artists using our system.

Normal attributes also vary along the curve, but their variation can be either a linear interpolation, or it can rely on the curve geometry to provide the x and y values (see Section 2). The (u,v) texture control points, just as all the other attributes, can independently be placed on either side of the curve, and their values are linearly interpolated along the curve.

Control points for geometry and attributes are stored independently, since they are generally uncorrelated. This leads to eight independent arrays in which the control points (geometry and attribute values) are stored together with their respective parametric position t along the curve:

DiffusionCurve:		
	$\mathrm{P}[n_{pos}]$	– array of $(x,y,$ t$)$;
	$C_l[n_{cl}]$	– array of (r,g,b,t);
	$C_r[n_{cr}]$	– array of (r,g,b,t);
	$\Sigma[n_\sigma]$	– array of (σ,t);
	$N_l[n_{nl}]$	– array of (x,y,z,t);
	$N_r[n_{nr}]$	– array of (x,y,z,t);
	$U_l[n_{ul}]$	– array of (u,v,t);
	$U_r[n_{ur}]$	– array of (u,v,t);

The user can optionally decide to deactivate any of the diffusion curve parameters. This allows for a variety of applications ranging from a full vector drawing created from scratch (all parameters are active) to replacing textures in an existing bitmap image (only normals and u,v coordinates are used), as shown in Chapters 4 and 5.

3 Rendering

Given a set of diffusion curves, the empty space between curves is filled in by solving a Poisson equation whose constraints are specified by the attributes across all diffusion curves. This section first defines the Poisson equation, and then describes the steps needed to pass from the diffusion curve structure to the final image. Two methods are proposed: (1) a GPU-based implementation for rendering images defined by a set of diffusion curves in realtime; (2) a

mesh-based solution, that transforms the diffusion curves into sets of triangles, to integrate it into classical vector systems.

3.1 Poisson equation

The mathematical tool at the heart of our approach is the Poisson partial differential equation with Dirichlet boundary conditions:

$$\Delta f = g \text{ over } \Omega \subset \Re^n, \text{ with } f(m) = d(m) \quad \forall m \in \partial\Omega$$

where Δ is the Laplace operator, f is an unknown function defined over the closed domain Ω, g is a known function defined over Ω, and d is a known function defined over the boundary $\partial\Omega$. In image processing techniques, g is generally considered as the divergence of a vector field $\mathrm{div}\,\mathbf{v}$ [PGB03].

The Laplace operator measures the "smoothness" of a function and is defined as the divergence of the gradient ($\Delta f = \mathrm{div}\nabla \mathrm{f}$). The divergence of a vector field indicates "how fast" a flow following the vector field expands or compresses. Intuitively, the Poisson equation computes the function f that smoothly interpolates the boundary conditions d, while following the local variations imposed by the vector field $\mathbf{v}$ as closely as possible.

In $2D$, the Laplace and divergence operators are given by:

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \text{ and } \mathrm{div}\,\mathbf{v} = \frac{\partial \mathbf{v_x}}{\partial \mathrm{x}} + \frac{\partial \mathbf{v_y}}{\partial \mathrm{y}},$$

for $\mathbf{v} = (\mathbf{v_x}, \mathbf{v_y})$ and (x,y) the standard Cartesian coordinates of the plane.

Using diffusion curves, the attribute values at any point in the image domain are given by the function $f(x,y)$, obtained while the Dirichlet conditions are the attribute values stored along the diffusion curves. This is slightly different from the classical approach, where the Dirichlet boundaries are only the outlines of the image. If we consider the color attribute, for example, and the domain Ω as the image domain, then the solution to the Poisson equation passes through all color constraints along the curve and interpolates as smoothly as possible between them. Colors depart from this smooth interpolation only if the vector field $\mathbf{v}$ is not constant ($\mathrm{div}\,\mathbf{v} \neq 0$). The question is then what variations should be imposed to the colors? Following the diffusion curve principle, the image is represented by positioning diffusion curves at its discontinuities, and is smooth everywhere else. Variations should only be expected across a diffusion curve, between its left and right colors. The vector field $\mathbf{v}$ is zero everywhere except on the curve, where it indicates the color derivative across the curve. For the diffusion curves, therefore, $\mathbf{v}$ expresses the gradient of the attribute field, noted $\mathbf{w}$.

For

$$\begin{aligned} DC &= \{(x,y) \in \Omega \mid (x,y) \text{ on a curve from the diffusion curves set}\} \\ \partial DC &= \{(x,y) \in \Omega \mid (x,y) \text{ stores a left or right color constraint imposed} \\ &\qquad\qquad \text{by the diffusion curves set}\}, \text{ and} \\ C &= \text{ the Dirichlet color constraints over } \partial DC, \end{aligned}$$

the Poisson equation used to compute the color image I becomes:

$$\frac{\partial^2 I}{\partial x^2}+\frac{\partial^2 I}{\partial y^2}=\begin{cases} 0, & \text{if } (x,y)\notin DC \\ \frac{\partial \mathbf{w_x}}{\partial x}+\frac{\partial \mathbf{w_y}}{\partial y}, & \text{otherwise.} \end{cases} \qquad (3.1)$$
$$I(x,y)=C(x,y) \text{ if } (x,y)\in \partial DC,$$

where the gradient $\mathbf{w}=(\mathbf{w_x},\mathbf{w_y})$ indicates the direction and the magnitude of the greatest rate of increase in color values across a curve. Blur, normals and (u,v) texture coordinates are specified by equations similar to the color Poisson equation.

To compute the final attribute values for the entire image space Ω, we discretize and solve the Poisson equation in two different ways: a raster-based grid and a mesh-based discretization. In the interest of clarity, we describe the rasterization and the diffusion process for colors, because it is more easily visualized. The normals and (u,v) attributes follow the same process as the one necessary to obtain a sharp color image (Figure 3.2).

3.2 Raster-based diffusion

Three main steps are involved in our color raster-based rendering model (see Figure 3.2): (1) rasterization of a *color sources* image, where color constraints are represented as colored curves on both sides of each Bézier spline, and the rest of the pixels are uncolored; (2) *diffusion* of the color sources — an iterative process that spreads the colors over the image; we implement the diffusion on the GPU to maintain realtime performance; and (3) *reblurring* of the resulting image with a spatially varying blur guided by the blur attributes. Technical details about these three steps are explained in the following paragraphs.

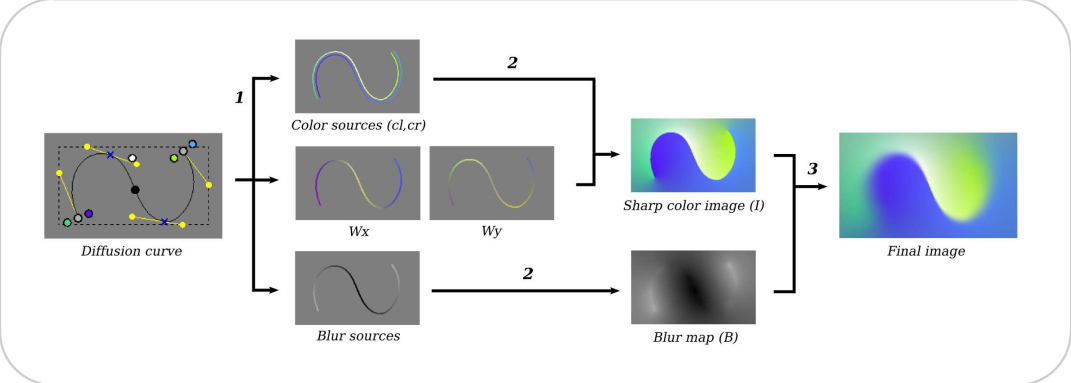


Figure 3.2: *Rendering diffusion curves requires (1) the rasterization of the color and blur sources, along with the gradient field* $\mathbf{w}=(\mathbf{w}_x,\mathbf{w}_y)$*, (2) the diffusion of colors and blur, and (3) the reblurring of the color image.*

3.2.1 Color sources

A diffusion curve has two sets of color control points, which are linearly interpolated along the curve. Using the interpolated color values, the first rasterization step renders the left and right color sources $c_l(t), c_r(t)$ for every pixel along the curve. An α mask is computed along with the rendering to indicate the exact location of color sources versus undefined areas (1 if the pixel contains a color source, 0 otherwise).

For perfectly sharp curves, these color sources are theoretically infinitely close (Figure 3.3 (a)). However, rasterizing pixels separated by too small a distance on a discrete pixel grid leads to overlapping pixels. In our case, this means that several color sources are drawn at the same location, creating visual artifacts after the diffusion (Figure 3.3 (b)). Our solution is to distance the color sources from the curve slightly, and to use a color gradient constraint directly on the curve. The gradient maintains the sharp color transition, while the colors, placed at a small distance d in the direction normal to the curve, remain separate (Figure 3.3 (c)).

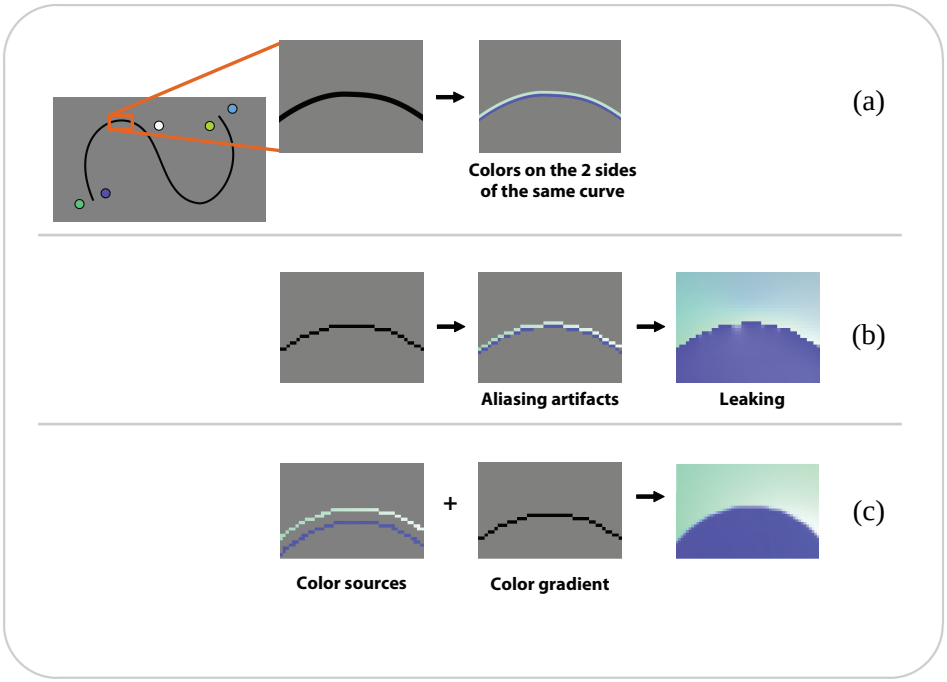


Figure 3.3: *From a vectorial curve to a pixel grid: (a) In a continuous space, color sources are infinitely close. (b) At rasterization time, considering the left and right color sources next to one another leads to overlaps, and subsequently to leaking diffusion artifacts. (c) Distancing to color constraints and using the gradient for sharp transitions creates the correct result.*

More precisely, the gradient constraint is expressed by the gradient field $\mathbf{w}$ defined in Section 3.1, which is zero everywhere except on the curve, where it is equal to the color derivative across the curve. We decompose the gradient field in a gradient along the x direction $\mathbf{w}_x$ and a gradient along the y direction $\mathbf{w}_y$. For each pixel on the curve, we use the finite difference method to compute the color derivative across the curve from the curve normal N and the left (c_l) and right (c_r) colors (we omit the t parameter for clarity): $\mathbf{w}_{x,y} = (c_l - c_r)N_{x,y}$.

We rasterize the color and gradient constraints in 3 RGB images: an image C containing colored pixels on each side of the curves, and two images W_x, W_y containing the gradient field components (Figure 3.2, step (1)). In practice, the gradient field is rasterized along the curves using lines of one pixel width. Color sources are rasterized using triangle strips of width $2d$ with a special pixel shader that only draws pixels that are at the correct distance d. In our implementation d is set at 3 pixels. Pixel overlap can still occur along a curve in regions of high curvature (where the triangle strip overlaps itself) or when two curves are too close to each other (as with thin structures or intersections). A simple stencil test allows us to discard overlapping color sources before they are drawn, which implies that solely the gradient field $\mathbf{w}$ dictates the color transitions in these areas. Figure 3.4 details the process and provides an example where the thin geometry is accurately rendered.

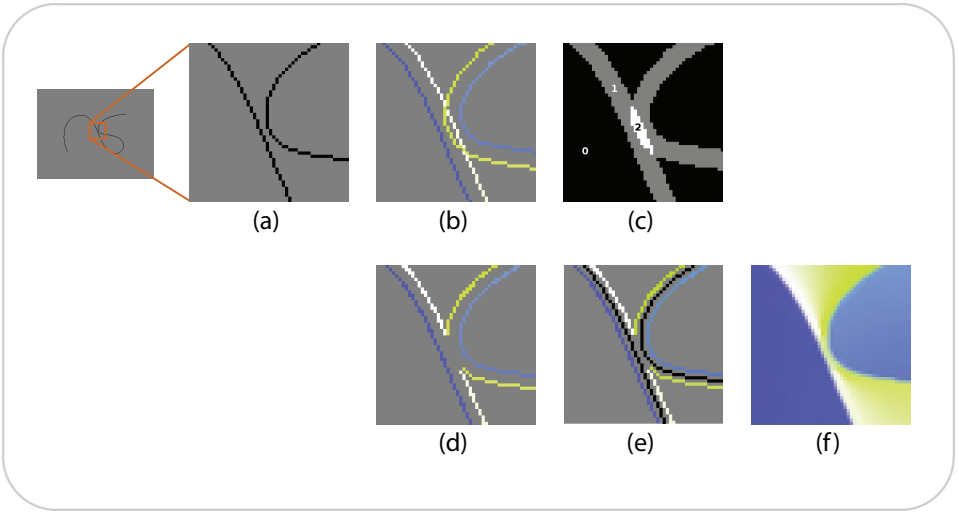


Figure 3.4: *Because colors are drawn at a distance from a curve, color constraints can superpose (b) when the actual curves do not intersect (a). To avoid artifacts, color intersections are detected (c), and the constraints are removed (d). The color diffusion is still accurate, because gradients positioned on the curves (e) guide the color variation.*

3.2.2 Diffusion

Given the color sources and gradient fields computed in the previous step, we next compute the color image I as the solution to a Poisson equation (3.1) (Figure 3.2, step (2)). To discretize the 2D equation described in equation (3.1), we use the fact that the raster image is a square grid of unit length. On such a grid, the required derivatives in the Poisson equation can be expressed at each point (i, j) by using the finite difference numerical method.

Given that the second order partial derivatives defining the Laplace operator are

$$\frac{\partial^2 I}{\partial x^2} = \frac{\partial}{\partial x}\left(\frac{\partial I}{\partial x}\right) \quad \text{and} \quad \frac{\partial^2 I}{\partial y^2} = \frac{\partial}{\partial y}\left(\frac{\partial I}{\partial y}\right)$$

and that the first order derivatives can be expressed accurately by central differences in the x

and y directions, the second derivatives in the context of a regular grid are

$$\frac{\partial^2 I}{\partial x^2} \approx \frac{\left.\frac{\partial I}{\partial x}\right|_{(i+\frac{1}{2}),j} - \left.\frac{\partial I}{\partial x}\right|_{(i-\frac{1}{2}),j}}{\Delta x} \text{ and } \frac{\partial^2 I}{\partial y^2} \approx \frac{\left.\frac{\partial I}{\partial y}\right|_{i,(j+\frac{1}{2})} - \left.\frac{\partial I}{\partial y}\right|_{i,(j-\frac{1}{2})}}{\Delta y}, \tag{3.2}$$

where Δx and Δy are the distances between two neighboring grid point in the x,y directions (Figure 3.5).

The first order derivatives with respect to x and y can be defined on either side of the grid pixel (i, j) with the same central difference:

$$\left.\frac{\partial I}{\partial x}\right|_{(i-\frac{1}{2}),j} \approx \frac{I_{i,j} - I_{i-1,j}}{\Delta x} \text{ and } \left.\frac{\partial I}{\partial y}\right|_{i,(j-\frac{1}{2})} \approx \frac{I_{i,j} - I_{i,j-1}}{\Delta y}$$
$$\left.\frac{\partial I}{\partial x}\right|_{(i+\frac{1}{2}),j} \approx \frac{I_{i+1,j} - I_{i,j}}{\Delta x} \text{ and } \left.\frac{\partial I}{\partial y}\right|_{i,(j+\frac{1}{2})} \approx \frac{I_{i,j+1} - I_{i,j}}{\Delta y}$$

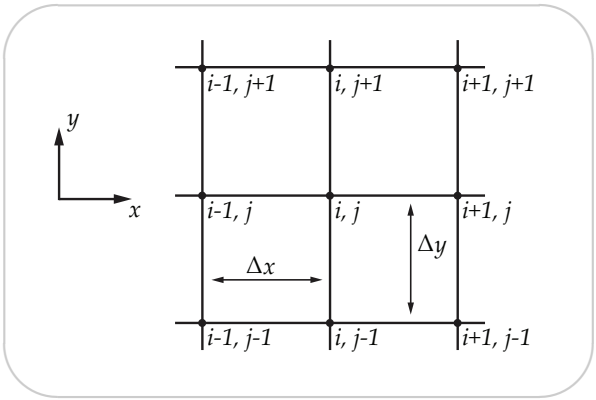


Figure 3.5:

Inserting the first order derivative approximations in equation (3.2) yields the second order derivative approximations:

$$\frac{\partial^2 I}{\partial x^2} \approx \frac{I_{i+1,j} - 2I_{i,j} + I_{i-1,j}}{\Delta x^2} \text{ and } \frac{\partial^2 I}{\partial y^2} \approx \frac{I_{i,j+1} - 2I_{i,j} + I_{i,j-1}}{\Delta y^2}$$

In a pixel grid, $\Delta x = \Delta y = 1$, and the discrete Poisson equation becomes

$$\Delta I_{i,j} = I_{i+1,j} + I_{i-1,j} + I_{i,j+1} + I_{i,j-1} - 4I_{i,j} = \operatorname{div} \mathbf{w_{i,j}},$$

Adding the color constraints, this leads to:

$$I_{i,j} = \begin{cases} C_{i,j} & \text{if pixel } (i,j) \text{ stores a color value } (\alpha\text{-mask} > 0 \\ \frac{I_{i+1,j} + I_{i-1,j} + I_{i,j+1} + I_{i,j-1} - \operatorname{div} \mathbf{w_{i,j}}}{4} & \text{elsewhere.} \end{cases} \tag{3.3}$$

The divergence of the gradient $\operatorname{div} \mathbf{w}$ is numerically computed with the same finite difference approximation:

$$\operatorname{div} \mathbf{w_{i,j}} = \frac{\mathbf{w_x}(\mathrm{i}+1,\mathrm{j}) - \mathbf{w_x}(\mathrm{i}-1,\mathrm{j}) + \mathbf{w_y}(\mathrm{i},\mathrm{j}+1) - \mathbf{w_y}(\mathrm{i},\mathrm{j}-1)}{2}.$$

The $M \times N$ equations thus obtained (where $M \times N$ is the size of the raster image) can be solved directly. But this requires solving a large, sparse, linear system, which can be very time consuming as $M \times N$ grows large.

To offer interactive feedback to the artist, we solve the equation iteratively with a GPU implementation of the **multigrid algorithm** [BHM00, GWL+03, MP08].

The idea behind multigrid methods is to use a coarse version of the domain to efficiently solve for the low frequency components of the solution, and a fine version of the domain to refine the high frequency components (Figure 3.6). The algorithm works in a V-like manner; color source image C and the gradients W_x and W_y are progressively downsampled (Figure 3.6 top). The solution is computed first at the lowest resolution, and then upsampled and refined (Figure 3.6 bottom). Jacobi relaxations are used to solve for each level of the multigrid. The method starts with the "initial guess" made by the coarser level. The finite difference equations (3.3) — with the corresponding local constraints — are applied again and again, updating the color values, until a maximum number of iterations is reached. For a given iteration k and a fixed resolution level l, the color value $I^k_{i,j}(l)$ is:

$$I^k_{i,j}(l) = \frac{I^{k-1}_{i+1,j}(l) + I^{k-1}_{i-1,j}(l) + I^{k-1}_{i,j+1}(l) + I^{k-1}_{i,j-1}(l) - \mathrm{div}\,\mathbf{w_{i,j}}(l)}{4}.$$

The color constraints are re-imposed after each iteration:

$$I^k_{i,j}(l) = C_{i,j}(l) \;\text{ if the } \alpha \text{ mask for pixel } (i,j) > 0.$$

To construct the image pyramid necessary for the multigrid solver, we downsample the gradient using a 3×3 kernel:

$$\begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix}$$

This filter is needed to capture all gradient directions from the finer scale and to preserve the gradient magnitude; a color variation of 1 in the x direction, for example, will thus be preserved as a variation of 1 in coarser scales, and summed to the neighboring variations. The color constraints are downsampled with an average filter: a pixel at coarse scale receives the average of the constraints of the four corresponding pixels in the finer scale. Note that the gray colored pixels in Figure 3.6 top are non-constrained points, and they are not considered when downsampling the constraints. Color upsampling uses the nearest neighbor technique to attribute to the four pixels in the finer scale the color value of the corresponding coarse-scale pixel.

Typically, for a 512×512 image we use $5i$ Jacobi iterations per multigrid level, with i the level number from fine to coarse. This number of iterations can then be increased when high quality is required. Our GPU implementation provides realtime performance on a 512×512 grid with a reasonable number of curves (several thousands).

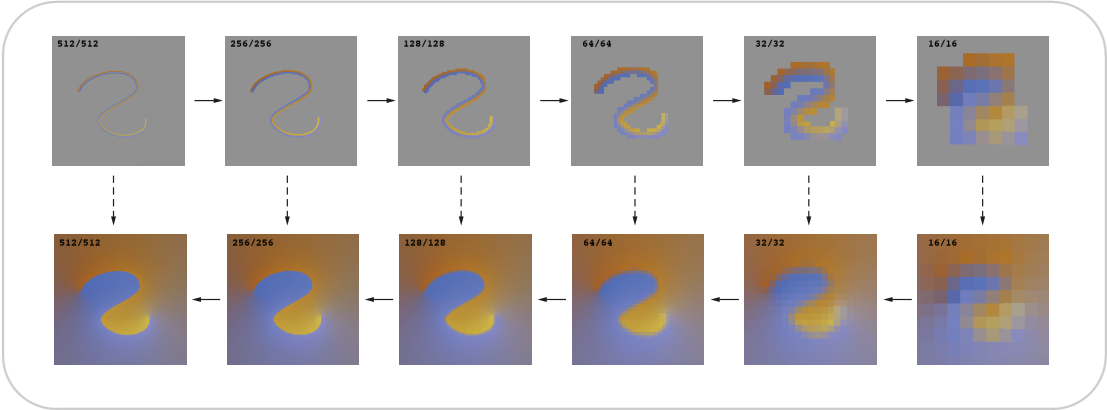


Figure 3.6: *The multigrid algorithm. Color and gradient constraints are repeatedly downsampled (top row). An initial solution is computed at the lowest level, by iteratively diffusing the color constraints. The solution is then refined at a finer scale, by using the coarse-scale solution and the finer-scale color constraints (bottom row).*

3.2.3 Reblurring

The last step of our color rendering pipeline takes as input the color image containing sharp edges, produced by the color diffusion, and reblurs it according to blur values stored along each curve. However, because the blur values are only defined along curves, we lack blur values for off-curve pixels. A simple solution, proposed by Elder [Eld99], diffuses the blur values over the image similarly to the color diffusion described previously. We adopt the same strategy and use our multigrid implementation to create a blur map B from the blur values. The only difference to the color diffusion process is that blur values are located exactly on the curve; no gradient constraints are therefore necessary. This leads to the following equation:

$$\begin{aligned} \Delta B &= 0 \\ B_{i,j} &= \sigma_{i,j} \quad \text{if pixel } (i,j) \text{ is on a curve} \end{aligned}$$

Giving the resulting blur map B, we apply a spatially varying blur on the sharp color image (Figure 3.2(3)), where the size of the blur kernel at each pixel is defined by the required amount of blur for this pixel. We use a routine implemented on the GPU [BFSC04], that iteratively blurs the image. This method is based on the observation that running n successive iterations of the diffusion equation

$$I^k_{i,j} = I^{k-1}_{i,j} + 0.25\Delta I^{k-1}_{i,j}$$

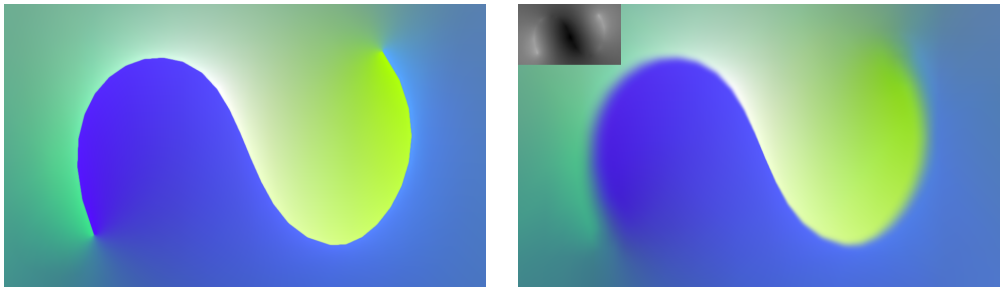
is equivalent (in the limit) to convolving the image I with a Gaussian kernel of width $\sqrt{2n\sqrt{2}}$. The step value 0.25 is chosen because it is the greatest value that ensures the stability of the equation [Rom03, BFSC04]. To stop blurring of pixels that have already reached their maximum blur value $B_{i,j}$, a weighting factor is introduced in the Laplace operator. By expressing the Laplacian with a finite difference, and including the weight, the explicit numerical equation

for the k-iteration can be written as:

$$I^k_{i,j} = w \cdot I^{k-1}_{(i,j)} + b_{(i+1,j)} \cdot I^{k-1}_{(i+1,j)} + b_{(i-1,j)} \cdot I^{k-1}_{(i-1,j)} + b_{(i,j+1)} \cdot I^{k-1}_{(i,j+1)} + b_{(i,j-1)} \cdot I^{k-1}_{(i,j-1)}$$

$$w = 1 - b_{(i+1,j)} - b_{(i-1,j)} - b_{(i,j+1)} - b_{(i,j-1)}$$

$$b_{m,n} = \begin{cases} 0, & \text{if } B_{m,n} > \sqrt{2k\sqrt{2}} \\ 0.25, & \text{otherwise.} \end{cases}$$



(a) (b)

Figure 3.7: *The color image result after reblurring the sharp image with the inset blur map.*

A reblurring example is shown in Figure 3.7. Despite the GPU implementation, the reblurring step is not adapted for multigrid approximation, and remains computationally expensive for large blur kernels (around one second per frame in our implementation). For real-time interaction, we bypass it during curve drawing and manipulations and reactivate it once the user interaction is complete.

3.2.4 Panning and zooming

Solving a Poisson equation leads to a global solution, which means that any color value can influence any pixel of the final image. Even though the local constraints introduced by the color sources reduce such global impact, this raises an issue when zooming into a sub-part of an image, because curves outside the current viewport should still influence the viewport's content. To address this problem without requiring a full Poisson solution at a higher resolution, we first compute a low-resolution diffusion on the unzoomed image domain (Figure 3.8 (a)), and use the obtained solution to define Dirichlet boundary conditions around the zooming window (Figure 3.8 (b)). This gives us a sufficiently good approximation to compute a full-resolution diffusion only within the viewport (Figure 3.8 (c)).

3.2.5 Rendering of the normals and (u,v) coordinates

Normals and (u,v) coordinates are diffused similarly to the color attribute, to obtain a smoothly varying interpolation between diffusion curves and a sharp transition across the curves. In the

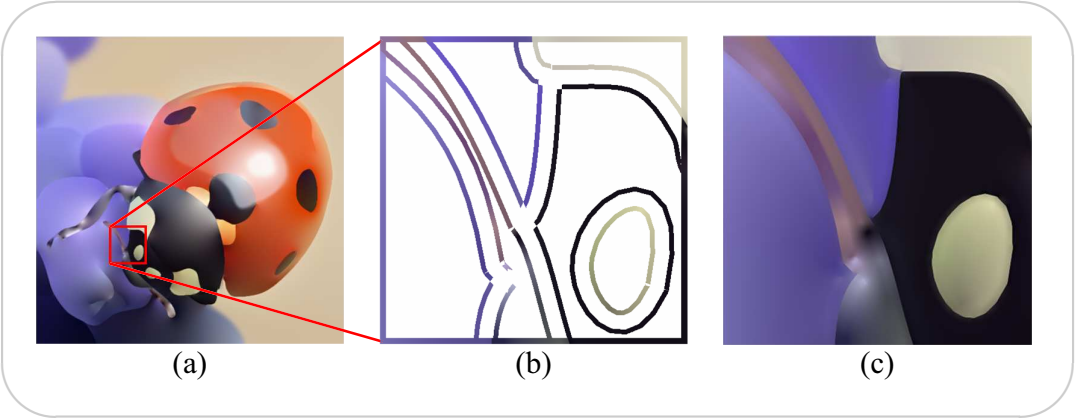


Figure 3.8: *The processing steps needed to zoom in: (a) the Poisson equation is solved at the low resolution. (b) The zoom-in window boundaries are fixed. (c) a new Poisson solution is computed inside the high resolution zoom-in window.*

special case of the normals, we re-normalize the normal-vectors after each Jacobi iteration in the multigrid, to ensure unit-length.

Even with the added computation of normal and u, v, on top of color diffusion, the system performs in real-time for a 512x512 resolution (about 50 frames per second), and interactive (10 fps) for 1024x1024, on a GeForce-GTX260 graphics card. This is possible on one hand because graphics cards have increased texturing capabilities, but mainly because we only update the diffusion result where needed. The color diffusion output, for example, is independent of the normal diffusion, and the user only edits one property at a time. Bitmap images play the role of a caching system for all diffusion computations.

3.3 Mesh-based diffusion

The raster-based rendering, while ensuring a smoothly-varying solution and real-time interaction, requires a specialized renderer, and is difficult to integrate in classical vector systems. The mesh-based diffusion represents an alternative rendering solution, that relies on the classical vectorial rendering of triangle meshes. This approach is a work in progress, that we are currently testing for sharp-color diffusion. We nevertheless consider that it completes this thesis, by showing that diffusion curves have the potential of "becoming" a standard vector primitive.

Mesh-based color diffusion consists of two steps: (1) *triangulation* of the image, that uses the diffusion curves geometry to divide the image into a set of triangles and (2) a *diffusion* process that associates to each triangle vertex a corresponding color value.

3.3.1 Triangulation

The triangulation process divides the image surfaces into triangles, and captures the discontinuities imposed by the diffusion curves by placing triangle edges along each curve. To this purpose, we use a constrained Delaunay triangulation (*CDT*) [Che87]. Given a set of *n* vertices in the plane together with a set of noncrossing edges, the *CDT* is the triangulation of the vertices with the following properties: (1) the prespecified edges are included in the triangulation, and (2) it is as close as possible to the Delaunay triangulation. The true Delaunay triangulation imposes that no vertex in the vertex set falls in the interior of the circumcircle (circle that passes through all three vertices) of any triangle in the triangulation. This property ensures a "nice" triangulation that maximizes the minimum angle for all triangles and avoids skinny triangles, making it suitable for diffusion. The *CDT* enforces the presence of user-defined edges in the generated mesh, with the result that the triangulation reflects the specified geometry, but accepts some edges which are not Delaunay.

The constraints we impose for the mesh generation are given by (a) the image boundaries and (b) the tesselated polylines that approximate the diffusion curve set at the desired resolution. Because edges should not cross each other, diffusion curves *intersections* are included in the polylines prior to triangulation. In practice, we use CGAL library[1] to detect polyline intersections and Jonathan Shewchuk's triangular mesh generator Triangle[2] to create the *CDT* (Figure 3.9 (b)).

(a) (b) (c)

Figure 3.9: *Mesh triangulation: (a) The result with a raster-based diffusion. (b) The triangulation. (c) The mesh-based diffusion result.*

3.3.2 Diffusion

The diffusion process is based on a discretization of the Poisson equation (Equation 3.1), as was the case for the raster-based diffusion (Section 3.2). However, in the regular pixel grid,

[1] **CGAL**, Computational Geometry Algorithms Library, `http://www.cgal.org`

[2] **Triangle**, A Two-Dimensional Quality Mesh Generator and Delaunay Triangulator, `http://www.cs.cmu.edu/~quake/triangle.html`

each point influenced equally all its neighbors. This is no longer the case for the mesh, where the vertex distribution is non-uniform. To obtain a good approximation of the diffused values, weights are considered in the discretization of the Laplace operator, to reflect the varying length of the edges in the mesh, and the corresponding shape and size of the triangles.

For a mesh $\mathbf{M} = (\mathbf{V}, \mathbf{T})$, with $\mathbf{V}$ the n vertices and $\mathbf{T}$ the set of triangles (i, j, k), the result of applying the discrete Laplacian to the color attribute I in equation (3.1) for each vertex i can be expressed as:

$$\Delta^d \begin{pmatrix} I_1 \\ I_2 \\ \vdots \\ I_n \end{pmatrix} = \begin{pmatrix} \operatorname{div} \mathbf{w_1} \\ \operatorname{div} \mathbf{w_2} \\ \vdots \\ \operatorname{div} \mathbf{w_n} \end{pmatrix}$$

Δ^d is the $n \times n$ matrix that defines the discrete Laplacian and expresses the equilibrium conditions for connected vertices:

$$\Delta^d_{ij} = \frac{3\,(\cot \alpha_{ij} + \cot \beta_{ij})}{2A}, \quad \text{if } (i, j) \text{ is a triangle edge}$$
$$\Delta^d_{ii} = -\sum_{j \neq i} \Delta^d_{ij},$$

where α_{ij} and β_{ij} are the two angles opposite to the edge in the two triangles having the edge (i, j) in common, and A is the sum of the areas of the triangles having i as a common vertex ([PJP93, DMSB99]). The mathematical deduction of these coefficients is given in the Bruno Lévy's HdR Habilitation thesis [Lév08] (Section 3.2.5, page 42).

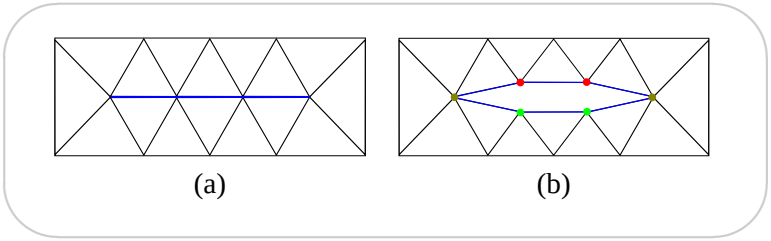


Figure 3.10: *To depict the color constraints, the diffusion curve edges in the triangular mesh (a) are doubled (b). Each side is then assigned a color (red and green, here), while at ending points the color is averaged.*

Constraints are imposed on the value of I along diffusion curves edges. Given that for colors we have two sets of constraints (left- and right-side), the edges that represent a diffusion curve need to be double edges. For this, a diffusion curve corresponds to a closed "hole" in the mesh, with left color constraints on one side, right color constraints on the other side, and an average color at the endings (see Figure 3.10). Thus, connections between left-side triangles and right-side constraints are broken (and vice versa). Because constraints can now be represented one "on top" of the other via two superposed edges, gradient insertion is no longer necessary, and we can consider the divergence $\operatorname{div} \mathbf{w_i} = 0$, for each vertex i in the mesh.

With the insertion of constraints, the Poisson equation becomes:

$$I_i = \begin{cases} C_i, & \text{if vertex } i \text{ is on a diffusion curve} \\ \frac{-1}{\Delta_{ii}^d} \left(\sum_{(i,j) \text{ edge in } \mathbf{T}} \Delta_{ij}^d I_j \right), & \text{overwise.} \end{cases}$$

We solve this linear system by the Jacobi iteration method, similar to Section 3.2. A preliminary result of this work is shown in Figure 3.9 (c).

3.4 Discussion

In the diffusion curves representation, the Poisson equation is used as the means to obtain a smooth interpolation between constraints. In this context, other smoothness functions can be considered that will lead to different types of variations in the attribute values.

For example, our diffusion is expressed as a minimization function for the first-order Laplacian, that creates a membrane-like variation (Figure 3.11 (a)). Minimizing some other order k of the Laplacian will interpolate differently between boundary conditions, as illustrated for 3D surfaces in Figure 3.11 (b) and (c).

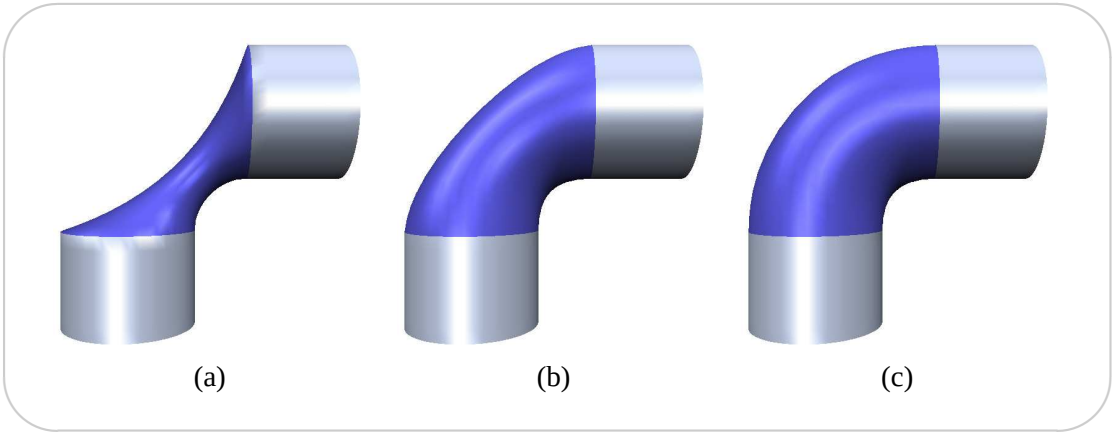


Figure 3.11: *The order k of the energy functional defines the stiffness of the surface in the support region and the maximum smoothness C^{k-1} of the boundary conditions. From left to right: membrane surface ($k = 1$), thin-plate surface ($k = 2$), minimal curvature variation ($k = 3$). Image taken from Botsch and Kobbelt [BK04]*

By considering such different piece-wise smoothing techniques and by combining them after the model of Botsch and Kobbelt [BK04], one might achieve a greater control on how attributes vary in the empty space between diffusion curves.

On the other hand, the Poisson equation is a tool extensively used in image processing applications, because it moves image manipulations from color space to color variations space. The gradient represents the image variations independently of the original colors, and the solution

of the Poisson equation reconstructs an image I from a modified gradient field by minimizing color discontinuities [PGB03, FLW02]. Nothing prevents the transfer of such bitmap editing techniques to the vector-based representation of diffusion curves, to obtain tools such as seamless copy and paste [PGB03] or image fusion [ADA+04, RIY04].

chapter 4

Creation and Manipulation

This chapter takes a more in-depth look at how a diffusion curves vector image can be manually created and manipulated by an user.

Section 1 shows how shapes and colors can be defined to create a vector image with complex color variations. Additionally, Section 1.3 presents shape manipulations that rely on the fact that diffusion curves are positioned at image discontinuities.

Section 2 discusses the possibility of decoupling shading variations from color variations observed in a material. This separation of shading allows the artist to interactively modify the illumination in the image without having to change the defined material colors.

Section 3 provides a way of enriching the diffusion curves vector graphics with textures. We discuss how a texture pattern can be created using diffusion curves specification, and how this pattern can be included in a diffusion curves image to create a textured vector graphics image.

To conclude, the proposed creation framework is evaluated in Section 4. First, the artist experience when creating vector art with diffusion curves is discussed. Second, diffusion curves are compared with existing representations.

1 Shape and Color

This section details the process of creating smooth-shaded vector images[1] using the diffusion curve primitive. Only the geometry (P), color (Cl and Cr) and blur (Σ) attributes are used.

The process of creating vector illustrations varies among artists. One may start from scratch and give free rein to his imagination while another may prefer to use an existing image as a reference. We provide the user with both options to create diffusion curves shapes and colors. For *manual* creation, the artist can create an image with our tool by sketching the lines of the drawing and then filling in the color. When using an image as a template, the artist can *trace* manually over parts of an image and we recover the colors of the underlying content.

1.1 Manual creation

When creating a diffusion curves drawing from scratch, the artist employs the same intuitive process as in traditional drawing: a sketch followed by color filling.

To facilitate content creation for the artist, we offer several standard vector graphics tools: editing of curve geometry, curve splitting, copy/paste, zooming, color picking, etc. We also developed specific tools: copy/paste of color and blur attributes from one curve to another, editing of attributes control points (add, delete, and modify), etc. The complete description of our prototype user interface is found in Appendix A. A different system, that follows the indications in our paper [OBW+08], but proposes new interface features, has been implemented by Henry Korol (`http://www.henrykorol.com/DiffusionCurves.rar`).

To illustrate how an artist can use our diffusion curves, we show in Figure 4.1 the different stages of an image being drawn with our tool.

1.2 Tracing an image

In many situations an artist will not create an artwork entirely from scratch, but instead use existing images for guidance. For this, we offer the possibility of extracting the colors of an underlying bitmap along a drawn curve.

The challenge here is to correctly extract and vectorize colors on each side of a curve. We also need to consider that color outliers might occur due to noise in the underlying bitmap or because the curve positioning was suboptimal. We first uniformly sample the colors along the curve at a distance d in the direction of the curve's normal. The sampling distance is the same as the one used in the rasterization step of rendering (Section 3.2.1). We then identify

[1] The creation, manipulation and vectorization of smooth-shaded vector images was presented in our paper [OBW+08] at SIGGRAPH 2008. It was a work done in collaboration with Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot and David Salesin.

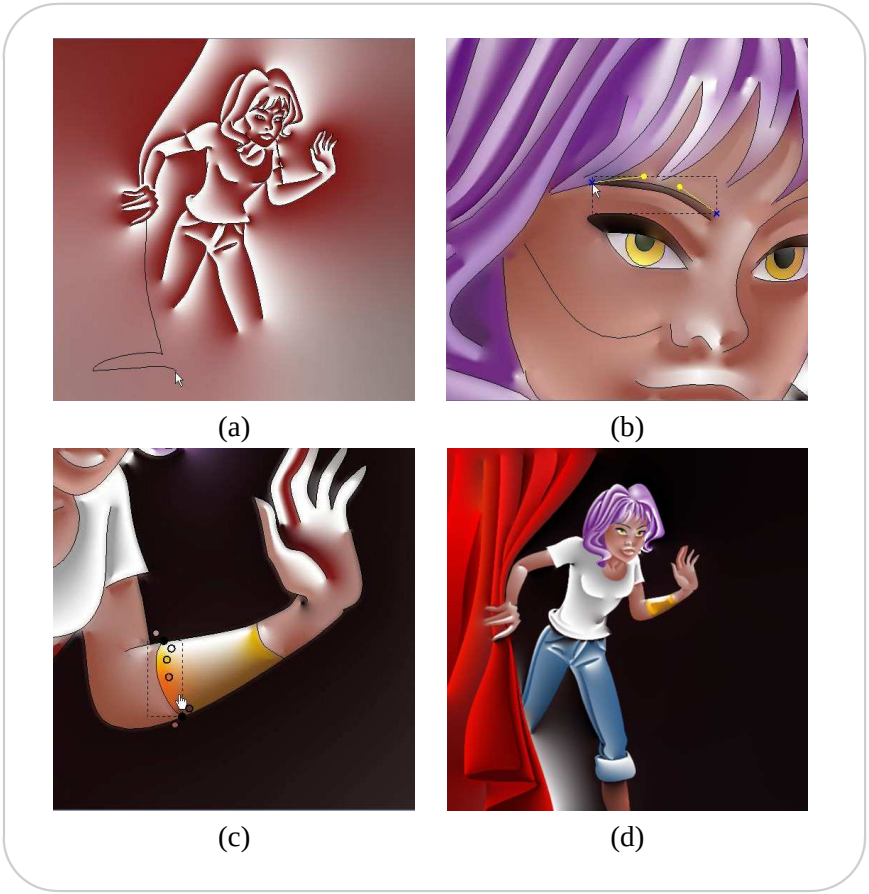


Figure 4.1: *Example steps for manual creation: (a) sketching the curves, (b) adjusting the curve's position, (c) setting colors and blur along the diffusion curve and (d) the final result. The image was created by an artist at first contact with the tool and it took 4 hours to create* © *Laurence Boissieux.*

color outliers by measuring a standard deviation in a neighborhood of the current sample along the curve. To this end, we work in CIE L*a*b* color space (considered perceptually uniform for just-noticeable-differences), and tag a color as an outlier if it deviates too much from the mean in either the L*, a* or b* channel. We then convert back colors to RGB at the end of the vectorization process for compatibility with our rendering system.

To obtain a linear color interpolation similar to that used for rendering, we fit a polyline to the color points using the Douglas-Peucker algorithm [DP73]. The iterative procedure starts with a line connecting the first and last point and repeatedly subdivides the line into smaller and smaller segments until the maximum distance (still in CIE L*a*b*) between the actual values and the current polyline is smaller than the error tolerance ε. The end points of the final polyline yield the color control points that we attach to the curve. The color vectorization process is illustrated in Figure 4.2.

A creative example that uses color sampling is illustrated in Figure 4.3(b)-top image, where an artist has drawn very stylistic shapes, while using the color sampling feature to reproduce the

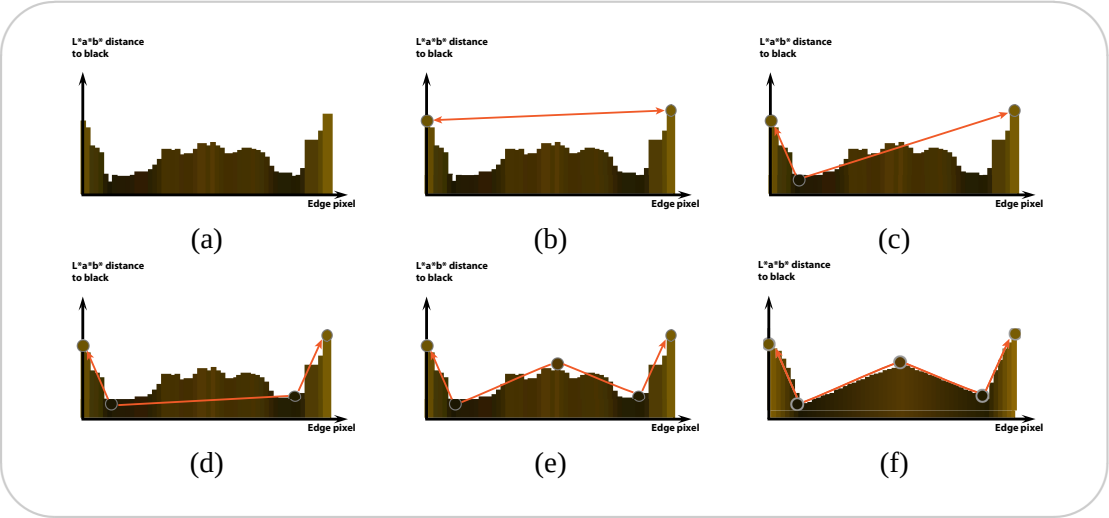


Figure 4.2: *Color sampling using the Douglas-Peucker algorithm [DP73]: (a) Original color sampled along a diffusion curvefrom an input bitmap; (b)—(d) successive polyline subdivisions; (e) final set of extracted color control points; (f) the rasterized color variation, obtained by linear interpolation between color control points.*

global tone of the original image, similarly to an in-painting process [BSCB00].

When tracing over a template, one would normally want to position the curves over color discontinuities in the underlying image. Since it is not always easy to draw curves precisely at edge locations in a given image, we provide some help by offering a tool based on *Active Contours* [KWT87]. An active contour is attracted to the highest gradient values of the input bitmap and allows the artist to iteratively snap the curve to the closest edge. The contour can also be easily corrected when it falls into local minima, or when a less optimal but more stylistic curve is desired. Figure 4.3(b)-bottom shows the image of a lady bug created using geometric snapping and color extraction. While the artist opted for a much more stylized and smoothed look compared to the original, the image still conveys diffuse and glossy effects, defocus blur, and translucency. The actual interface tools we implemented for tracing the image are described in Appendix A.

1.3 Shape manipulation

Because a diffusion curve position marks a discontinuity (until now, a color discontinuity), geometric deformations of the diffusion curve shape reflect in coherent deformations of the drawing. An example of such global stylization is shown in Figure 4.4 (b).

The nature of diffusion curves also means that they only represent important features of the image, and that they are independent from one another. We can take advantage of these properties to attach an *importance* value to each curve. This notion of importance can be used to adjust the amount of detail present in the final image, and to create more or less complex

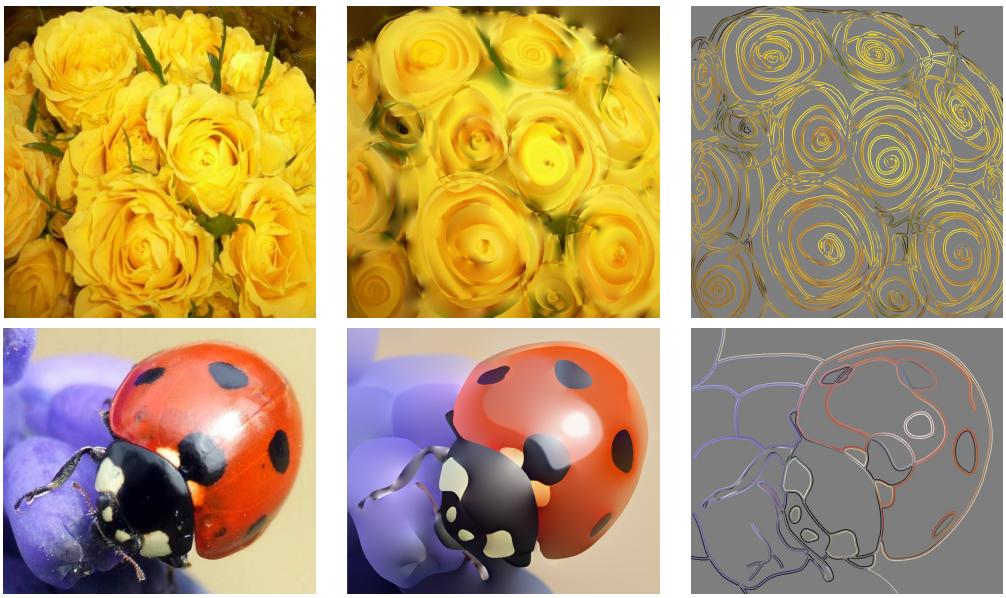


Figure 4.3: *Tracing with diffusion curves: (a) Original bitmaps; (b)* top*: Result of a stylistic tracing using color sampling (drawing time: less than a minute) © Philippe Chaubaroux;* bottom*: Result of a tracing using active contours and color sampling (drawing time: 90 minutes) © Adrien Bousseau. (c) The corresponding diffusion curves (color sources have been thickened for illustration).*

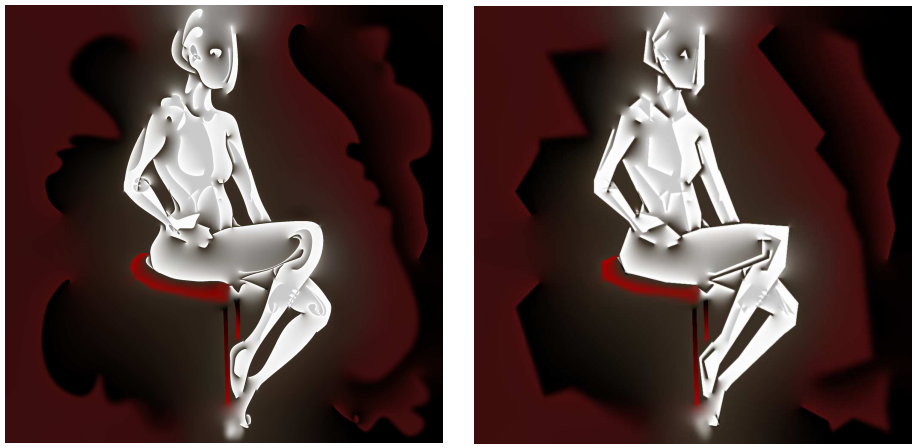


Figure 4.4: *Example of shape manipulation: (a) Original diffusion curves drawing © Philippe Chaubaroux. (b) Global shape stylization applied to (a);*

appearances. Detail removal is especially useful for maintaining the readability of images at different resolutions; details appear as we zoom in, but are discarded for small scale versions (see Figure 4.5).

The importance of a curve is initially the level of zoom at which that curve was first drawn, and can subsequently be changed by the user.

(a) (b)

Figure 4.5: *Example of detail removal: (a) Original diffusion curves drawing © Laurence Boissieux, with a simplified version in the upper right corner. Note how the folds and hair retain their readability because less important diffusion curves have been removed. (b) The same simplified version, shown at a larger scale.*

Diffusion curves, as vector-based primitives, benefit from the editing advantages of traditional vector graphics: curve shapes and colors can be directly modified (Figure 4.1), and keyframing animation is easily performed via linear interpolation of geometry and attributes (Figure 4.6).

Figure 4.6: *Keyframing with diffusion curves: Three keyframes of an animation © Laurence Boissieux.*

2 Shading

In the creation process described up to now, shading and material variations were inextricably tied together in a single attribute — color — that the artist used to manually create smooth color gradients. This is evident in our example images; Figure 4.5 shows how color variations can depict color gradation in fabrics, while in Figure 4.1 (d) the same type of gradients depict shadows in the curtain folds.

However, in this setup, changing the illumination implies manipulating the entire color attribute; this can prove cumbersome in some cases, and hinders the flexibility of the system. This is especially the case when considering animations, where characters move under a fixed light, making the material color constant and the shading variable [PFWF00, Joh02].

We therefore proposed to decouple shading manipulation from material color variations. In this section, we will show how shading can be defined and manipulated by using two other attributes of our vector primitive: the α transparency value of the color, and the *normals*; these attributes increase the *appearance* possibilities for the depiction of lighting effects.

Normals

Surface normals are commonly used in shading models for 3D rendering. Physically-based reflectance models — such as the Lambertian, Phong [Pho75] or Oren-Nayar [ON95] models — take into account light direction and surface normals to create photorealistic approximations of illuminated 3D scenes. Non-photorealistic lighting models also rely on normals to create stylized shaded scenes. Toon shading, for example, uses surface normals to create shadows and highlights that mimic the style of comic books and cartoons [AWB06, BTM06, TABI07].

Shadows and highlights can thus be automatically computed from normals, for a given light position and using a predefined reflectance model. With normal information, a large number of material properties can be imitated, in varied rendering styles, by simply modifying the reflectance model, and not the original image. Our *normal* attribute increases the flexibility of the diffusion curve vector primitive, and simplifies the user task.

Just as with colors, normals can be specified on both left and right side of the diffusion curve geometry. Values are then interpolated in the free space between the curves. To specify the normals along diffusion curves, we follow the inspiration of previous planar surface inflation methods [Joh02, JC08]. These approaches rely on the fact that lines in contour drawing appear mostly as a *border setting* between objects, or delineate the essential shape changes of an object interior. Changes in normals are then depicted by these contours, and it is common to assume that the surface normal implied by the contour lines is orthogonal to the defining curve's instantaneous tangent.

We make use of this contour property to allow users to conveniently specify convex, flat, or concave surfaces. In a *depth-slope* mode, we fix normals to be oriented along the instantaneous

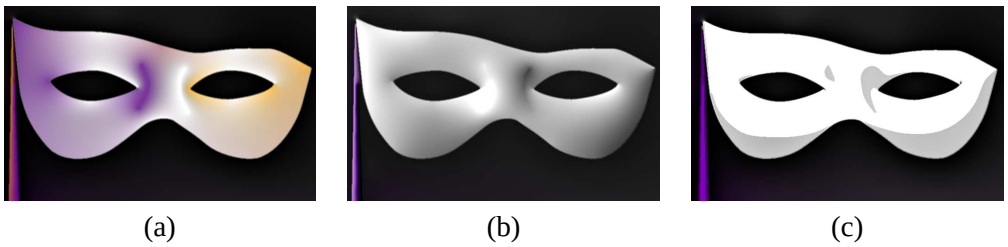


Figure 4.7: ***Shading effects****: (a) An artistic shading, realized with manual color controls. (b) A realistic shading effect based on the normals.(c) Toon shading, using the same normal information as (b).*

normal to the diffusion curve. The starting shape is given by the hand-drawn curve, and all the user has to specify is the slope orthogonal to this curve. Real-time feed-back allows direct control of the amount of inflation and deflation, and control points can be placed anywhere along the curve to impose slope values. Screen captures illustrating this process are presented in Figure 4.8.

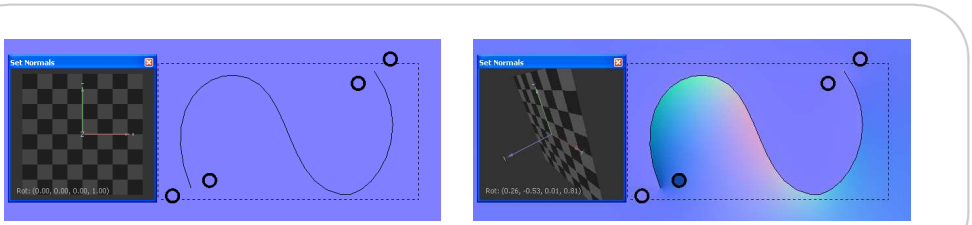


Figure 4.8: ***The normal widget****: Screen captures of the normal values being defined along a diffusion curve.*

In addition to this high-level control, the user can also choose a *free normal* mode. In this mode, the user specifies for each control point the complete unit vector (x,y,z). Values are then linearly interpolated along the curve.

Once the normal values are defined, shading can be automatically computed and stylized using various lighting models (Figure 4.7 (b) and (c)). Shadows and highlights can be interactively updated by simply changing the *light position* (Figure 4.9). The tools used for specifying normals are described in Appendix A.

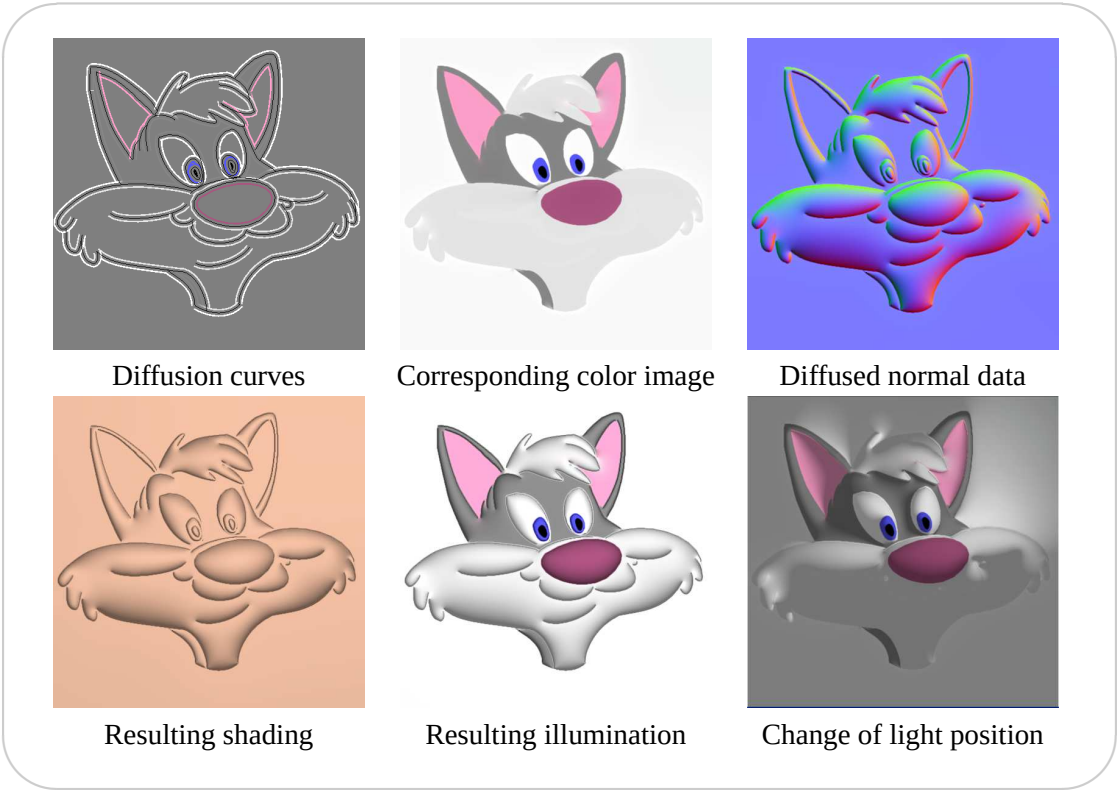


Figure 4.9: *Example of shading creation. Cat inspired by "Lumo" [Joh02].*

3 Texture

Textures add detail and interest to an artwork, but are equally significant for inferring scenic depth, surface orientation and other 3D shape properties from a 2D image. This section discusses how artists using diffusion curves can define and deform complex textures directly in 2D images, without requiring a 3D model of the depicted scene[2]. Three steps are proposed for creating a textured design. (1) The artist starts by creating a texture with motifs of variable color and shape. (2) A line drawing is created with diffusion curves, and is used as a support drawing for the textures. (3) The textures are positioned inside the support drawing and deformed to reflect the artist's intention.

3.1 Creating the texture-map

Our texture-map creation tool lets users design *regular* and *near-regular* vector textures from scratch (Figure 4.12). The user starts by drawing a texel that is automatically replicated throughout a grid. The texel is in itself a complete diffusion curve drawing.

Normal information is used to model textures with a physical macro-structure [LM99], for which the shading would be cumbersome to depict with manual appearance variations alone. An example of texture with normal variations is the flower texture in Figure 4.10.

When including textures with normals in the support drawing, our model places a texture "on top" of the suggested shape. The final normal vector used for shading is the normalized sum of the texture and shape normals (Figure 4.11 (a)). While the use of normals allows for automatic shading effects based on pre-defined lighting models, artistic lighting often involves physically unrealizable shading. In such cases the user can still manually define colored shading gradients via the color attribute, and use the α values of each color to indicate how much of the original color is mixed with the texture (Figures 4.11 (b)). Note, how this effect is used in Figure 4.17 to separate texture (gray) from manual shading (color), to achieve a more complex combined result.

Once the example texel has been drawn, the user can define the spacing between neighboring texels by interactively adjusting the grid spacing. For a more varied appearance, several example texels can be defined. In the spirit of [BA06], our system automatically creates new texel instances by interpolating the shape, normal, color and blur parameters of these examples. We use a simple point-to-point correspondence, where we assume that the i-th control point (for geometry, color, or blur) matches the i-th point on the corresponding curve in all texels. We enforce this in our interface by automatically adding corresponding controls to all texels when the user edits one texel instance. We then perform a linear interpolation between instances. This straightforward approach works well in practice, gives real-time feedback, and creates satisfactory texel variations. Figure 4.10 uses such random interpolation for shape and colors.

[2] This system for designing and manipulating regular or near-regular textures in 2D images is a work done in collaboration with Holger Winnemöller, Joëlle Thollot and Laurence Boissieux. It has been published at EGSR 2009 [WOBT09].

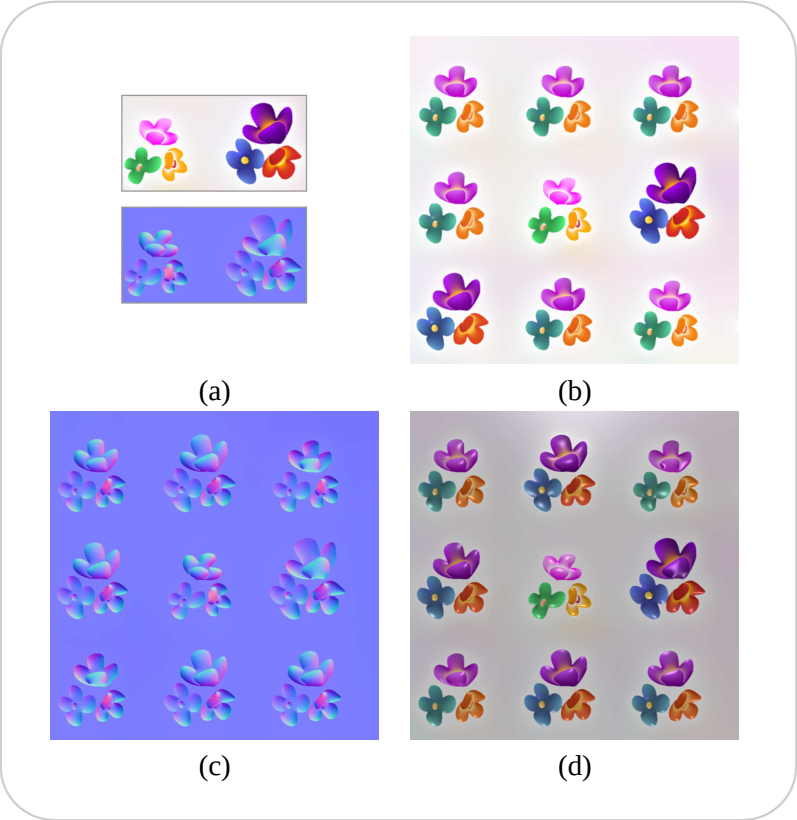


Figure 4.10: ***Near-regular texture****: An example of automatic creation of texels from 2 user-defined exemplars. (a) The user input of color and normal variation (artist time: 25 minutes). (b) The near-regular color texture map. (c) The generated normal map. (d) A lighting effect shown on the macro-structure* © *Laurence Boissieux.*

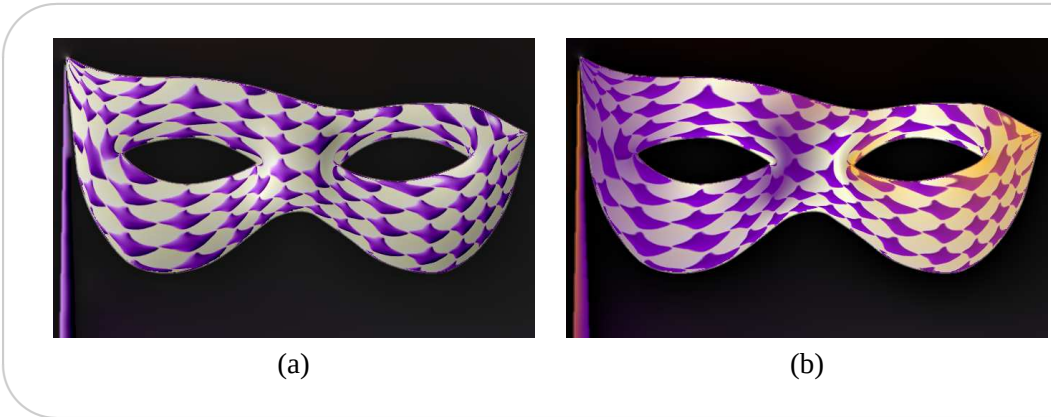


Figure 4.11: ***Shading effects****: (a) A realistic shading effect based on the normals, for a drawing with texture. (b) A manual shading, with* α *transparency colors layered on top of a textured region* © *Laurence Boissieux.*

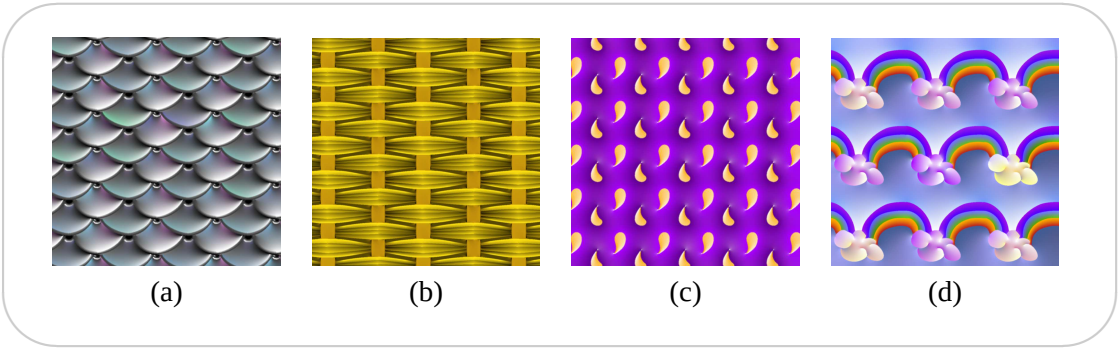


Figure 4.12: ***Texture examples****: Examples of textures designed with our texture-map creation tool: from realistic (a) to artistic (d). Artist time varies from 5 to 30 minutes © Laurence Boissieux*

Using the method we described in our rendering approach (Section 3.2), the texel attributes are diffused to obtain texture-maps with smoothly varying attribute values. Color and normals are computed independently and stored in two separate bitmap images. However, our system still preserves the resolution independence of vector graphics as the rasterization is recomputed for any given zoom level. During draping, we represent and apply the texture maps as bitmaps, and only recompute them for zooming actions. This allows us to obtain real-time visual feedback, which would otherwise be intractable. Additionally, this decouples the draping system from the texture-map representation, allowing our system to handle virtually any texture generation method (bitmap samples, procedural, etc.) that produces raster output. The interface is detailed in Appendix A.

3.2 Creating the support drawing

The support drawing is a vector drawing created with our diffusion curve primitives. The vector curves (Bézier splines) are *supporting* the control points containing all the parameters — color, blur, normals and (u,v). The user can optionally decide which parameters are used and which are deactivated. This allows for a variety of applications ranging from a full drawing created from scratch (Figure 4.14) to draping textures over an existing bitmap image, as shown in Figure 4.13.

Contrary to color and texture-draping parameters, which vary anywhere in the image, there is only one texture-map associated with a given texture region. We therefore automatically compute a planar map from the supporting drawing, separating the drawing into closed regions (Figure 4.14). And we allow the user to attach a texture to a planar map region. In practice, we compute a planar map arrangement from the diffusion curves geometry with the help of the CGAL[3] library.

[3] http://www.cgal.org/

3.3 Draping Textures

In this section, we focus on how to position, flow and distort a texture-map within the supporting diffusion curves drawing. By taking inspiration from garment and furniture design (Figure 4.13) we identify the following prevalent types of draping features:

- Shape contours - delineate the extent of textured regions
- Creases & folds - exhibit sudden change of normals
- Occlusions - where one piece of material overlaps another
- Seams - where one piece of material abuts another

All of these features describe discontinuities of one type or other, i.e. features are assumed to be mostly smooth except for a few distinct lines and curves. The proposed draping parameters, attached to diffusion curves, mark these discontinuities and model their variations. Two of the diffusion curve attributes are particularly used for texture draping: *normals* and (u,v) coordinates direct manipulation. Artists design normal fields to indicate shape-based distortions of the texture. In addition, artists can locally manipulate 2D texture-coordinates using a rubber-sheet technique based on (u,v) controls.

Figure 4.13: *(a) Our texture draping approach supports editing operations that allow for precise placement of textures in an image. Note that all of the demonstrated edits are specified along discontinuity curves. (b) Given a set of texture-edit curves, we can apply any number of vectorial textures (here, composited over a photograph). The inset vectorial texture swatches are also designed using our system, except for the rightmost swatch, which is a bitmap texture.*

3.3.1 Draping parameters

Given a texture-map, high-level parameters are provided for its inclusion in the supporting vector drawing. *Global affine transformations* (scale, translation, rotation) permit the artist to

quickly place the texture in the desired area. The other important role of the *normals* attribute, aside from shading, is to suggest surface shape for texture draping. This has the additional advantage of directly correlating the two shape cues (shading and texture); plausible deformations of both cues are automatically computed when the normal values are modified.

Finally, for a finer control over how the texture folds and ripples, the artist can use a direct (u,v) coordinate adjustment (Figure 4.14).

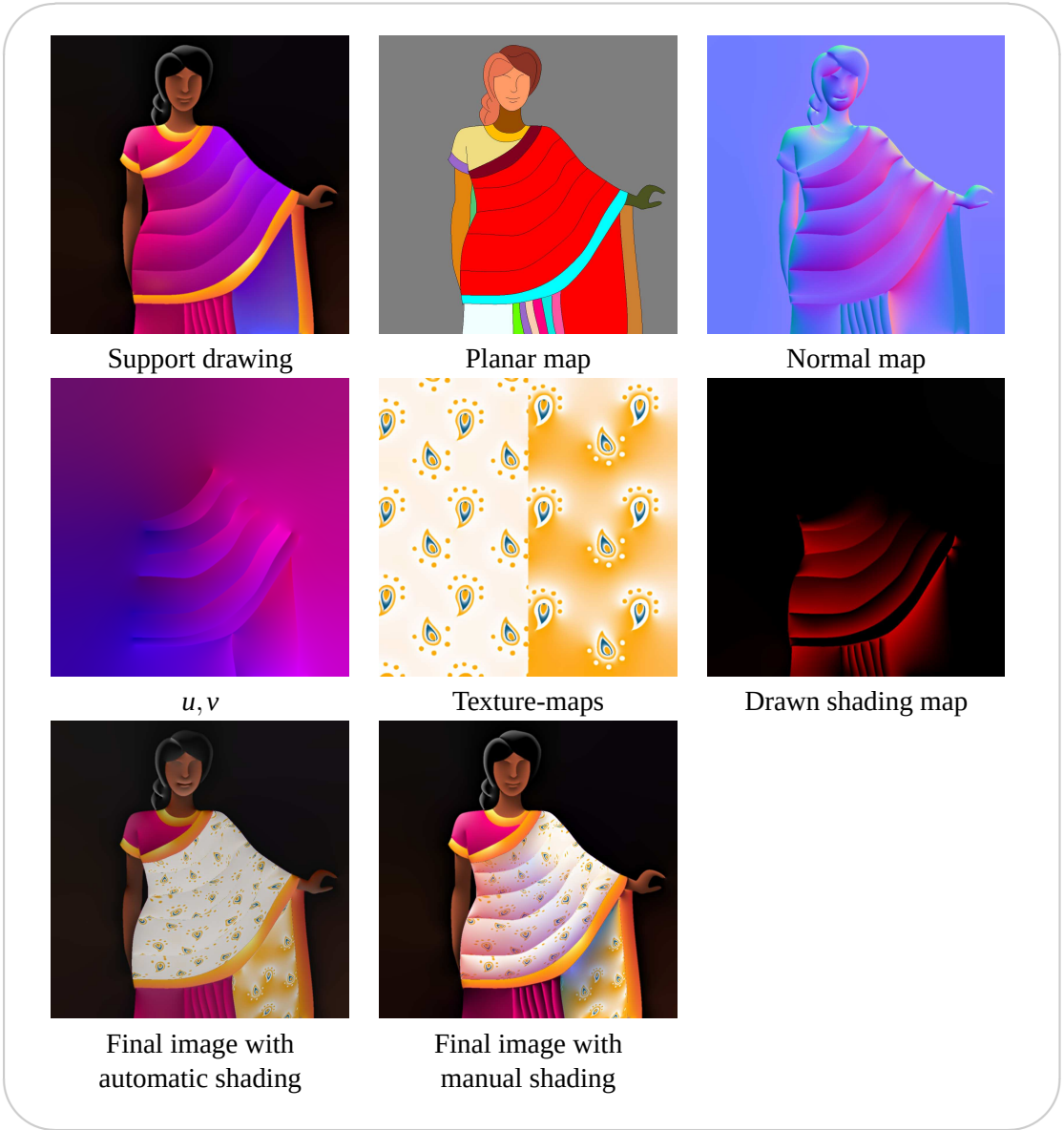


Figure 4.14: *In this figure the first lines show the set of inputs used to create a complete drawing. The last line shows the resulting image with an automatic shading (based on the normal map), or a manual shading (based on a drawn shading map)* © *Laurence Boissieux.*

3.3.2 Texture attachment

A texture is included in the support drawing by attaching it to a region in the planar map (as illustrated in Figure 4.14). For this, the artist specifies a point inside the region, and thus also defines the texture's center. When the drawing is modified and the planar map updated, the attachment point's image-space coordinate dictates which new region corresponds to the texture. If several texture attachment points fall within the same region, the user simply selects the active texture or re-distributes other textures to different regions.

3.3.3 Parallax mapping

Given the normal map computed from the normal parameters, we provide the user with a warp parameter that scales the inflation-amount of the surface implied by the normals. We use the parallax mapping technique [Wel04], which warps the texture to give the impression of parallax foreshortening. Given a texture applied to a flat polygon (in our case the image rectangle), parallax mapping offsets each texture coordinate to suggest complex surface shape.

In practice, a height-field needs to be computed from the normal-map. As in other parts of our system, we solve a Poisson equation for this purpose [WSTS08]. The gradient field is given by the expected difference in height between neighboring pixels, considering how the corresponding normals vary.

When a texture map is applied to a plane surface (the image rectangle), the final appearance is flat-looking, and very different from what a textured uneven surface would look. Figure 4.15 (a) shows that, when looking at the flat surface along the depicted eye vector, the textured point A is visible. However, should the actual uneven surface be seen, point B would be visible. The idea of parallax mapping is to use the textured flat surface, but to correct the texture coordinate corresponding to point A, so the texture of point B is displayed instead. The scheme in Figure 4.15 (b) illustrates how one can modulate the eye vector by the surface height to obtain a texture coordinate offset.

Three components are required to warp the texture map according to the parallax: the starting "flat" (u,v) texture coordinate at a point P in the polygon (T_0), the surface height h at point P, and the normalized eye vector V pointing from the pixel. An offset is then computed by tracing a vector parallel to the planar surface from A, the point on the uneven surface directly above P to the eye vector. This new vector is the offset and can be added to T_0 to produce the new texture coordinate T_n.

$$T_n = T_0 + (h \cdot V_{x,y}/V_z)$$

However, this equation assumes that the surface point corresponding to T_n has the same height as the point at T_0. For small offsets, heights are likely to be very close, so this approximation will yield good results. For shallow viewing angles, however, the proposed equation will lead to increasingly large offsets. This greatly reduces the possibility of the point at T_n actually

being close to B, the observed point in the real surface. A simple solution is to limit the offset, so that it is never longer than h (as illustrated in Figure 4.15 (c)). For an initial image pixel P with texture coordinates T_0, the final texture coordinates become:

$$T_n = T_0 + (h \cdot V_{x,y})$$

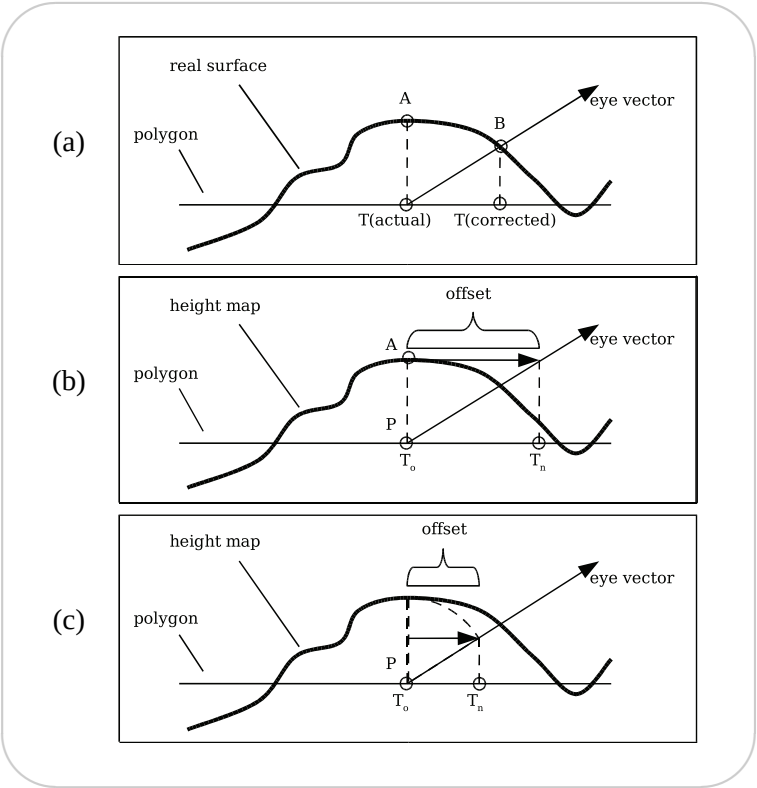


Figure 4.15: ***Parallax mapping:*** *(a) The observed texture does not depict the uneven surface, because the texture map has been flattened onto the polygon. (b) Calculating the correcting offset. (c) Calculating the bounded offset. Images taken from Welsh's tutorial [Wel04].*

Because positive heights make the texture appear closer to the viewer, thus upsampling it, we ensure that texture deformation artifacts are minimized by rendering the texture at the maximum height, and considering the image plane at height 0.

3.3.4 Direct texture coordinate control

For design distortions, and shape distortions that cannot be modeled with a normal field (Figure 4.17), the user can locally offset (u,v) coordinates (Figure 4.14). Our implementation is inspired by rubber-sheet techniques, where the user can specify and pin exact texture coordinates at chosen control-points along a curve, and the remaining texture stretches in-between to fit the constraints. As elsewhere, this is achieved with linear constraint interpolation and Poisson diffusion.

Figure 4.16: ***Example of parallax mapping:*** *(a) Checkerboard texture is placed in the image, but no parallax mapping is performed. (b) Parallax mapping is used to deform the texture, but no shading is applied. (c) Manual shading is added. (d) The final texture is shown, with no parallax mapping. (e) The final shaded texture* © *Laurence Boissieux.*

To initialize the (u,v) coordinates, the artist can use a sampling option. Default positions and values are automatically computed to create the least possible distortion in the texture, while adding as few control points as possible along the chosen curve. To do so, we use the Douglas-Peucker algorithm [DP73] to find a set of points that approximate the selected Bézier curve and place (u,v) coordinates on them, so that the texture lies flat in the image space. The Douglas-Peucker sampling strategy is described for color sampling in Section 1.2. In the current case, we approximate the geometry of the Bézier curve: the initial input is the tessellated polyline approximation of the Bézier curve drawn on the screen. We progressively simplify the polyline

until the distance from the original tessellation exceeds 0.1; the point coordinates are between -1 and 1.

Parallax mapping and (u,v) mapping can be combined easily for complex folds and rippling results (Figure 4.14). In that case, the (u,v) coordinates are used as initial coordinates for the parallax warping. Both normals and (u,v) coordinates react to curve deformations by following their respective control points and thus creating smooth adjustments in texture deformation.

(a) (b)

Figure 4.17: ***Example of direct texture control****: Here, an artist managed to skillfully drape the texture to suggest curly hair, flow the scales texture along the mermaid's tail, and apply fins to the tail's tip - all by direct manipulation of u,v coordinates. (a) Textures only. (b) Manual shading applied to textures* © *Laurence Boissieux.*

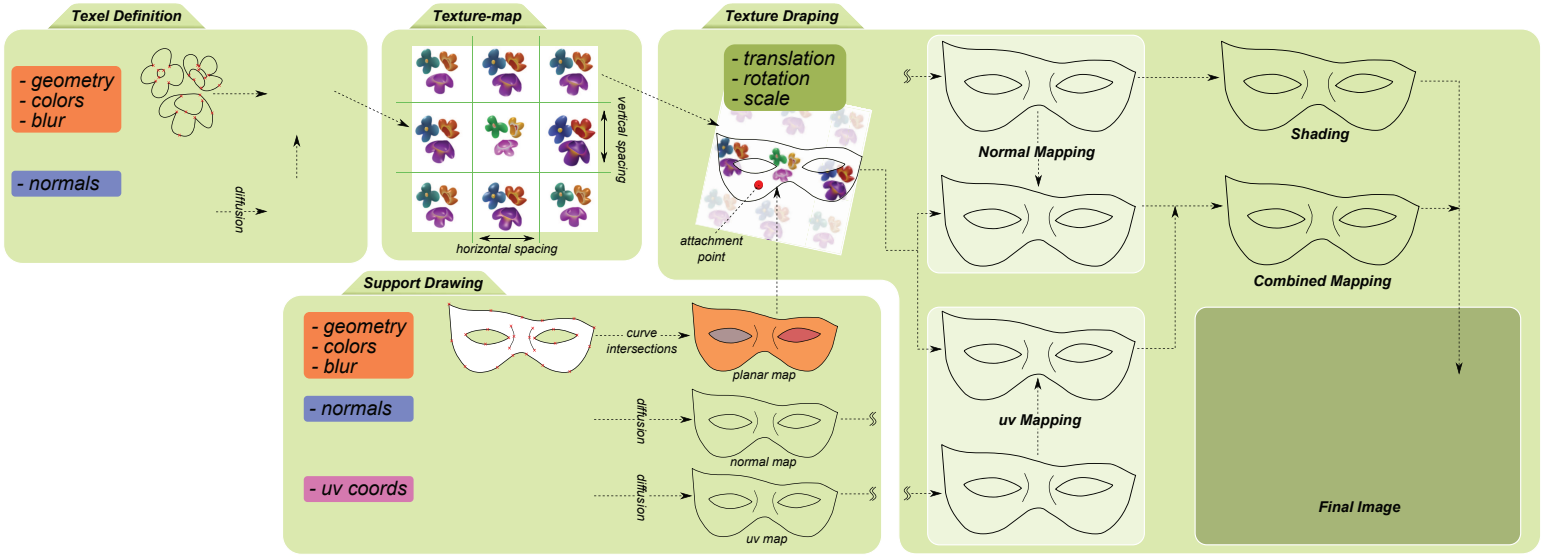


Figure 4.18: *Diffusion curves: Our vector design system allows the creation of complex color gradient, shading, and textures. A drawing typically starts with the creation of a* ***support drawing*** *and* ***texels****. Texels are combined into* ***texture-maps****. The texture-map is* ***draped*** *over the image using either* ***normal*** *controls,* ***u,v*** *controls, or a combination of both. The final image can be optionally* ***shaded*** *automatically using normals, or manually using the supporting drawing's diffusion curve colors.*

4 Discussion

The complete diffusion curves system is summarized in Figure 4.18. Using our system, an artist can manipulate shape and color, define shading, and integrate texture into the vector drawing. Artist validation for our approach, as well as comparisons with previous methods, are discussed in the next sections. Color (Section 4.1) and texture (Section 4.2) are treated separately.

4.1 Shape and color

To validate our approach and to collect valuable practical feedback, we had several artists use our diffusion curves prototype for shape and color. Most figures in Section 1 were generated in these sessions.

All artists using our system were well versed in digital content creation tools, with no technical background. They were given a brief paper tutorial (similar to the interface description in Appendix A), amounting to approximately 10 minutes of instructions. The artists were able to create many varied and intricate examples from the very first session and found the manipulation of diffusion curves intuitive after a short accommodation phase. Manual image creation took anywhere from several minutes (Figure 4.3(b)) to a few hours (Figure 4.1).

Comparison with Gradient Meshes

In the previous sections, we have discussed the shape and color attributes of our vector representation, and explained the various options at an artist's disposal to create smooth-shaded images thanks to this intuitive representation. We now compare our approach with the most commonly used vector tool for creating images with similarly complex color gradients: Gradient Meshes.

Representational efficiency: In terms of sparsity of encoding, both gradient meshes and diffusion curves are very efficient image representations. A direct comparison between both representations is difficult, as much depends on the chosen image content (for example, gradient meshes require heavy subdivision to depict sharp edges and it can be difficult to conform the mesh topology to complex geometric shapes). Furthermore, Price and Barret [PB06] presented a more compact sub-division gradient mesh, yet all available tools employ a regular mesh. While the diffusion curves representation appears more compact at first glance (see Figure 4.19), it should be noted that each geometric curve can hold an arbitrary amount of color and blur control points (see Table 4.1). So, while the sparsity of encoding of both representations can be considered comparable, we would argue the flexibility of diffusion curves to be a significant benefit, as it allows us any degree of control on a curve, without a topologically-imposed upper or lower bound on the number of control points.

Figure	Curves	*P*	*Cl*	*Cr*	Σ
Roses, Fig. 4.3 left	20	851	581	579	40
Lady bug, Fig. 4.3 right	71	521	293	291	144
Curtain, Fig. 4.1	131	884	318	304	264

Table 4.1: *Number of curves, geometric control points (P), left and right color control points (Cl, respectively Cr) and blur control points (Σ) for the images presented in Section 1.*

Usability: We believe that diffusion curves are a more natural drawing tool than gradient meshes. As mentioned previously, artists commonly use strokes to delineate boundaries in an image. Diffusion curves also allow an artist to evolve an artwork gradually and naturally. Gradient meshes, on the other hand, require careful planning and a good understanding of the final composition of the intended art piece. Most gradient mesh images are complex combinations of several individual — rectangular or radial — gradient meshes, often overlapping. All these decisions have to be made before the relevant image content can be created and visualized.

Topology: In some situations, the topology constraints of gradient meshes can be rather useful, for example when moving a gradient mesh to a different part of an image, or when warping the entire mesh. Such manipulations are also possible in our representation, but not as straightforward. For moving part of an image, the relevant edges have to be selected and moved as a unit. More importantly, without support for layering and transparency, it is difficult to ascertain how the colors of outer edges should interact with their new surroundings. A mesh warp could be implemented as a space warp around a group of edges.

Limitations

Diffusion curves attach color (and all other attributes) to lines. While this allows great flexibility, it can also pose a problem at *intersections*.

Currently, diffusion curves present a specific (although predictable and meaningful) behavior: the colors attached to the two intersecting curves essentially compete with each other, which creates a smooth color gradient after diffusion (Figure 4.20(a)). If this default behavior is undesirable, the user can correct it by either adding controls on each side of the intersection, or by splitting the curves in several parts with different colors (Figure 4.20(b)). Automating such behaviors would represent a powerful tool for easing user interactions.

Note however that this behavior is less of a problem than intersections in classical planar maps, because diffusion curves do not attach colors to regions. Therefore, colors are not “lost” when a region is not followed in the new configuration; when a line moves, the color follows.

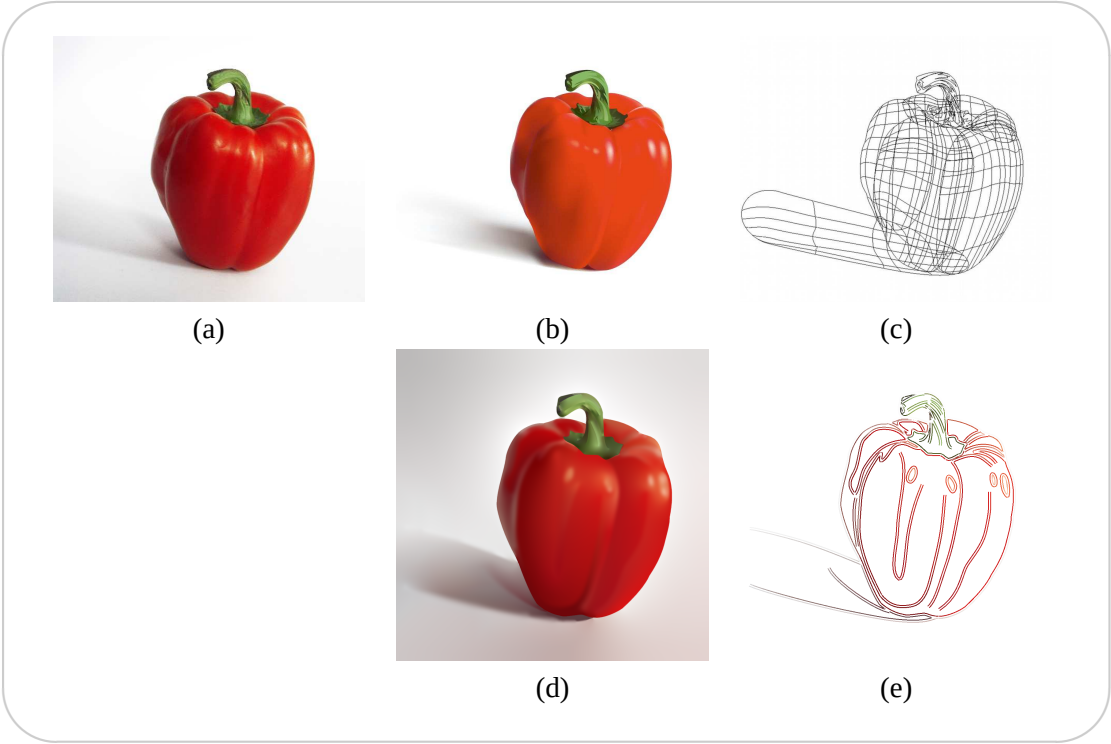


Figure 4.19: *Gradient Mesh comparison: (a) Original photograph; (b,c) Manually created gradient mesh (© Brooke Nuñez Fetissoff http://lifeinvector.com/), with 340 vertices (and as many color control points); (d,e) Our drawing created by manually tracing over the image; there are 38 diffusion curves, with 365 geometric, 176 left-color, and 156 right-color control points.*

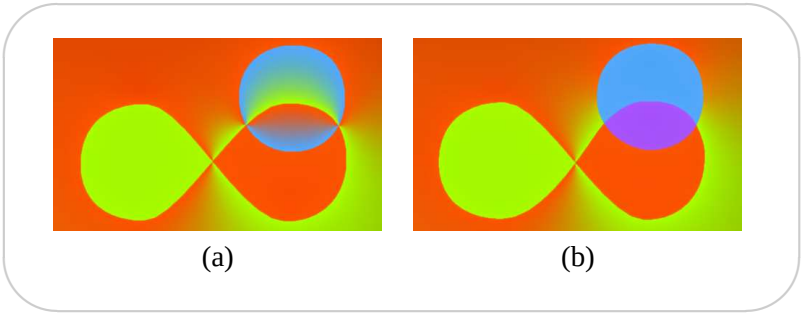


Figure 4.20: *The default behavior of diffusion curves at intersections (a) can be corrected by curve splitting and color editing (b).*

4.2 Texture

To evaluate the feasibility and utility of our system to create textured 2D images, we assigned a professional digital artist, proficient with modern 2D and 3D art creation tools, the task of creating a number of different texture types and designs, and of reporting on his experience with our system. The figures in Sections 3 and 2 represent some of this artist's work with our system.

Usability

Table 4.2 lists detailed timings for several textured artworks in this manuscript. Durations are listed separately for the creation of the support drawing, the texture draping, and the texel generation for that figure.

Figure	Support	Draping	Texels
Dress, Fig. 4.13	10 min(G)	30 min	5-30 min
Overview, Fig. 4.18	5 min (G)	20 min	40 min (G+C+N) for 2 texels
Sari, Fig. 4.14	1 h (G+C)	65 min	20 min (G+C)
Sari, Fig. 4.16b	1 h (G+C)	1h (N)	7 min (G+C)
Mermaid, Fig. 4.17	1 h (G+C)	45 min (UV)	5 min each (G+C)

Table 4.2: *Timings for selected Figures. Notations:* G–*Geometry;* C–*Color;* N–*Normals;* UV–*Tex. Coords.*

After using our system, we asked the artist to give us feedback about his experiences, both positive and negative. On the positive side, he noted that he found the *interactions* to be very intuitive. First, he would sketch out the texels and support drawings like he would design on paper. Then, he would fill in colors, as if painting them into the drawing. Adding normals was easy, especially in the automatic-inflation mode. He also liked the ability to tweak the texture placement by adjusting (u,v) coordinates. On the negative side, he complained about various *interface* aspects. For example, controls for settings normals and (u,v) coordinates were displayed in separate dialog boxes, and he would have preferred to adjust the parameters in-place. Initially, he found it too time-consuming to adjust (u,v) coordinates to realize his intentions. After we implemented the automatic (u,v) initialization (Sec. 3.3.4), he found the system much easier to use.

Comparisons

Alternative 2D Draping To compare our system against 2D warping approaches, we gave the artist the following task. He was to design two simple 3D reference shapes in a 3D mod-

eling package and texture them with a checkerboard texture (Figure 4.21 (a)). He then had to replicate the 3D rendering output as closely as possible with our system (Figure 4.21 (c)), and PHOTOSHOP's *Liquefy* tool (Figure 4.21 (b). Note, that this required not merely giving a good impression of shape, but to match each texel – a much more difficult task.

The two 3D shapes he designed were a simple *S-shape*, and a more complex *Seat-shape*. Using our system for the *S-Shape*, he spent $1'29''$ on the support drawing, $2'27''$ on setting normals, $4'25''$ on adjusting automatically placed (u,v) points, and $1'25''$ on adding and setting additional (u,v) points, for a total of just under ten minutes. Figure 4.21 (d) shows the normal control points (*diamonds*) and (u,v) control points (*circles*) the artist specified. He commented that much of the time spent on (u,v) adjustments was due to difficulties with not visualizing the texture-map in our texture-coordinate editor. Timings for the Seat-*shape* were similar, but added up to only $9'31''$, indicating that labor is proportional to the 2D complexity of the suggested shape, not its 3D complexity.

Using the *Liquefy* tool, the artist started with a rectangular checkerboard texture and spent $10'57''$ deforming it on the *S-Shape*, and $32'22''$ on the *Seat-Shape*. As evident in Figure 4.21 (b), the artist did not manage to control the exact contours of the shape. He commented that the warping approach was tedious due to the requirement of frequent and careful masking, and constantly changing the radius of the distortion tool.

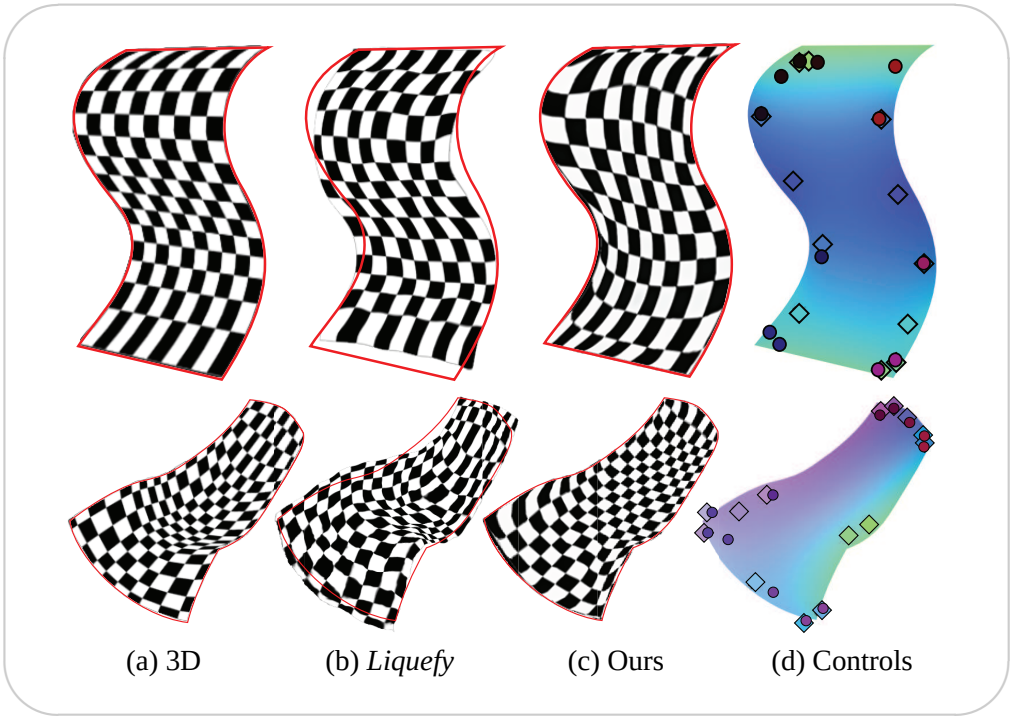


Figure 4.21: *Comparison with* Liquefy *Tool. Top row: S-Shape. Bottom row: Seat-Shape. (a) 3D result. (b) Liquefy Tool. (c) Our system. (d) Control Points & normal-map.*

3D Texturing We also asked the artist to compare our system with standard 3D texturing. As reference, the time to model and texture the above 3D *S-Shape* was about half that for replicating it using our system. The timings for the *Seat-shape* were comparable. As these numbers favor our system for complex 3D shapes, and since we envision our system being used in design workflows that are conceptualized in 2D, we performed a second test with the artist, complementary to the one above. Here, we replaced the sari texture in Figure 4.22 (a) with a checkerboard texture and asked the artist to create a 3D model to achieve the same image.

The artist took $3h45'$ to generate a 3D model of the draping. As Figure 4.22 (c) shows, this included only the sari but no background elements. He worked for an additional hour to adjust (u,v) coordinates using a professional skinning tool. This is compared to two hours total for our system, including geometry and color for background elements. When asked about his experience, he said that he favored 3D modeling for simple geometric shapes, but preferred the natural 2D design approach of our system for the complex shapes of drapings and folds that he created. He also pointed out that he was unaware of a straightforward 3D method to create the artistic design of the mermaid's hair in Figure 4.17.

Figure 4.22: *Comparison with 3D Modeling. (a) Draping template. (b) as (a) with checkered texture. (c) 3D model with checkered texture. (d) 3D model with texture from (a).*

Limitations

We acknowledge several limitations of our implementation. For very simple shapes, a 3D modeling system is quicker to use. In general, our system is not intended to replace accurate 3D systems, but rather to allow for quick and convenient prototyping of complex texture draping designs. Additionally, some aspect of our interface design proved to be cumbersome. While we hope to streamline the interface in the future, we feel this does not detract from the fundamental interactions, which an artist using our system quickly learned and mastered.

Currently, our system only supports regular or near-regular textures. In Section 3.3, we note that any texture-generation approach which outputs bitmaps can be used in our draping system.

We want to investigate several such approaches, and determine what additional user-parameters are necessary to control different types of textures.

In summary, we have designed a system with which an user can design a complex vector drawing, complete with color gradients, shading and macro-structure. Moreover, the user can employ the same interaction paradigm for the support-drawing design, as well as for texture definition and texture draping.

chapter 5

Vectorization of Color and Shape

Vector graphics, by their geometric definition, have some advantages over raster graphics. Most notably, they are resolution independent. Their content can be seen and printed at any desired size, without the upsampling artifacts visible in raster graphics. Also, vector graphics are more easily editable, and thus preferred for applications such as animation. But digital photographs are always recorded in raster format. Vectorization — the process of converting raster into vector graphics — is therefore a very useful process. This chapter studies the automatic vectorization of bitmaps into the diffusion curve vector representation.

To transform a bitmap image into a diffusion curve representation, we rely on bitmap edges. Edges are points in a digital image at which the image brightness changes sharply, and are as such the natural counterpart of diffusion curves in bitmaps. Edges also contain most of the visually important information present in an image [Lin98, Pal99] and can be leveraged to create a nearly complete representation of the image [Eld99]. The first vectorization step is therefore to extract edges from an image, along with their color and blur information (Section 1). The second step is to vectorize edge positions, colors and blur values, to obtain a diffusion curves set (Section 2). Limitations and comparisons with other vectorization methods are discussed in Section 3.

1 Data extraction

Many approaches exist to find edges and determine their blur and color attributes. We rely on a Gaussian-scale space approach, because this theory has been developed by the computer vision community for images where no a priori information is available, to extract perceptually important features. Using the scale-space analysis, we can create an *edge structure* that captures the degree of blur at each image discontinuity. Additionally, the Gaussian scale space can be used to derive the edge importance, automatically providing a hierarchial organization of the edge structure. This hierarchy can be transferred to the diffusion curves set and, as mentioned in Section 1.3, be used to simplify or modify a vector image, in accordance to the importance of each diffusion curve element.

In the following, we give a quick overview of the Gaussian-scale space, and describe how this theory is used to extract edges, their importance, and their profile (color and blur attributes), from a given image.

1.1 Gaussian scale space

Scale space methods base their approach on representing the image at multiple scales, ensuring that fine-scale structures are successively suppressed and no new element is added (the so-called "causality property" [Koe84]).

The motivation for constructing scale-space representations originates from the basic fact that real-world objects are composed of different structures at different scales of observation. Hence, if no prior information is available about the image content, the state-of-the-art approach for deriving the image structure is to use the successive disappearance of scale features to create a hierarchy of structures [Rom03].

Gaussian scale space is the result of two different research directions: one looking for a scale-space that would fit the axiomatic basis stating that "we know nothing about the image" and the other searching for a model for the front-end human vision [Mar82, FF87, Wan95, Rom03]. Since our purpose is to define a human-vision-like representation of an image content we have no a priori on, this scale-space fits our needs.

A scale-space is a stack of images of increasing scales. The basic Gaussian scale space is thus a stack of images convolved by Gaussian kernels of increasing variance[1]. In the general case, Gaussian derivatives of any order can be used to build the stack, allowing one to create scale-spaces of edges, ridges, corners, laplacians, curvatures, etc.

To obtain a diffusion curves representation, the important feature is the edge. We thus settle on studying the image structures represented by a hierarchy of edges in the Gaussian scale space. As edges are defined by gradient information, we only need to convolve the original image with Gaussian derivatives of order 1, one for each image dimension. These Gaussian derivatives G_x

[1] For numerical stability, one usually starts with a variance $\sigma_0 = 1$

and G_y are computed as follows:

$$G_x(x,y;\sigma) = g(y) \cdot g'(x) \text{ and}$$
$$G_y(x,y;\sigma) = g(x) \cdot g'(y)$$

with

$$g(i) = \frac{e^{-\frac{i^2}{2\sigma^2}}}{\sqrt{2\pi}\sigma} \text{ and } g'(i) = -\frac{e^{-\frac{i^2}{2\sigma^2}} i}{\sqrt{2\pi}\sigma^3}$$

where the width σ of the kernel corresponds to scale and $i \in \{x,y\}$. Given an input image I, we thus build two different scale spaces: an horizontal gradient $I_x = I \otimes G_x$ and a vertical gradient $I_y = I \otimes G_y$.

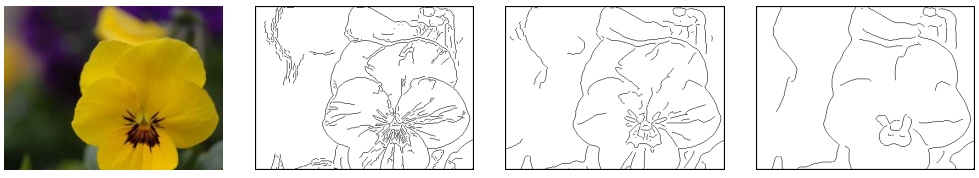
Contrary to classical approaches, which define the gradient in the luminance channel, we use the multi-channel color gradient method described in Di Zenzo [Zen86]. This means that gradients are computed for each color channel, and then combined to obtain a single magnitude value in each pixel. This allows us to detect sharp color variations in iso-luminant regions of the image, where a luminance gradient would fail.

1.2 Structure extraction

Starting from the multi-scale gradient values, we extract the edge-based image structure S corresponding to the edges, their importance and profile (color on each side and blur).

Edge extraction

From the first-order Gaussian derivative scale space, we want to build a hierarchy of edges holding structural importance. We first extract edges at all the available scales in order to get the richest possible information. For this task we use a Canny edge detector [Can86] on the multi-channel color gradient image: it is a state-of-the-art edge detection method that processes the Gaussian derivative information at each scale to give thin, binary edges. Its main quality resides in using hysteresis thresholding that results in long connected paths and avoids small noisy edges (see Figure 5.1).



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

Figure 5.1: *Edge importance. (a) The input image. (b-d) Canny edges at increasing scales.*

After applying the Canny detector, we are left with a multi-scale binary mask C_σ that indicates at each scale the edges locations. Figure 5.2 illustrates such a typical edge scale-space for a simple 1D example. Due to the nature of Gaussian scale-space, three different cases can occur: (a) an edge exists and suddenly drops off at a higher scale; (b) two edges are coming toward each other and collapse at a higher scale; (c) some "blurry" edges only appear at a higher scale. To simplify further computations, we "drag" edges corresponding to case (c) down to the minimum scale and note C^*_σ the resulting multiscale edge mask.

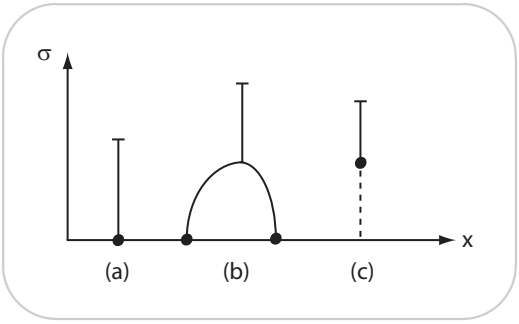


Figure 5.2: *Three different events in a 1D Gaussian scale-space: (a) an edge drops off at a high scale; (b) two edges collapse ; (c) a blurry edge is created. In the last case, we drag the edge down to the finest scale for convenience.*

Edge importance

As shown in Figure 5.2, there is a great deal of coherence along the scale dimension in the multi-scale edge representation. The main idea behind scale-space techniques is to try to extract this coherent *deep structure*, by linking edges at different scales. In particular, because of the causality property of Gaussian scale-space, an edge that disappears at a given scale will not reappear at a higher scale; hence an important measure of structure along scale is *lifetime*, as edges that live longer will correspond to more stable structures.

Unfortunately, extracting an edge lifetime is not trivial, since edges move in Gaussian scale-space (this corresponds to Figure 5.2 case (b)). This motivated edge focusing techniques, that track edges at increasing scales [Gos94]. In this manuscript, we take an alternative approach which revealed simpler to implement: instead of considering each pixel p belonging to an edge, we consider its projected point $\mathcal{P}_\sigma(p)$ onto the closest edge at scale σ (we use a distance field for this purpose). We can then define the membership of any pixel $m_\sigma(p)$ as the binary function that indicates whether p can be considered to belong to an edge at scale σ:

$$m_\sigma(p) = \begin{cases} 1 \text{ if } ||\mathcal{P}_\sigma(p) - p|| < T_\sigma \\ 0 \text{ otherwise} \end{cases}$$

The choice of the threshold distance T_σ is essential to get a good approximation for our membership function. Bergholm [Ber87] proved that the edge shifting is less than a pixel when the scale σ varies by less than 0.5. Therefore, we increase our σ values by $\Delta\sigma = 0.4$ at each scale

(a) (b)

Figure 5.3: *Edge importance. (a) The input image. (b) The lifetime measure reflects the importance of edges: "older" edges correspond to more stable and important structures.*

and use $T_\sigma = \sigma/\Delta\sigma$. This approach is similar in spirit to the morphological linking method of Papari et al. [PCPN07].

Finally, using membership for linking purpose, we compute the lifetime $L(p)$ at each edge pixel p in the finest scale by summing up membership values (Figure 5.3). Considering the successive scale values $\sigma_i, i \in 1..N$, where N is the size of our scale-space stack, we write lifetime as:

$$L(p) = \arg\min_i \{\sigma_i | m_{\sigma_i}(p) = 0\}$$

This can be seen as a simpler, easier-to-manipulate version of Lindeberg edge strength measure [Lin98]. Lifetime is thus considered as a measure of structural importance, and can supply the diffusion curve representation with its *importance* value (Section 1.3).

Edge profile

In the previous section, we mainly relied on edge locations and their persistence along scale. Another concern is to deal with their *profile* (color values and degree of blur). In this work, as in previous work [Lin98, EG01b], we rely on a simple assumption: the profile of an edge gradient is modeled as the convolution of a Dirac (its location and color difference between the two sides of an edge) with a spatially varying Gaussian kernel (its blur). For instance, in a photograph with depth-of-field, out-of-focus edges are blurry (with a wide profile) while in-focus edges are sharp (with a thin profile).

Our second measure of structure then consists, for each edge, in finding the *best scale* that locally corresponds to its blur.

The best scale search is another form of *deep structure* that has been studied by Lindeberg [Lin98]. Following his approach, we first compute a normalized gradient magnitude scale-space by $||\nabla I|| = \sqrt{\sigma(I_x^2 + I_y^2)}$. The best scale $B(p)$ at an edge pixel p is then identified as the one which gives the first local maxima along the scale axis in this normalized gradient magnitude stack.

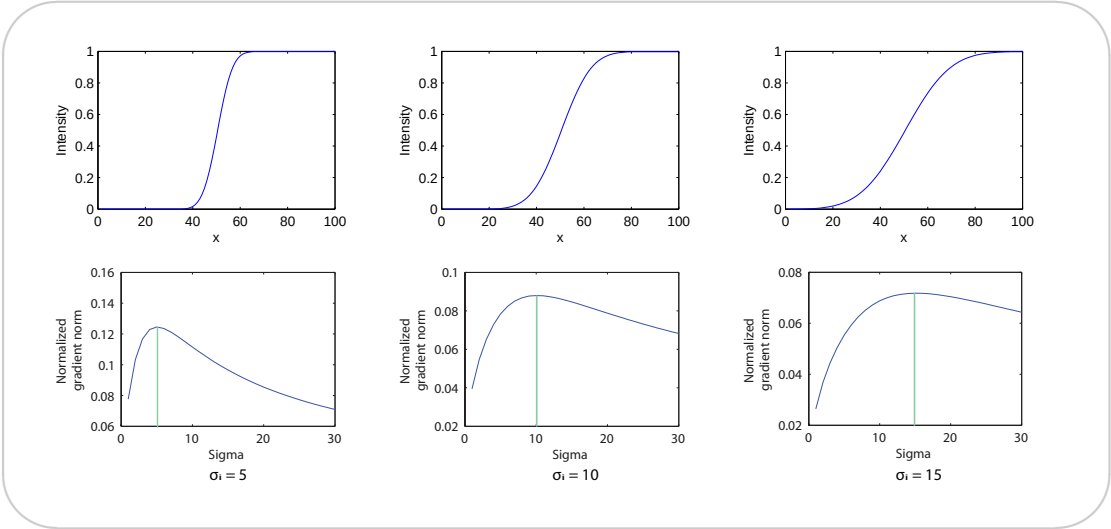


Figure 5.4: *Best scale estimation. Top: 1D edges blurred with* $\{\sigma_i\} = \{5, 10, 15\}$. *Bottom: normalized gradient magnitude scale space proposed by Lindeberg. The best-scale measures (the local maxima) are at the blur scale* $\{\sigma_i\}$*, hence representing well each edge profile.*

But as with lifetime computation, we need to link "moving edges" at different scales using the projection operator $\mathcal{P}_\sigma$ again: $||\nabla I(p)|| = ||\nabla I(\mathcal{P}_\sigma(p))||$. Figure 5.4 shows how best scales can be well estimated for edges of increasing blur.

We are now able to "re-blur" the edges using the best scale. Moreover, we use this ideal scale also to localize edges, because it is at that scale that the edge shape is closest to the shape perceived by the human vision. It should be noted that very blurry edges are difficult to detect and parameterize accurately. In our system we find that very large gradients are sometimes approximated with a number of smaller ones.

After the best scale search, we are left with an edge map, which contains the edge locations and the blur values for the edge pixels. One last processing step is needed to obtain the full edge profile: colors on both sides of the edge must be extracted explicitly. To this end, we rely on the blur values to know how far from the edge position the unblurred colors are. We connect pixel-chains from the edge map and proceed to sample colors in the original image on each side of the edge in the direction of the edge normal. In practice, the gradient normal to the edge is difficult to estimate for blurry edges, so we use the direction given by the normal of a polyline fitted to each edge. For an estimated blur σ, we pick the colors at a distance $3 \cdot \sigma$ from the edge location, which covers 99% of the edge's contrast, assuming a Gaussian-shaped blur kernel [Eld99]. While the $3 \cdot \sigma$ distance ensures a good color extraction for the general case, it poses numerical problems for structures thinner than 3 pixels ($\sigma < 1$); in this particular case, color cannot be measured accurately.

At the end of the data extraction, the bitmap image is represented by edges organized in pixel chains, and with values of blur, left- and right-side colors, and importance, attached to all pixels belonging to an edge.

2 Conversion to diffusion curves

This pixel-based data is transformed into diffusion curves by a vectorization process. For vectorization of edge positions, we take inspiration from the approach used in the open source Potrace©software [Sel03]. The method first approximates a pixel chain with a polyline that has a minimal number of segments and the least approximation error, and then transforms the polyline into a smooth poly curve made from end-to-end connected Bézier curves. The conversion from polylines to curves is performed with classical least square Bézier fitting based on a maximum user-specified fitting error and degree of smoothness. For attribute vectorization, we use the same method as the one described for extracting colors in image tracing (Section 1.2).

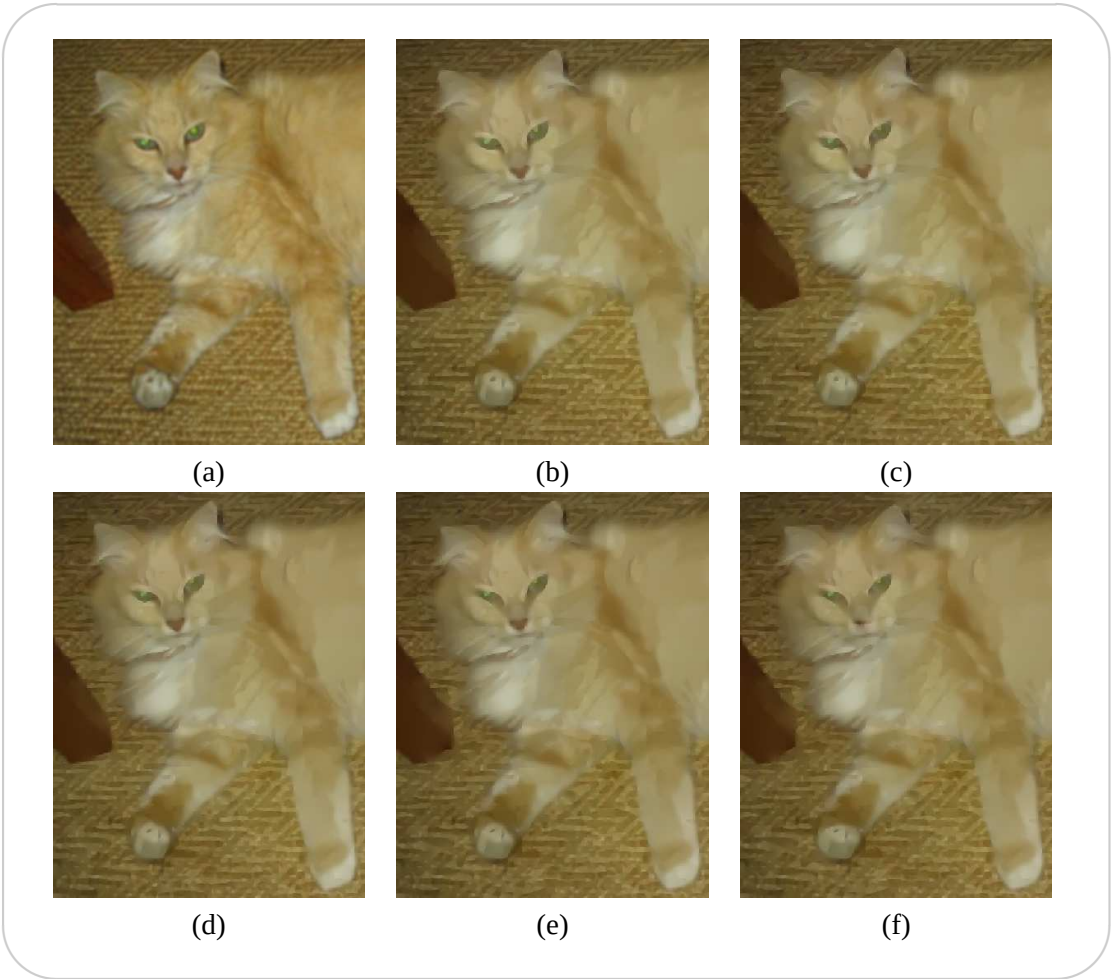


Figure 5.5: *Vectorization results with different fitting errors for the edge positions. (a) Original image. (b) The image reconstruction using only the extracted data (Section 1). (c)–(f) Result after conversion to diffusion curves, using different fitting errors for the edges: (c) 1; (d) 5; (f) 25; (e) 50;*

Several parameters determine the complexity and quality of our vectorized image representation. For the edge geometry, the Canny threshold determines how many of the image edges are to be considered for vectorization; a despeckling parameter sets the minimum length of a pixel chain to be considered for vectorization; and finally, two more parameters set the smoothness of the curve fitting and the fitting error. For the blur and color values, two parameters are considered: the size of the neighborhood for eliminating outliers, and the maximum error accepted when fitting the polyline. For most images in this manuscript, we use a Canny high threshold of 0.82 and low threshold of 0.328, we discard pixel chains with less than 5 pixels, we use a smoothness parameter of 1 (Potrace default) and we set the fitting error to 1, so the curve closely approximates the original edges (Figure 5.5 shows results with different fitting errors). For attributes, we consider a neighborhood of 9 samples, and the maximum error accepted is 2 blur scales for the blur and 30 CIE L*a*b* units for colors. Figure 5.6 gives an example of image vectorization using the proposed method.

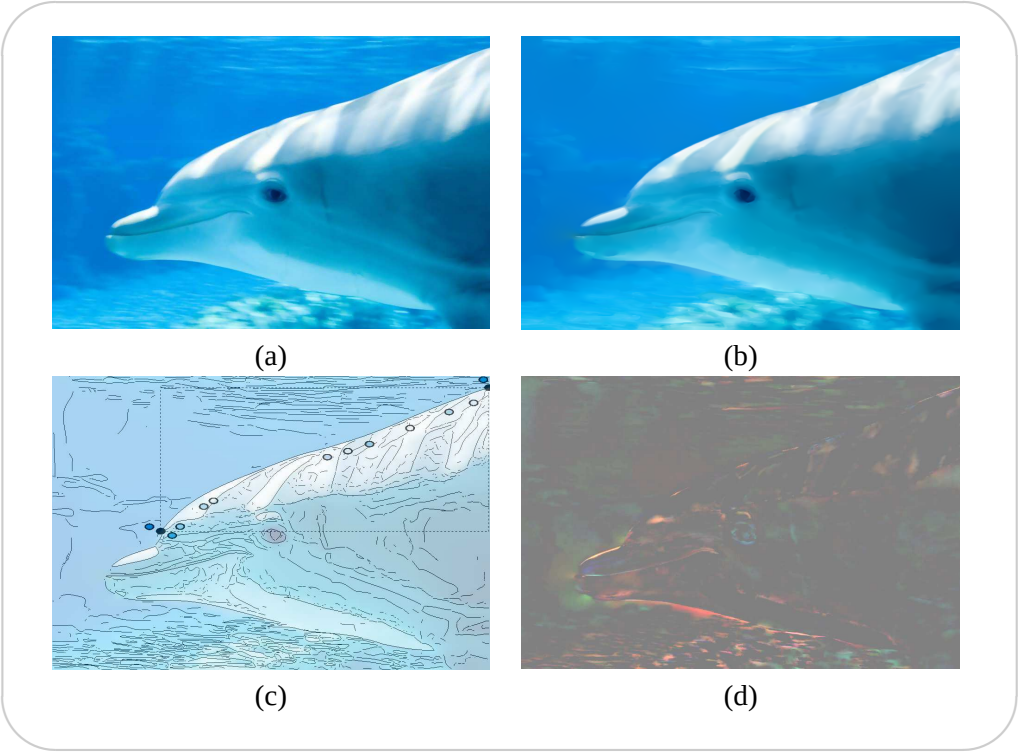


Figure 5.6: *Example of our reconstruction: (a) original image; (b) result after conversion into our representation; (c) automatically extracted diffusion curves; (d) RGB difference between original and reconstructed image (amplified by 4); note that the most visible error occurs along edges, most probably because, through vectorization, we change their localization.*

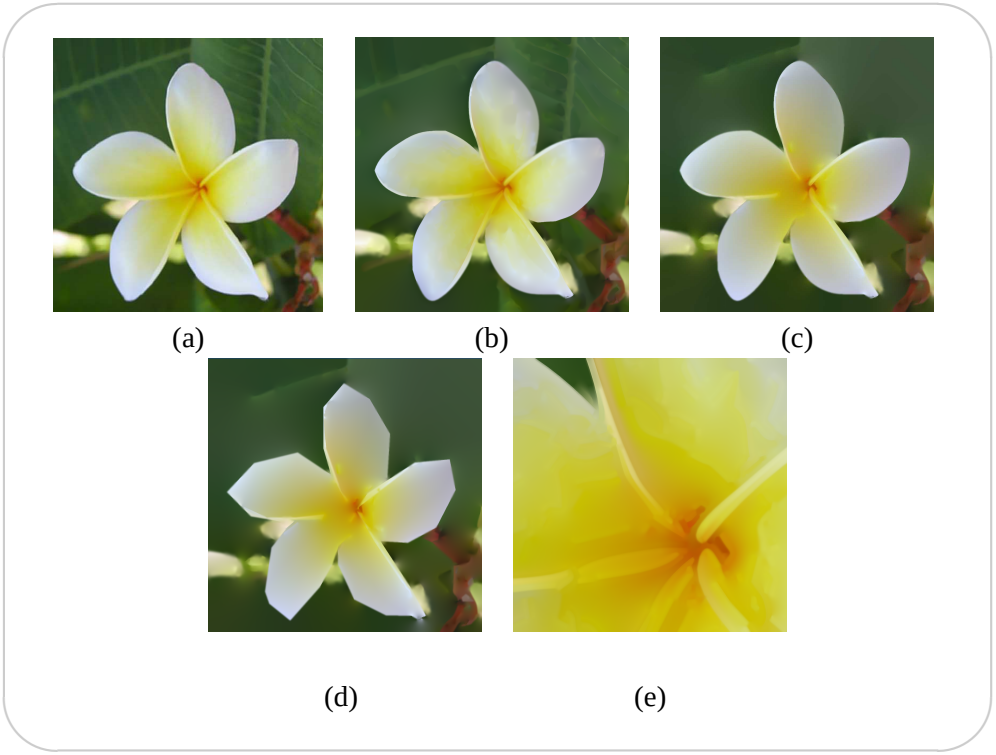


Figure 5.7: *Stylization effects: (a) Original bitmap; (b) Automatic reconstruction; (c) Reconstruction simplified by removing edges with low lifetime; (d) Global shape stylization applied to (c); (e) Enlargement of (b).*

3 Discussion

A diffusion curve vectorized image benefits from the advantages of traditional vector graphics: zooming-in preserves sharp transitions (Figure 5.7 (e)); curve shapes and attributes can be easily modified to obtain effects such as that presented in Figure 5.7 (d); the importance measure can be used to adjust preservation of detail (Figure 5.7 (c)).

Compared to region-based vectorization approaches — such as the vectorization methods proposed by Adobe Live Trace© and Lecot and Lévy [LL06] — our contours need not be closed boundaries. This in turn results in smooth color variations between two curves, and avoids the posterization effect typical to region-based methods (see Figure 5.8).

Gradient meshes [SLWS07, PB06, LHM09] can represent objects with complex variations of smooth colors. Diffusion curves achieve similar vectorization results (see Figure 5.9 for a comparison). But we believe that diffusion curves are a “lighter” representation, better equipped for subsequent manipulation of the vectorized result.

One limitation of our vectorization approach comes from the approximations made during the process of extracting diffusion curves from a bitmap. Canny edge detection, blur detection, and data vectorization can all introduce sampling errors. Especially the blur detection is known to

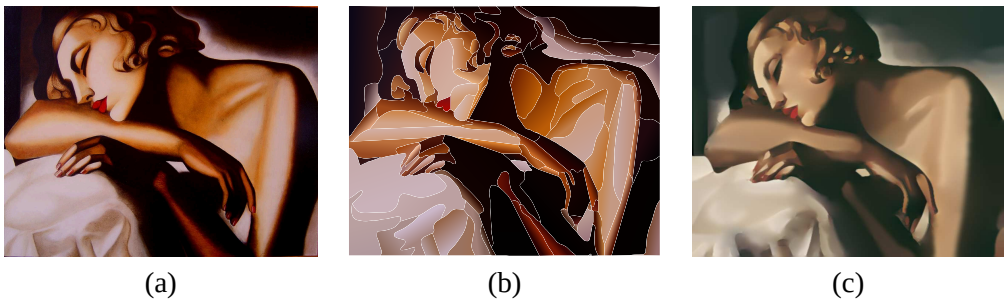


Figure 5.8: *Comparison with Ardeco [LL06]: (a) original image; (b) the vectorization result. Note that, while the gradient inside regions is well approximated, sharp transitions between regions are noticeable. Image taken from [LL06]. (c) Our result. Note that the smooth color transition is preserved both along and across the geometric curves.*

Figure 5.9: *Comparison with Gradient Meshes [SLWS07]: (left) Original bitmap; (middle) Gradient Meshes; (right) Diffusion curves.*

be a hard problem, and prone to error.

Another limitation, common to all vector graphics, occurs in images or image regions that contain many small color or luminance variations, such as textures. In practice, most of the visual information of highly textured regions is captured by the automatic conversion, but imprecision occur when the texture is composed of many small structures (small compared to the distance d defined in Section 3.2.1). Moreover, the large amount of curves required to represent textures

makes a vector representation inefficient and difficult to manipulate (Figure 5.10.

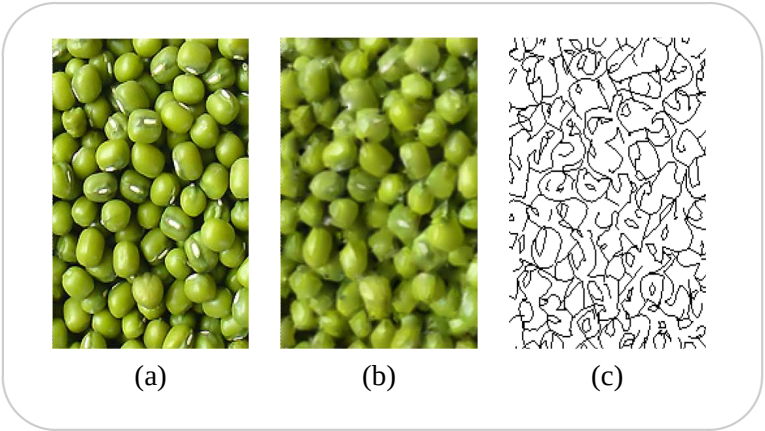


Figure 5.10: *Limitation: diffusion curves have precision issues with thin structures (such as textures). (a) Original bitmap; (b) Converted result; (c) Diffusion curves.*

One possible solution for texture vectorization could be to rely on recent compute vision advancements [GZW07] or user input to separate detail from important structure. Texture could then be synthesized at the required magnification level and reapplied over the smoothly varying color of the vectorized structure.

chapter 6

Photograph Manipulations via Raster Diffusion Curves

By vectorizing a bitmap, new manipulation options are made available, especially because each discontinuity can now be individually edited, and fine tuning is possible for colors and blur. However, the vectorization process is inextricably linked to simplification of shape, color gradient and blur variations. It is thus interesting to study the image manipulation capabilities of the bitmap version of the diffusion curve[1].

Our approach is to rely on the hierarchial structure provided by the bitmap edges together with their importance (Section 5.1), and to guide user manipulations by considering the relevance of each structure edge when simplifying or enhancing the content (Figure 6.1).

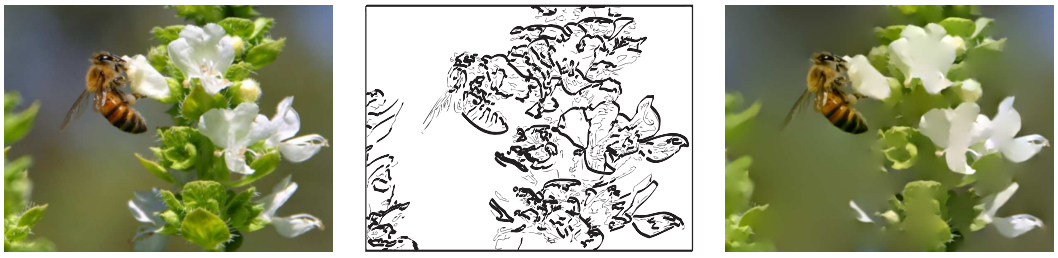


Figure 6.1: *Our approach takes as input a bitmap image (left), and allows a user to manipulate its structure in order to create abstracted or enhanced output images. Here we show a line drawing with line thickness proportional to their structural importance (middle), and a reconstruction of color information that focuses on the bee and removes detail around it (right).*

[1]The bitmap structure manipulation was presented in our paper [OBBT07] at NPAR 2007. It was a work done in collaboration with Adrien Bousseau, Pascal Barla and Joëlle Thollot.

One important difference with the vectorization process is that the edge color constraints are not used for bitmap manipulations; instead, gradient values are placed on the edge. Two main reasons are behind this:

– First, without the support of edge poly-lines, color extraction needs to use discrete gradient normals, and these could prove unreliable. While inconsistencies in color sampling is not detectable if colors conserve their original pixel position, color errors are problematic when displacing edges (and therefore colors).

– Second, using only gradient constraints has the advantage of seamless editions [PGB03], even though with the downside of unexpected color results (Figure 6.2).

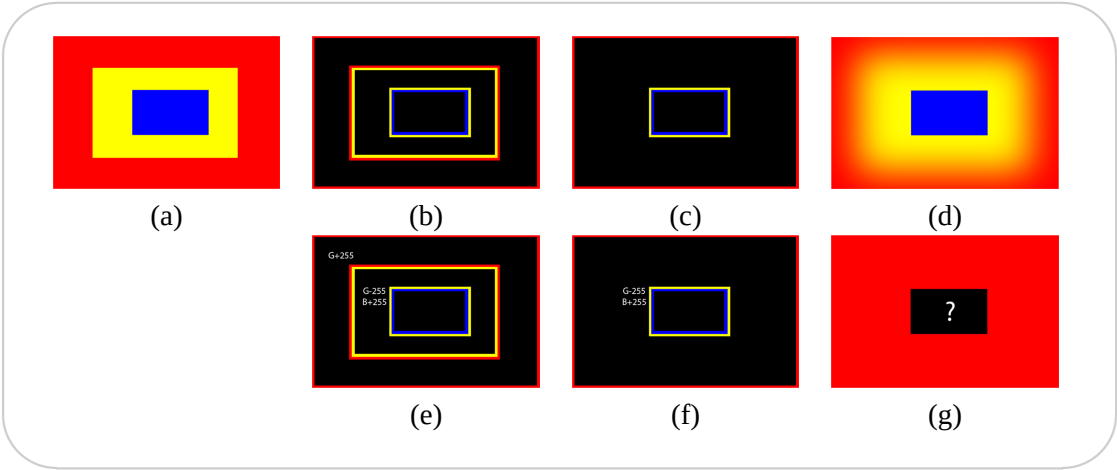


Figure 6.2: *Color constraints vs. gradient constraints: (a) Original image; (b) Corresponding edges, with color constraints on each side; (c) An edge is removed; (d) Diffusing the remaining color constraints creates a color gradient; (e) Gradient constraints retain* transitions *of color, rather than the color itself; (f) A gradient constraint is removed; (g) The diffusion fills the empty space with an uniform color, but the remaining gradient variations can create an unpredictable color.*

In such case, the bitmap diffusion curves manipulation system consists of three steps: (1) extraction of the edge structure S, as described in Section 1; (2) the use of S as a high-level control for user-defined image manipulations, and output a manipulated bitmap structure S^*; (3) reconstruction of an image O from the manipulated edge set using the Poisson equation. Figure 6.3 sums up our approach for bitmap manipulations.

In the following, we mainly present the gradient reconstruction step, because it is specific to the bitmap approach. Several edge manipulation techniques are presented in Section 2.

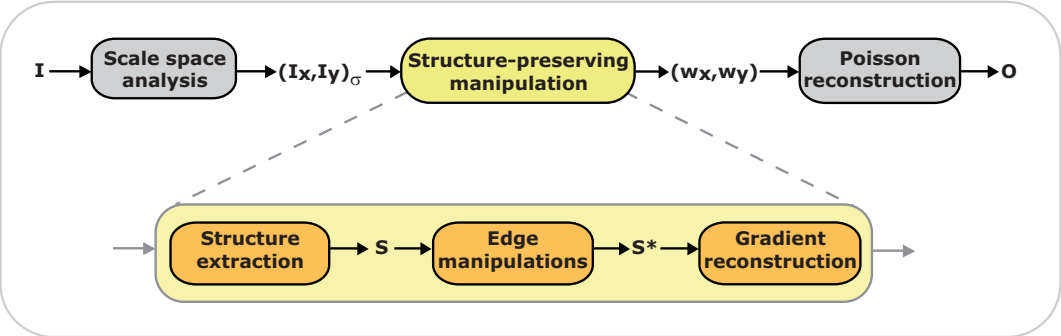


Figure 6.3: *Overview of our method.*

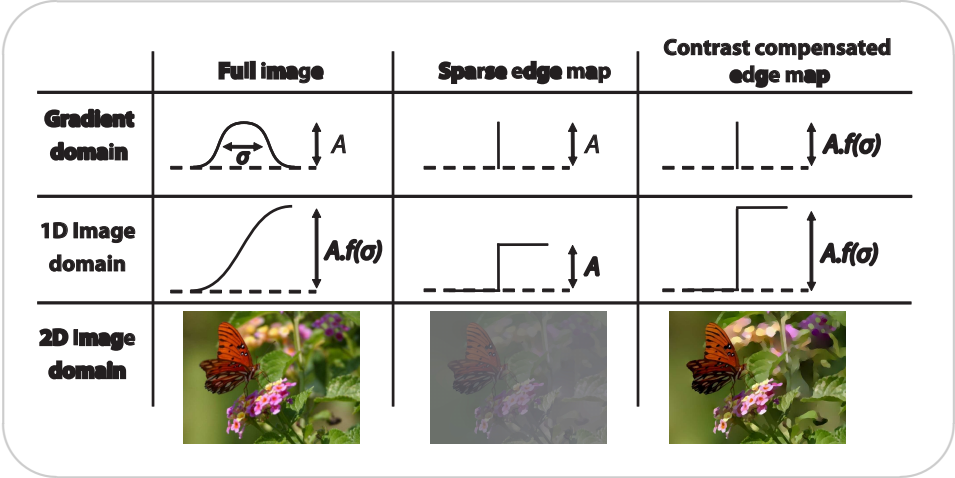


Figure 6.4: *Gradient reconstruction with different profiles.* $\mathbf{f}(\sigma)$ *is our contrast value compensation function, that depends of the detected blur value.*

1 Poisson reconstruction using only gradients

Using our manipulated set of edges S^*, we wish to reconstruct the corresponding image by solving a Poisson equation. Considering only gradient constraints, this translates into building a vector field $\mathbf{w}$ that corresponds to our new edges.

We propose to use the scale space information to estimate the original gradient profiles and correctly reproduce the contrast and blur of the input image (Figure 6.4). However, taking only original gradient values at edge locations as suggested by Perez et al. [PGB03] results in a gradient field that does not capture the whole original contrast, nor the original blur (Figure 6.5, (a) and (b)). This is because we only consider the central value of the profile, loosing all its surrounding information.

A simple solution to the contrast problem would be to apply a histogram equalization on the reconstructed image to match the original contrast. However the very low dynamic range of the reconstructed image leads to strong quantization artifacts (Figure 6.5(c)).

We thus need to take into account our knowledge of edge profiles to compute the correct contrast (Figure 6.4). Our model of an edge represents blurry edges that appear in the input image I as the convolution of a step function H by a 2D Gaussian kernel G_B, where B is the local best scale. When we measure I_x (resp. I_y) at scale B on edge locations, we get the following contrast values:

$$I_x \;=\; H \otimes G_B \otimes \frac{\partial G_B}{\partial x} \;=\; H \otimes \frac{\partial G_{B_2}}{\partial x} \;=\; \frac{\partial H}{\partial x} \otimes G_{B_2}$$

with $B_2 = \sqrt{2B^2}$. However, to recover the original contrast value of the profile, we are precisely interested in the value of $\frac{\partial H}{\partial x}$. This corresponds to the deconvolution of I_x (resp. I_y) by G_{B_2}. Unfortunately, deconvolution is known as an ill-posed problem, particularly sensitive to noise and quantization [Rom03]. To avoid this problem, we propose to simplify our model for the sake of contrast correction: we replace the 2D Gaussian derivative by a 1D Gaussian derivative $\widetilde{G}_x = g'(x)$. This way, we can derive an analytical solution for the correction problem.

We model a directional edge gradient $I_{\{x,y\}}$ as the 1D convolution of a step function H of amplitude A by a Gaussian kernel g_σ and a Gaussian derivative g'_σ, resulting in:

$$\begin{aligned} I_x(0) \;&=\; \big(H \otimes g_\sigma \otimes g'_\sigma\big)(0) \;=\; \big(H \otimes g'_{\sqrt{2\sigma^2}}\big)(0) \\ &=\; \textstyle\int_{-\infty}^{+\infty} H(t) \cdot g'_{\sqrt{2\sigma^2}}(-t)dt \;=\; \int_{0}^{+\infty} A \cdot g'_{\sqrt{2\sigma^2}}(-t)dt \\ &=\; A \cdot g_{\sqrt{2\sigma^2}}(0) \;=\; \tfrac{A}{2\sigma\sqrt{\pi}} \end{aligned}$$

For each edge pixel p, we only need to multiply the gradient value found in I_x (resp. I_y) by $2B(p)\sqrt{\pi}$. This correction gives a final contrast close to the original one, and we find that our approximation works well in practice, with no visible artefacts (see Figure 6.5(d)).

Finally, even if using edge locations and correcting their contrast does give a convincing result, blurry edges become sharp in the reconstructed image. Therefore, we also re-blur the edges, as seen in Figure 6.5(e). This process remains optional as the sharp result provides an interesting cartoon style.

2 Applications

The edge structure $S = \{C^*_\sigma, L, B\}$ — where C^*_σ are multi-scale Canny edges together with their lifetime L and best scale B — can be manipulated in various ways. The main idea is to select a subset E of the multi-scale Canny edges C^*_σ according to lifetime L. After manipulation, we are thus left with a new, simpler structure $S^* = \{E, B\}$. Based on this schema, we propose three image manipulations, that can be seen as variations of recently proposed methods of editing in the gradient domain. Our contribution is to use the high-level structural information provided by our approach to guide these gradient manipulations.

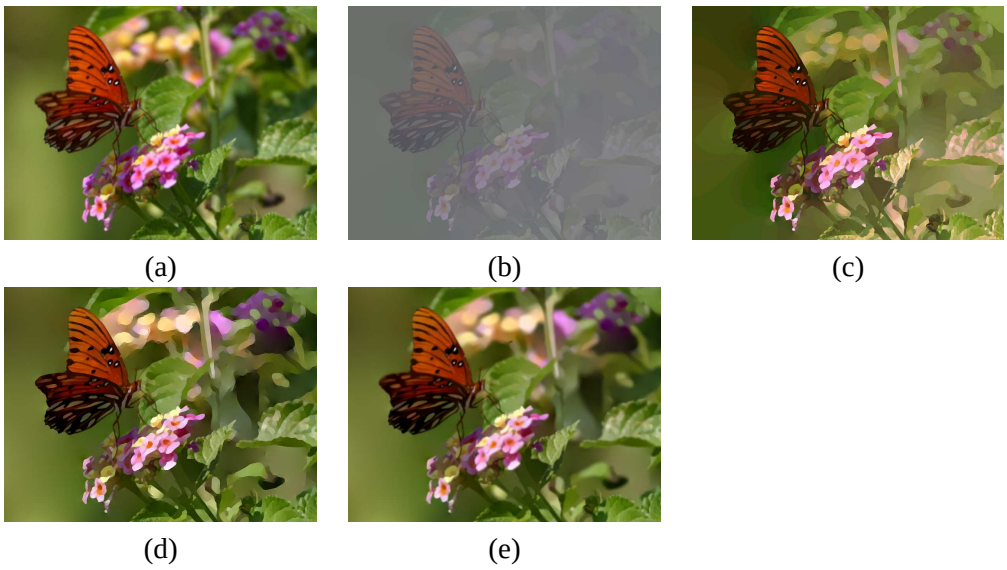


Figure 6.5: *Gradient reconstruction. (a) Input image. (b) Reconstructed image using only the original gradient values at edge positions. (c) Reconstructed image with histogram equalization. Note the quantization artefacts. (d) Reconstructed image using contrast correction. Note that blurry edges become sharp if the profile is not taken into account. (e) Full reconstruction using contrast correction and re-blurring.*

2.1 Detail removal

We use the lifetime information as a threshold value to seamlessly remove details while keeping important structures. Such image editing operations are similar to the seamless cut and paste operations proposed by Perez et al. [PGB03] and Elder et al. [EG01b], except that we provide a high level control to the user, who has only to select the desired level of detail (Figure 6.6).

2.2 Multi-scale shape abstraction

We propose a shape abstraction method that adapts the level of abstraction to the scale of the features in order to preserve the informative content of the picture. In practice, we select for each edge its last available version in the scale space using lifetime. As shapes become more and more smoothed along scales due to the Gaussian filter, relevant structures will have increasingly rounded shapes while details will keep their original silhouettes.

In opposition to previous approaches [DS02] that remove texture details and abstract shapes at the same time, our approach selects for each edge (including edges belonging to texture details or other small elements) the shape of its last scale. Hence, our approach still keeps most of the meaningful structural information, while simplifying its shape, as seen in Figure 6.7.

Figure 6.6: *Detail removal: (a) original image, and (b-d) several levels-of-detail automatically generated by our method.*

This application can be seen as a fusion of multi-scale images, similar in spirit to other image fusion methods like the ones of Agarwala et al. [ADA+04] and Raskar et al. [RIY04].

2.3 Line drawing

The edge lifetime information offers a powerful high-level parameter for any line drawing algorithm. Figure 6.8 presents the rendering of vectorized edges with a different width to enhance important structures from details. Figure 6.1 (middle) also shows an example of this application.

2.4 Local control

In order to offer a local control to the user, each image manipulation can be weighted by a gray-level map indicating the desired amount of abstraction (Figure 6.9). This mechanism is essential to be able to focus on a given zone of the input image, and efficiently grabs visual attention. We take advantage of the Poisson reconstruction to obtain seamless transitions

Figure 6.7: *Shape abstraction: (a) original image, and (b) our shape abstraction result. Notice how the thin details are kept, while shapes of bigger objects are abstracted (e.g. the poles).*

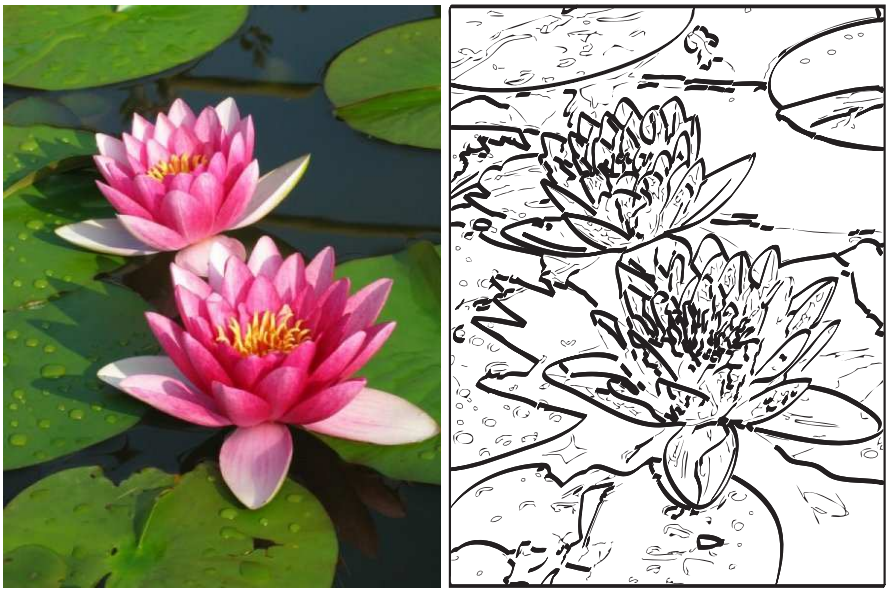


Figure 6.8: *Vectorized edges, with a larger width for relevant structures (i.e. those having greater lifetime).*

between regions of different weights.

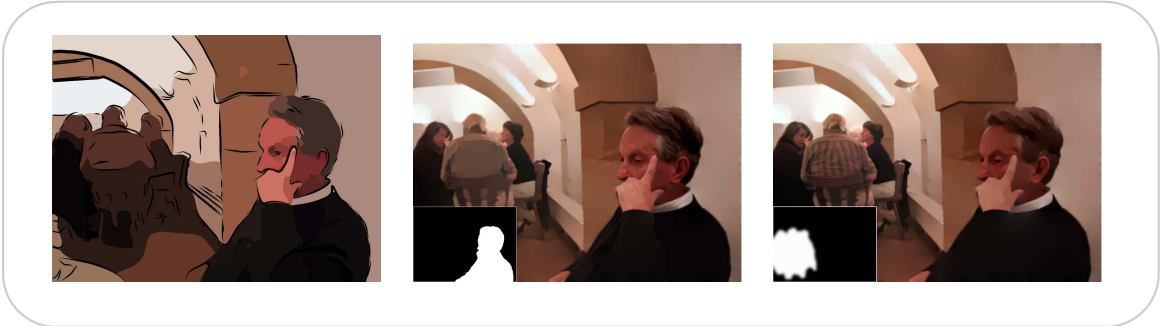


Figure 6.9: *Local control: original image of DeCarlo et al. [DS02] and our results for two different user-specified control maps.*

3 Discussion on bitmap diffusion curves

A number of previous techniques focused on creating enhanced or abstracted renderings from arbitrary photographs.

Generally the previous methods manipulate an image globally without using the image structure [WOG06], or rely on the user to define what is important [WXSC04, KCC06, WLL+06]. As a result, the content either cannot be controlled, or its control involves tedious user interactions. We propose a method that automatically extracts the relevant structural information, and can be subsequently used to enrich automatic stylization systems or to assist the user in her task.

The interest of expressing the image content with an automatically created structure is well illustrated in Figure 6.10. Here we show a failure case of Winnemöller et al.'s abstraction approach [WOG06]. Although their method gives convincing results in many cases, this specific example shows how they cannot get rid of high-contrast texture lines without abstracting the cat too far. Manual approaches would, in this case, require the user to paint over the entire textured region. In contrast, our approach allows us to simply remove detail edges regardless of their contrast.

Previous work made use of Gaussian scale space [Her98] or saliency maps [CH05, CH03] in order to guide painterly stylizations. However, saliency maps identify image regions that already grab visual attention in the original image, and using them to guide stylization will only preserve these attention-grabbing regions. In contrast, our approach extracts a structure that allows the user to *intentionally* manipulate the image, possibly modifying its attention focus (i.e. changing its subject, see Figure 6.9 middle and right), and hence conveying a particular message.

DeCarlo and Santella [DS02, SD04] were the first to use a meaningful visual structure in photo abstraction. They use color regions as structural units and create their hierarchy of regions from a pyramid of down-sampled versions of the image. But for coarser-level regions the shape simplifies and the borders move slightly. Therefore, there is no perfect overlap between

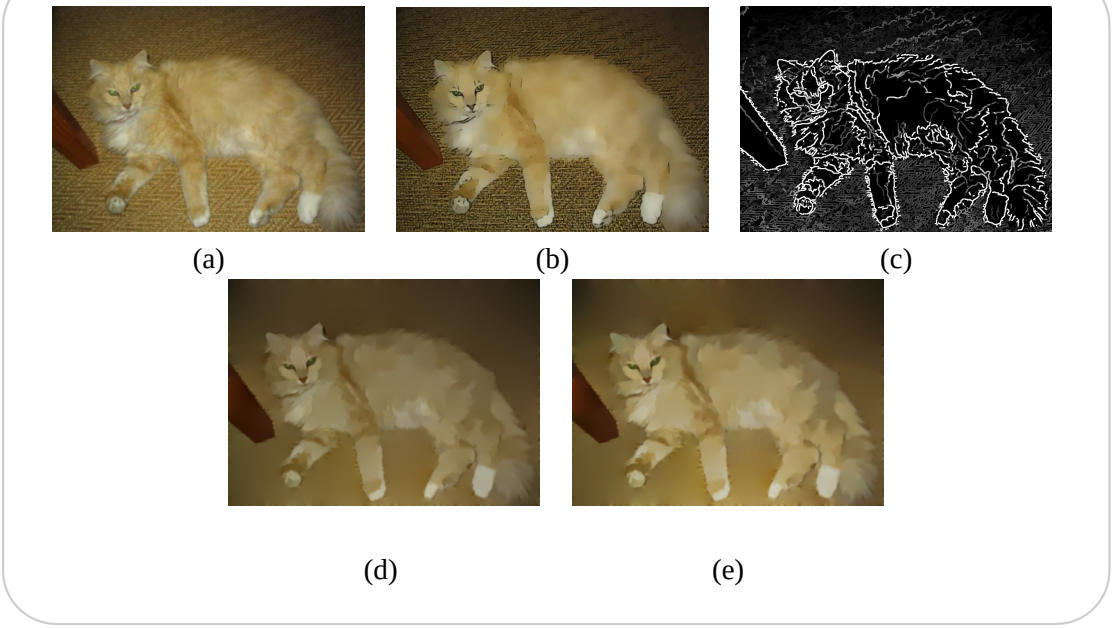


Figure 6.10: *Comparison with the failure case of Winnemöller et al. [WOG06]. (a) Original picture. (b) Winnemöller et al. abstraction failure: note how the carpet details are preserved while the fur is abstracted away. (c) Our lifetime map. (d) Our detail removal abstraction preserves the cat structure and abstract the carpet. (e) We apply histogram equalization as a post-process to fine tune contrast.*

Figure 6.11: *Comparison with the DeCarlo et al. [DS02]. (a) Original picture. (b) DeCarlo et al. results exhibit flat color regions with shape simplification (c) Our result simplifies the image while keeping smooth color variations and original shapes.*

finer and coarser regions. When mixing different levels of detail in the same image, this becomes problematic because the information at different scales has to be unified in a single image. Moreover, while DeCarlo et al.'s method couples simplification of shape with detail suppression, ours allows to remove details *without* necessarily simplifying shapes (as shown in Figure 6.11).

Bangham et al. [BGH03] extend DeCarlo and Santella's work by improving the region segmentation. Their region hierarchy is based on a morphological scale-space and has the advantage of preserving region shapes. But since only the region size is considered, and not its contrast, they tend to eliminate visually important cues that have a high contrast but small size.

The use of Poisson reconstruction is also an important advantage for our editing methods. While other diffusion methods will try to blur unwanted details, a Poisson approach will simply ignore it in the reconstruction (by not considering the color variations for that detail). This is again well illustrated by the example in Figure 6.10, since the texture lines do not appear in our image.

Future work We see our approach as a starting point for any subsequent stylization. As such, one possible venue for future work resides in developing such stylized renditions that take advantage of structural information. As an example, we created two preliminary results, shown in Figure 6.12: a drawing, and a watercolor. There are many connections to establish between style parameters and structure information, and we hope this work will motivate future research along this direction.

Figure 6.12: *Different stylizations obtained from our abstracted images, in a drawing and watercolor style.*

Finally, Poisson image editing is a powerful tool for raster image manipulations [PGB03]. However, as we have discussed in the course of this chapter, relying on gradients alone to perform manipulations can lead to unexpected color results (for example, pasting a yellow on black circle onto a blue background might lead to the circle becoming white). On the other hand, using color constraints everywhere will have the undesirable result of preserving colors of deleted elements; in the circle example, it will preserve the black border of the circle. It is therefore interesting to combine these two methods, and explore methods of locally choosing whether to use gradient or color during the editing process (to obtain an yellow circle on a blue background). We believe that the diffusion curve bitmap structure, by its localized nature, could support such editing tools.

chapter 7

Conclusion

This manuscript proposes contours as a means to represent, create and manipulate digital images. The distinguishing characteristic of this novel image representation — called diffusion curves — is that it uses line drawings as a base. Line drawings record the discontinuities observed in a scene, and in a similar way diffusion curves correspond to the discontinuities present in an image. This thesis demonstrated that the diffusion curve principle (editing an image via its discontinuities) can be leveraged to represent and edit varied image properties as piece-wise-smooth data, and that such a representation is powerful, simple and intuitive.

1 Summary of contributions

The main contribution of this dissertation is the *vector primitive* of diffusion curve. With a single core element — a geometric curve with attributes attached on either side — vector diffusion curves can depict smooth color gradients, shading variations and complex texture deformations. This representation offers most of the benefits usually found in vector approaches, such as resolution independence, exact editability, and compactness. At the same time it allows to depict highly complex image content.

– In representing color gradients, vector diffusion curve images are comparable both in quality and coding efficiency with the state-of-the-art vector primitive (the *gradient mesh*), but are considerably simpler to create (according to several artists who have used both tools).

– For texture definition, this novel vector representation caters both for texture *creation*, and for texture *draping*. In the first case, intricate textures can be designed using a traditional sketching paradigm, that combines user-given vector drawings into regular and near-regular texture-maps. In the process of texture draping, the artist can deform the texture to suggest folds, ripples, or any other deformation that suits the artist's intent. The proposed model achieves compelling results without requiring the building of a 3D

model. As before, diffusion curves only represent image discontinuities, and this makes them a “lighter representation” to use in the creation process.

– Shading in vector graphics is generally represented through color variations. While diffusion curves can represent complex shading via color variations, a method to decouple shading from color is also proposed in the course of this thesis. This allows the artist to change the illumination in the drawing, without altering the underlying material color. Additionally, such a shading can be applied to mixed textured and non-textured elements to obtain a unified-looking illumination for the entire scene.

– Finally, the prototype system designed for creating and editing vector diffusion curves is real-time. This is due to a GPU-accelerated rendering proposed in this manuscript. As demonstrated by the resulting drawings, this allows artists to interactively design and manipulate color, shading and textures in 2D vector images.

A second contribution is a *vectorization* method that captures the complex color variations present in a raster image and transforms them into vector diffusion curves. The proposed approach relies on detecting discontinuities in a bitmap, and extracting color and blur along these discontinuities, to approach in vector graphics the realistic aspect of raster photographs.

And lastly, a contour-based *raster representation* is proved a powerful tool for creating enhanced representations of photographs. Image discontinuities, augmented with an importance value, are used to guide user manipulations and to preserve relevant image content.

2 Perspective

The use of contours as the basic image element has been shown to be a powerful tool for representing complex imagery, while still preserving a simplified and easy-to manipulate structure. Coupled with the descriptive power of vector graphics, contour-based representations can achieve an extraordinary level of control over the creation process. By using such an approach, a number of aspects of image creation can be explored.

2.1 Vector textures

Considering textures, the work described in this manuscript concentrates on how to allow users to drape a fabric-like texture map in a 2D image, via texture deformation and positioning. However, textures have a complicated nature, and it will be interesting to profit from the vector capacity of describing and parameterizing textures to further study varied texture behaviors.

For example, our current implementation supports a single texture per planar map region. While this proved useful and convenient for the regular and near-regular textures that are the focus of this manuscript, this approach does not extend easily to mixed texture approaches

[MZD05]. An interesting area of research would be to attach texture synthesis attributes to diffusion curves directly, diffuse these outwards and have textures grown dynamically.

Also, given the resolution independence of a vector-based representation it will be interesting to investigate level-of-detail considerations and hierarchical textures, akin to Han et al. [HRRG08].

Another very interesting question is the animation of texture, for example for mixed 2D and 3D cartoon animations. Offering an user the possibility of manipulating and animating textures raises interesting and novel questions; it can also help enrich the visual effects that are currently possible in cartoon animations. To exemplify the need for such research, the "Gankutsuou: The Count of Monte Cristo" animation (Figure 7.1(a)) was acclaimed for its rich, textured appearance, that set it apart from classical cartoon drawings. But unfortunately the flat and static aspect of the texture materials is very visible and visually disturbing, especially in movement. The research challenge here will be to create and animate texture effects like the ones in the static hand-drawing shown in Figure 7.1(b).

(a) image from the "Gankutsuou" animation

(b) hand drawing by Shigenori Soejima

Figure 7.1: *Examples of cartoon drawings with texture. (a) Flat texture applied to a cartoon animation. (b) Texture patterns that deform to suggest shape, in a hand-made static drawing.*

2.2 Vectorization of shading and texture

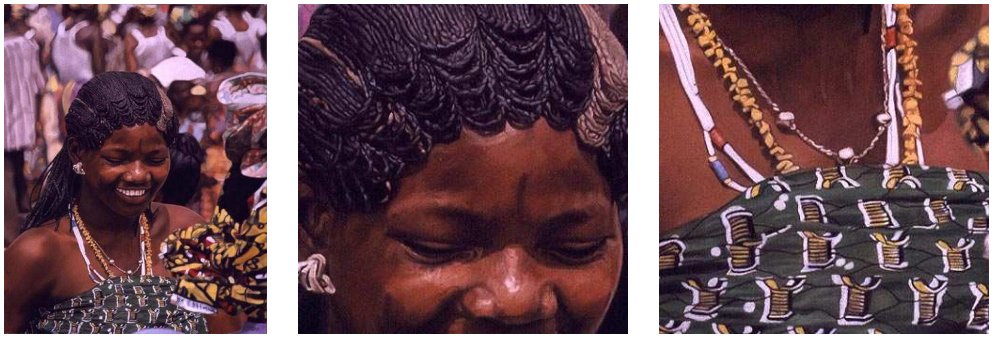
Vectorizing a bitmap image entails, in some sense, finding the semantic meaning comprised in the pixel grid. This is not easy, because a single point of color encodes all the characteristics of the captured scene at that point.

To the best of our knowledge, vectorization techniques (including our proposed method) only aim at capturing color and color gradients. No distinction is made between a surface material and a cast shadow, and textures are not treated differently from uniformly painted surfaces.

Texture vectorization is, indeed, a standing limitation for all vectorization techniques. The reason for this is that separating shading from reflectance and delimiting texture from object boundaries are still open challenges in image processing.

However, recent research has made advancements in these two domains. While methods that aim at fully estimating the *shading* in a bitmap image [Wei01, TFA05] remain impractical for vectorization purposes (Weiss et al. [Wei01] demands several images of the same scene, and Tappen et al. [TFA05] relies on classifiers and cannot disambiguate previously unencountered shading configurations), the related problem of shadow removal imposes more constraints on the input, and thus has better results. Shadow removal methods focus on cast shadows with clearly defined boundaries, and separate them from the lit image either automatically [FHD02, FDL04] or by user interaction [MTC07, WTBS07, SL08]. Such knowledge of shadowed regions can help the vectorization process by creating a separate layer for the shadow, and preserving the uniformly-painted object in a single region.

Textures present in natural images are warped by scene geometry and perspective projection. Several papers in the last years have employed user interactions to delimitate the textured regions and locally describe the geometry supporting the textures [LLH04, PSK06, ELS08]). For regular textures, feature matching algorithms can be used to automatically discover the deformations imposed on the texture by the underlying geometry [LT05, HLEL06]. Similar user interactions could be utilized to infuse semantic meaning into the vectorization process. Textures, for example, could be represented as a warped grid of vectorized texels, over a vectorized version of the supporting object.



(a) Ouidah Woman © Alicia St. Rose (b) detail (c) detail

Figure 7.2: *Example of hyperrealism art. Pastel on paper © Alicia St. Rose.*

On a more general level, expressing the real world through mathematical models has been extensively studied and used in 3D applications, and realistic "illusion worlds" can now be created. This knowledge of "how the real world can be designed" could be incorporated into 2D contour-based vector graphics, to allow the creation of ever-more complex 2D artworks, both in a realistic and expressive manner.

Another source of inspiration for creating a “convincing illusion” could also be the technique of the hyperrealism (Figure 7.2), where a very realistic look is used to transmit the artist’s personal view and message [Ros06]. As is the case with many other painting styles, the hyperrealist art usually starts with a line drawing[1].

[1] See `http://www.aliciastrose.com/` for a “making of” example of hyperrealism

Appendices

appendix A

Diffusion Curves Interface

This appendix describes the graphics interface used by artists to create and manipulate diffusion curves in the manner presented in Chapter 4. The screenshot in Figure A.1 shows the two principal windows used in our GUI:

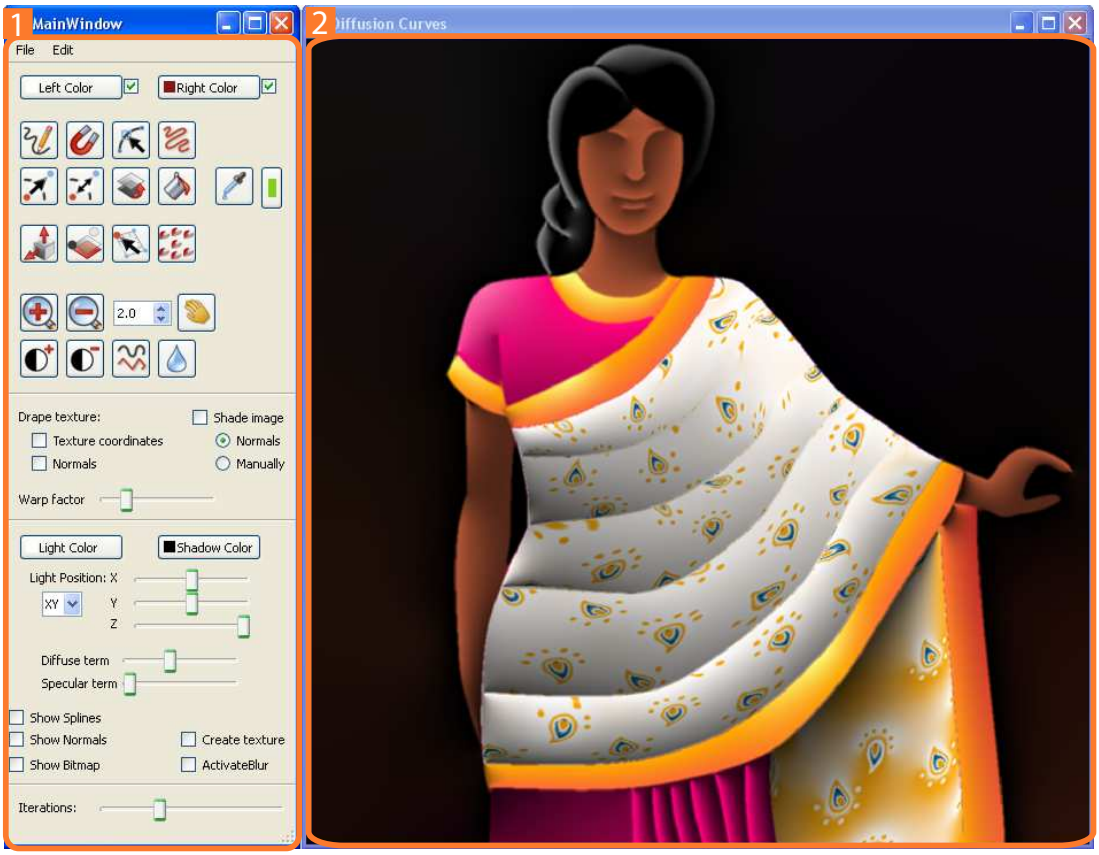


Figure A.1: *Diffusion curves main window.*

(1) *The Main Toolbox* provides access to all the proposed interaction tools. It contains the highest level menu, a set of icon buttons that can be used to select tools, and various toggles.

(2) *The Diffusion Curves* window is the canvas inside which diffusion curves are drawn and manipulated.

With the proposed GUI, an user can create diffusion curves images in two steps. (1) The starting point is drawing a curve inside the *The Diffusion Curves* window, as described in Section 1. (2) Each attribute of the created curve is afterwards independently edited to fit the user's intent. The tools allowing these manipulations are presented in Sections 2, 3 and 4. The sections dedicated to editing reproduce the organization in Chapter 4; this way, each manipulation from the "Creation and Manipulation" chapter has a counterpart in the current appendix, that describes the interface tools developed for the specified edition.

1 Drawing a diffusion curve

The tool used for draw a diffusion curve is the *Draw* tool from the Main Toolbox.

Draw tool: Draw a curve in free hand style.

The Draw tool lets the user trace a curve inside the Diffusion curves window, as if drawing with pencil on paper. This gives the shape of the curve, while the other attributes described in Chapter 3(left and right color, blur, normals and (u,v) texture coordinates) are automatically added to the curve extremities. Initial color and blur values can be given before drawing, as described in Table A.1.

In the **editing** step, the curve's shape and the other attached attributes can each be modified separately. To ensure independent manipulations, the user has at his disposal a different set of tools for each attribute. But all sets share a similar editing pipeline, and have a consistent behavior:

- First, the user selects a tool from the *Main Toolbox* to indicate what attribute is being modified.
- *Left Click* is used to change the attribute values of existing control points.
- To adjust the placement of control points, the *Right Click* is employed. On right clicking, a pop-up menu permits the addition and the deletion of control points. Other operations, specific to the selected attribute, are also shown in the pop-up menu.

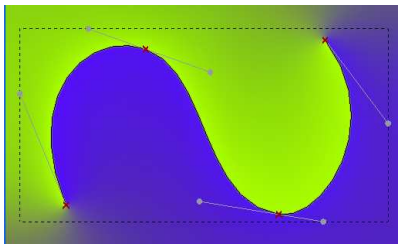
	 Example of a diffusion curve.
	A diffusion curve has a different *color* on each side. Prior to drawing, the pen colors can be chosen by clicking on the Left color and Right color buttons.
	*Blur* values can be defined for a curve. An initial blur can be chosen before drawing, with the spline blur slider.
	A curve has *normal* attributes on both sides. The default normal values point toward the viewer (the drawing is seen from the front).
	A diffusion curve has (u,v) texture coordinates defined on either side. The initial values are set to the geometric position of the curve, so the texture will lie flat inside a drawing region.

Table A.1: ***Diffusion curve attributes:*** *A description of all the attributes attached to the the diffusion curve's shape, and their default behavior. Some of the attributes can be set prior to drawing using the tools in the left column.*

2 Editing the shape and color

This section describes the tools developed in our GUI to enable the editing operations from Section 4.1.

2.1 Manual creation

The shape and color attributes are independently edited, and different tools are available for each of two attributes. A **shape**, given by a Bézier spline, can be modified by using the *Modify Shape* tool from the Main Toolbox.

Modify Shape tool: Select a curve in order to modify its shape.

To deform a diffusion curve, the following steps are needed:

(1) Select the *Modify Shape* tool.

(2) Select the curve. At this point, the Bézier controls are made visible, as shown in Table A.2 (a). The control points are drawn in red, while the corresponding tangents are colored in grey.

(3) Position the mouse point on a control point and select it with a *left click*. Holding the left button pressed while moving the mouse moves the control point inside the canvas (Table A.2 (b)).

The same select-and-drag action can be used to move the entire curve, if the mouse left click is inside the curve selection box (Table A.2 (c)), but not on the control points.

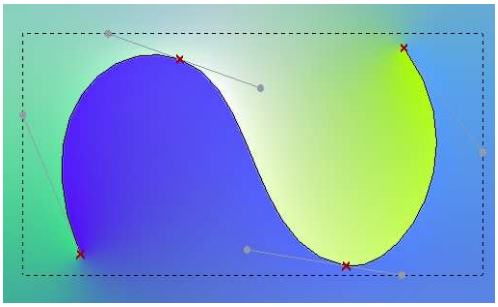
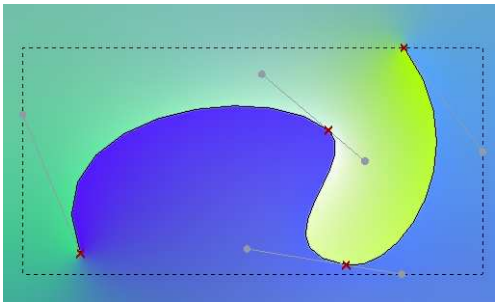
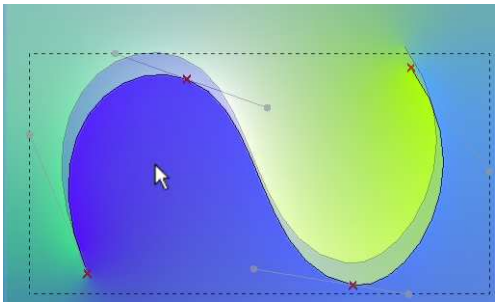
(a)	
(b) Click Left on *control points*: modify their position.	
(c) Click Left inside the curve *selection box*: move the curve.	

Table A.2: *The* Left Click *options for shape editing:* *The shape controls shown in (a) can be repositioned by dragging-and-dropping them (b). The entire curve can be moved in the same way (c).*

To modify the number of Bézier control points, the *Right Click* is used. For a curve selected with the *Modify Shape* tool, right clicking pops up a specific menu, illustrated in Table A.3 (a). This menu allows the user to add a control point at the mouse position (Table A.3 (b)), or to

delete the closest control point (Table A.3 (c)). A third possibility is to split the curve in two at the mouse position, as in Table A.3 (d).

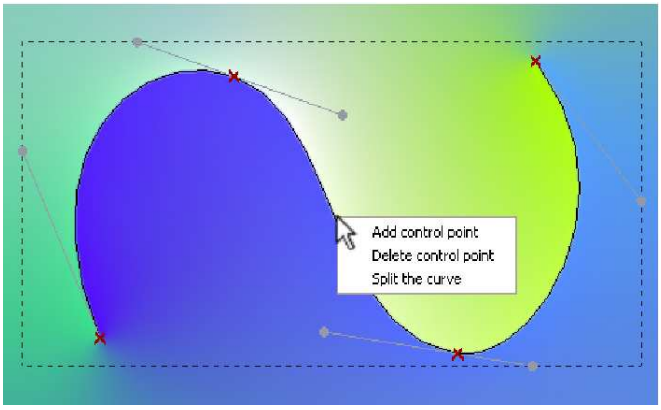
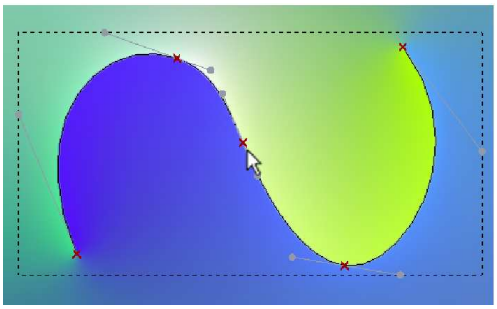
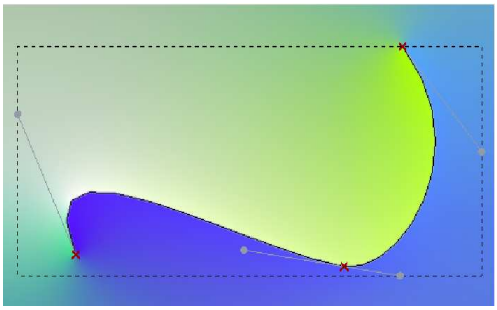
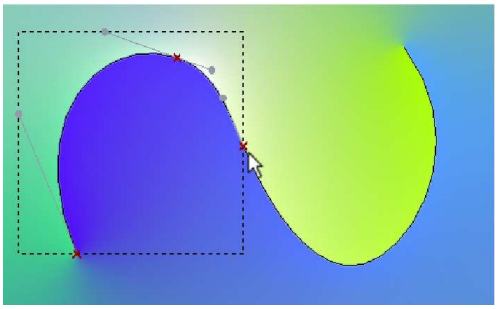
(a)	
(b) Add control point.	
(c) Delete control point.	
(d) Split the curve.	

Table A.3: ***The*** **Right Click** ***options****: (a) Screen capture of the menu choices when the 'Modify Shape' mode is active. (b)-(d) The result after applying each menu option on the initial curve.*

The diffusion curve's **colors and blur** are edited with the *Modify Color* tool from the Main Toolbox.

Modify Color tool: Select a curve and edit its colors and blur.

To change the value of color and blur control points, the user has to:

(1) Select the *Modify Color* tool.

(2) Select the curve. This displays the curve's color and blur control points. As illustrated in Table A.4 (a), the curve has two sets of color control points, one for each side; these controls are indicated by colored dots on the left and right of the curve. Blur control points are indicated by grey-level dots placed on the curve.

(3) Select a control point by *left clicking* on it. To change the color of a selected control, several options are possible, illustrated in Table A.4. When a blur control is selected, a slider dialog pops up and allows the user to modify the blur value.

By default, a drawn curve has color and blur controls only at the end-points. To manage the number of controls for a curve selected with the Modify Color tool, the user has to *right click* on the curve. This displays a menu with add and delete options for the left colors, the right colors, and for the blur (Figure A.2).

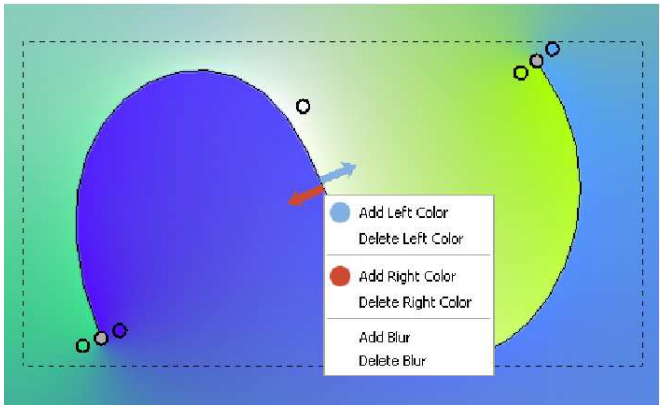


Figure A.2: Right Click *menu*. *Screen capture of the menu options for the 'Modify Color' tool. Color and blur controls can be added or deleted this way.*

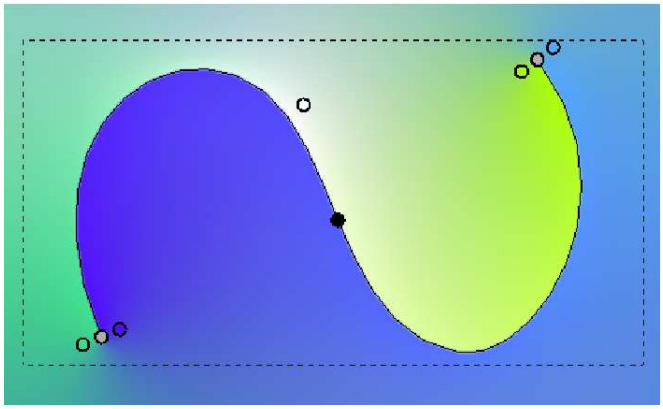
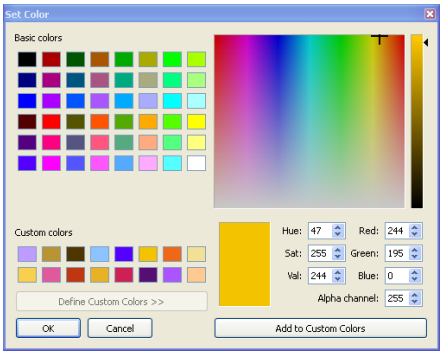
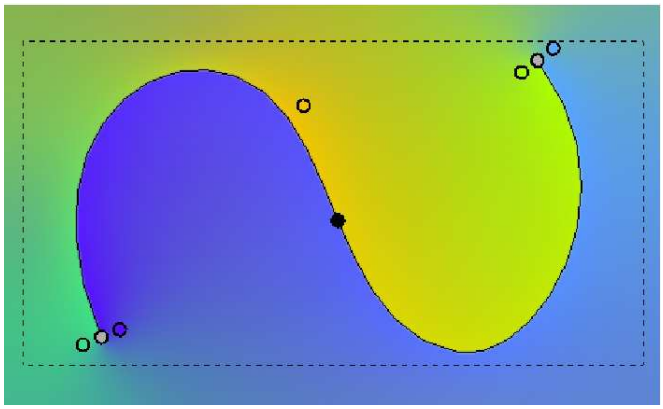
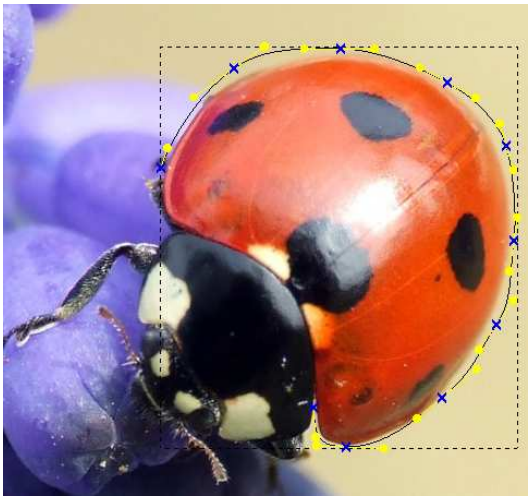
(a)	Select a curve with the *Modify Color* tool.	
(b)	*Click Left* on color controls to select them. Change colors using the color setting dialog: 	
(c)		Paste a color bucket color on color points touched by a left click.
(d)		Pick a color in the drawing and fill in the color bucket. Another way of defining the color in the color bucket is by using the color dialog.
(e)		Switch colors for the selected curve (Left ↔ Right).

Table A.4: ***Color changing options:*** *The colors of a selected curve are changed by using* Left Click *and a number of other tools. The 'Modify Color' tool is activated.*

2.2 Tracing an image

The "Manual creation" section focussed on the tools needed by users when creating an image completely from scratch. Another possibility is to rely on an existing image for guidance (Section 4.1.2). For this, our GUI proposes a new set of tools that lets the user utilize a bitmap image as a starting point, and manually *trace* over parts of the image. The colors are recovered automatically from the underlying image.

The steps necessary to trace over a bitmap image are explained below, in Table A.5. The table gives the steps in order, by illustrating the interface tools used (in the left column) and by showing the effect of each tool (the right column).

(a)	File→Load bitmap	Load a bitmap
(b)	☐ Show Bitmap	Toggle the bitmap display.
(c)		Draw a curve over a bitmap. 
(d)		Snap the curve to the bitmap contours. 

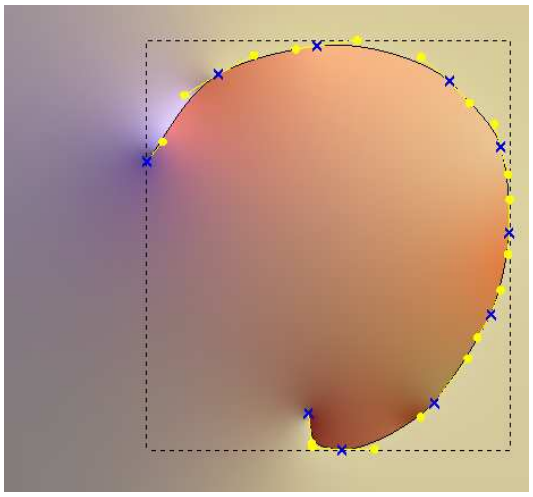
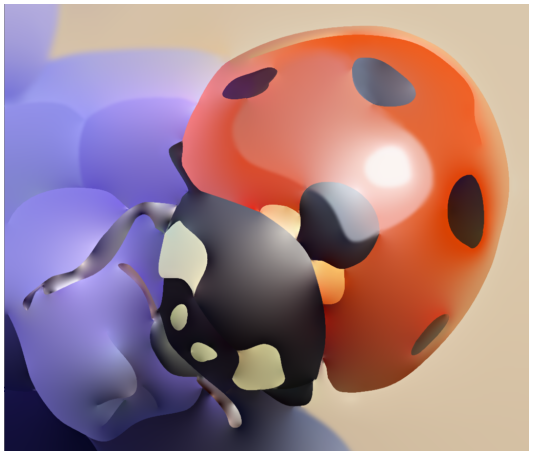
(e)		Read the colors in the bitmap and transfer them to the curve. 
(f)		Final result. 

Table A.5: ***Steps to trace over an image:*** *(a) First, load the support image by using the File menu from the Main Toolbox. (b) In the Diffusion curves window, show the bitmap image instead of the current diffusion curves drawing by checking the 'Show bitmap button'. (c) Draw a curve over an image feature. (d) Position the curve over the exact image discontinuity by repeatedly pressing the Magnet button. This automatically attracts the curve to the closest discontinuity. (e) When the curve is correctly placed over the image, sample the colors by pressing the 'Pick colors' button. (f) Repeating the process for all the interesting image features, a diffusion curves drawing with compelling coloring is obtained.*

2.3 Global manipulation

As diffusion curves are mark the drawing discontinuities, global deformations applied to their shape reflect in coherent stylization effects for the drawing (Section 4.1.3). Global modifications of diffusion curves color and blur attributes are also seamlessly integrated in the drawing.

Our GUI includes tools that allow the user to select multiple curves and to apply various editing operations on the entire selection. The multiple curve selection is done by using *Multi-selection* tool.

Multi-selection tool: Select multiple curves that will undergo the same transformations.

- *Shift* + *Left Click* adds the chosen curve to the multi-selection.
- *Ctrl* + *Left Click* lets the user scribble over the image. Every curve touched by the scribble is selected.

To add a curve to the multi-selection, two ways are proposed. One is to directly click on the curve; the other is to scribble over an area in the drawing, and all curves that are scribbled over are added to the selection. This second type of selection is illustrated in Figure A.3.

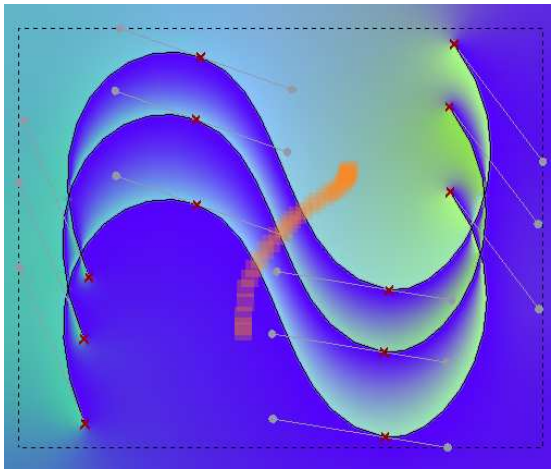


Figure A.3: ***Multi-selection using scribbles:*** *When the Multi-selection tool is active, holding* $Ctrl + LeftClick$ *lets the user scribble over the curves he wishes to select.*

Several global editions are available in our GUI for a multiple curve selection. Delete and Copy/Paste operations can be applied globally by using the *Edit* menu from the Main Toolbox, or by typing the corresponding keyboard shortcuts (see Figure A.4 for a screen capture).

Other global operators in our GUI modify either the *shape* of the selected curves – by smoothing or sharpening them – or the *color* – by globally changing the contrast. These effects are demonstrated on a single curve in Table A.6.

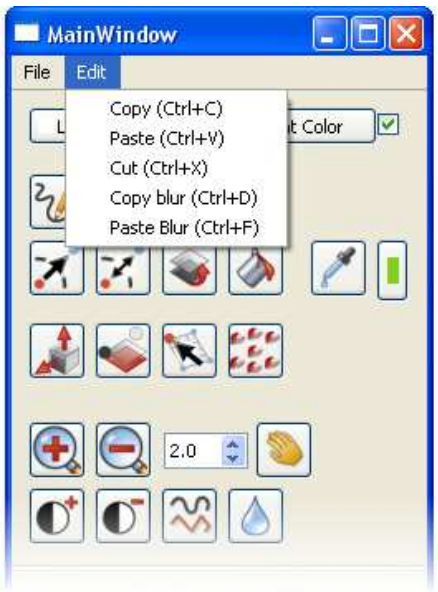
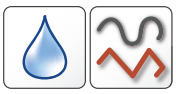
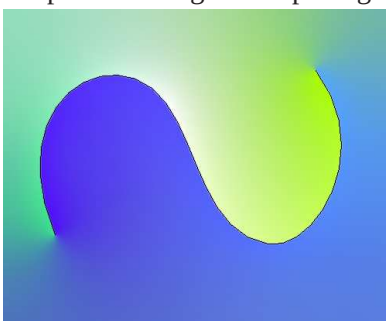


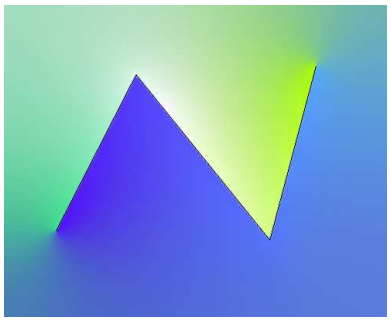
Figure A.4: ***The* Edit *menu:*** *Screen capture of the Main Toolbox, with the Edit menu displayed. This menu allows the copy, cut, paste and delete operations for one or multiple selected curves.*



Shape smoothing or sharpening:

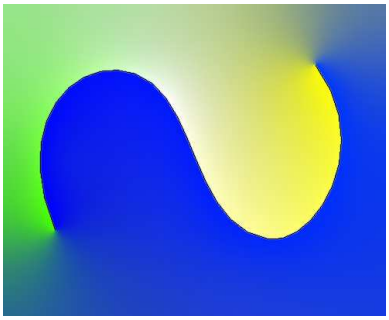


Initial drawing

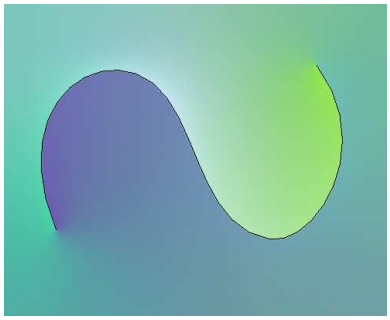


Sharpening

Contrast modification:



Increased contrast

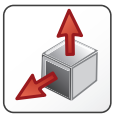


Lowered constrast

Table A.6: ***Global manipulations:*** *These operations can be applied to multiple curves at once.*

3 Manipulating the shading

To define and manipulate the shading, the diffusion curves system relies on *normals*, as is explained in Section 4.2. Normals are positioned the same way the colors are, on the left- and right-side of a curve, and they are edited using tools similar to the color tools described in Section 2.1. When the *Modify Normals* tool is activated, selecting a curve shows the attached normal values.



Modify Normals tool: Select a curve in order to manipulate its normal values.

Left Click on a normal control point selects the control point and lets the user define a new value via a normal widget (Figure A.5).

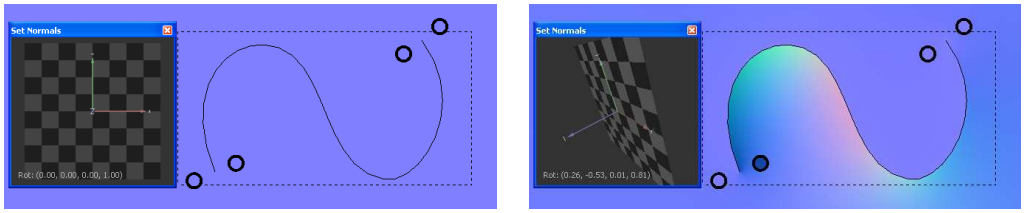


Figure A.5: ***The normal widget****: Screen captures of the normal values being defined along a diffusion curve.*

Adding and deleting normal control points can be done by *right clicking* on the selected curve (Figure A.6). New control points can thus be added at the mouse position, either to the left or to the right of the curve. When deleting points, the point closest to the mouse position on the chosen side (left or right) is marked for deletion.

The menu displayed when using the right click also allows an user to specify how normal values are defined and interpolated along the curve. When the 'Convex' or 'Concave' setting is selected, the normals are oriented along the instantaneous normal to the curve, in such a way that the surface implied by the normals is convex, respectively concave. In this set-up, only the slope orthogonal to the diffusion curve is given by the user. When the third option – the 'Free mode' – is selected, the user can specify the complete normal vector, and is not restricted to a fully convex or an entirely concave side.

Using the normal values, the shading of an image from a given *light position* is automatically computed. The light can be interactively positioned by the user through the use of various interface tools, shown in Figure A.7. The shading tools from the Main Toolbox allow an user to define:

- the light and shadow coloring;
- the light position;
- the desired material properties (the diffuse and specular term).

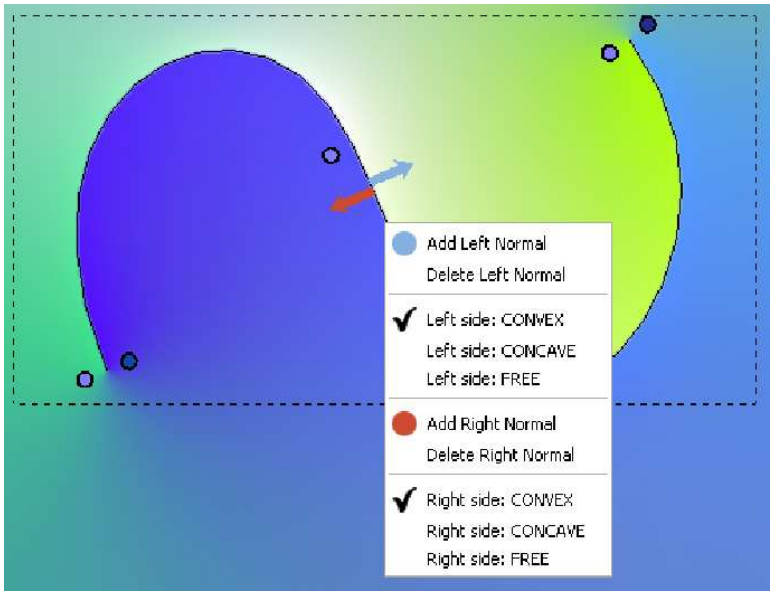
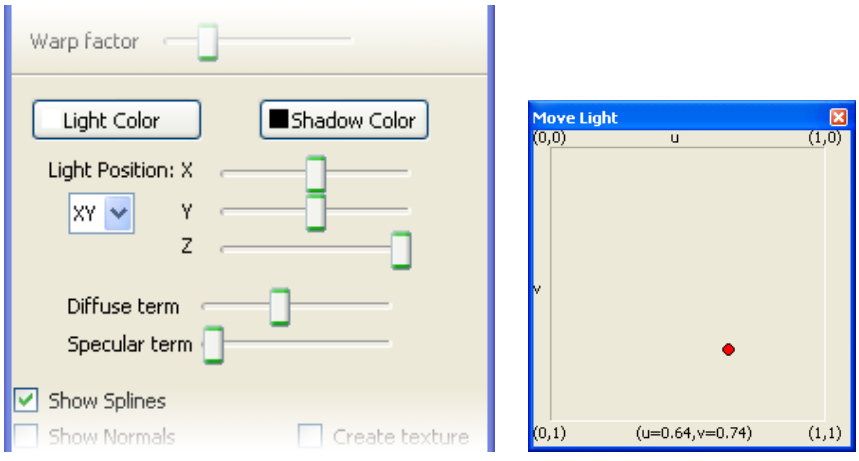


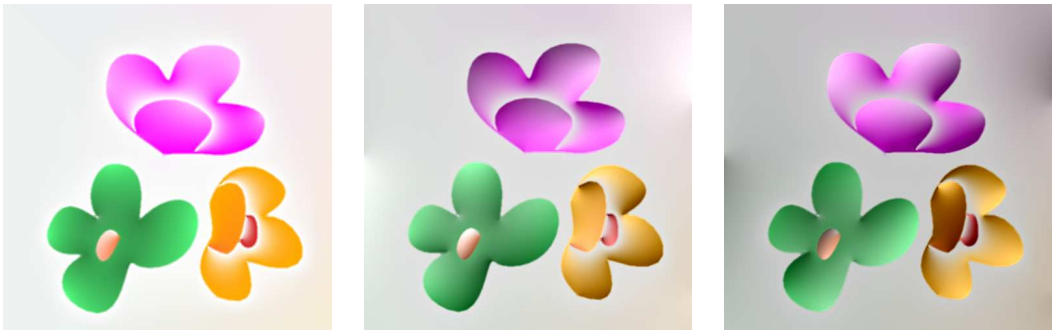
Figure A.6: ***The* Right Click *menu****: Screen capture of the menu choices when the 'Modify Normals' tool is active.*

The light (x,y) position can equally be changed by using the *Move Light* dialog. This allows the user to place the light at the desired positioned with a simple drag-and-drop action. The re-lighting results, exemplified in Figure A.8, are updated in real time.



(a) (b)

Figure A.7: ***The shading interface tools****: (a) The shading tools from the Main Toolbox. (b) The 'Move Light' dialog, that allows the user to change the light position by drag-and-dropping.*



(a) (b) (c)

Figure A.8: ***The re-lighting results****:(a) Unshaded drawing. (b) and (c) Shading results for two different light positions.*

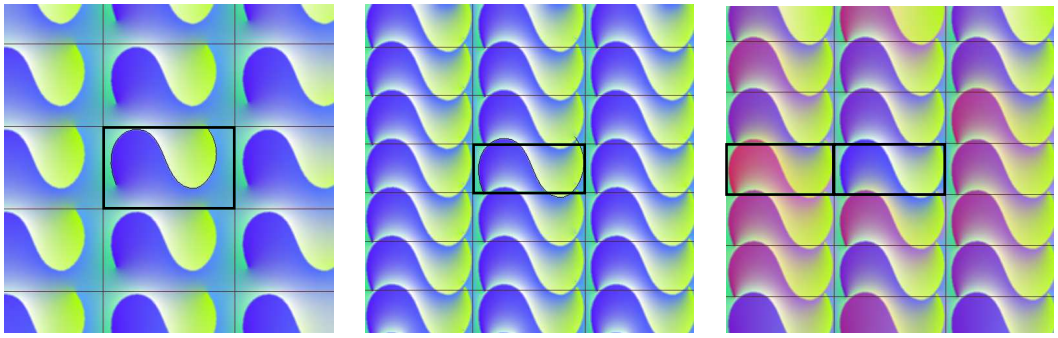
4 Adding textures

To create a textured design, three steps are proposed in Section 4. 3. (1) First, the user creates a texture with motifs of variable color and shape. (2) A line drawing is designed with diffusion curves, and is used as a support drawing for the textures. (3) The textures are positioned inside the support drawing and deformed to reflect the user's intention. Step (2) can be accomplished using the tools described in Sections 2 and 3. For the first and third steps, new interface tools are included in our GUI.

4.1 Creating the texture-map

The texture-map creation tools allow users to create *regular* and *near-regular* textures. To draw new textures, the *Create textures* mode has to be activated by clicking on the 'Create texture' button in the Main Toolbox menu. In texture mode, any drawing done by the user is replicated throughout a grid, as in Figure A.9 (a). The highlighted grid square is the user-defined texture element, whereas all the surrounding elements are automatically generated. The grid spacing can be modified by the user (Figure A.9 (b)) by dragging the grid lines horizontally or vertically; the texture is automatically recreated after each grid modification.

To create near-regular textures, the user has the option of defining multiple texture elements (or texels). The remaining, automatically generated texture instances interpolate between the user-defined patterns to create the texture. A new user defined element is created by *right clicking* in a grid square, and choosing 'Add new texel' from the displayed menu. The user can subsequently edit the colors, shapes and normal values of the newly created texture element, as illustrated in Figure A.9 (c). When the user adds or deletes a control point (for gemetry, color or normals), the corresponding controls in all user-defined texels are added or deleted. This is done to preserve the point-to-point correspondence needed for the automatic texel generation.



(a) (b) (c)

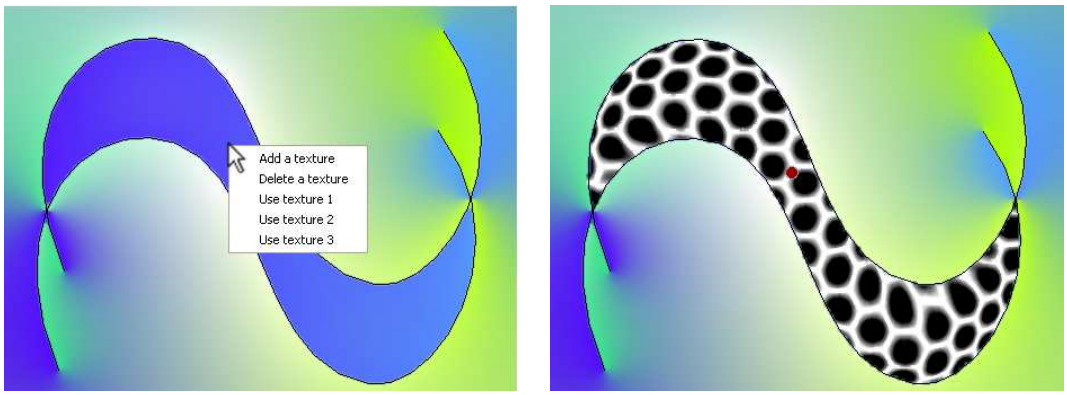
Figure A.9: ***The regular and near-regular textures****:(a) A user drawing (highlighted square) is replicated in a grid, to form a regular texture. (b) Grid spacing is modified and pattern is automatically regenerated. (c) Two user-defined texture elements (highlighted) are used to create a near-regular texture.*

4.2 Draping textures

Texture attachment The created texture maps or, optionally, an arbitrary bitmap, can be added to the support drawing. Managing the texture inclusion is done by selecting the *Include textures* mode.

Include textures tool: Manage the textures attached to the support drawing: add, delete, and reposition textures.

Once this mode is activated, the user can assign a texture to any region enclosed by the drawing's lines by *right clicking* inside the region. A menu is then displayed that, aside from adding a texture, allows the deletion or the reusing of already defined textures (Figure A.10 (a)).



(a) (b)

Figure A.10: ***The* Right Click *menu****: (a) Screen capture of the menu choices when the 'Include textures' mode is active. (b) The added texture, with its attachment point.*

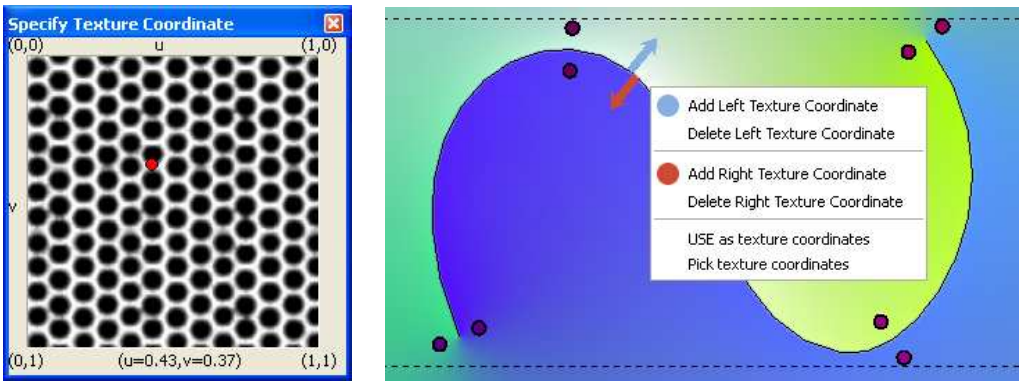
When attaching new textures to the drawing, the clicked point becomes the attachment point and defines the place where the texture center is found (Figure A.10 (b)). The texture can afterwards be re-positioned by dragging-and-dropping its attachment point to the new location.

Texture warping To complete the texture integration into the diffusion curves drawing, deformations can be applied to the texture maps, to suggest scenic depth, surface orientation or other artistic intentions. These distortions rely on two attributes – normals and (u,v) coordinates – attached to curves in the supporting drawing.

The interface tools for defining the normals have been detailed in Section 3. The (u,v) manipulation tools rely on similar interaction techniques. First, to access the (u,v) values, the *Modify* $(u,v)s$ mode has to be selected.

Modify $(u,v)s$ tool: View the (u,v) control points when a curve is selected and activate the tools used for manipulating the (u,v) attribute.

In this mode, the (u,v) control points are shown when a curve is selected. To modify the corresponding (u,v) value, a control point has to first be selected by a *left click*; a dialog window (Figure A.11 (a)) allows to user to specify a new value.



(a) (b)

Figure A.11: ***The*** (u,v) ***manipulation tools****: (a) The dialog window used to define new* (u,v) *values. (b) Screen capture of The* Right Click *menu.*

Control points can be added or deleted by *right clicking* on the curve. As for the other diffusion curves attributes, this displays a menu with several options. In the case of (u,v)s, the proposed actions, shown in Figure A.11 (b), allow the user to manage the control point on either side of the curve, but also let him choose whether the curve in its entirety diffuses (u,v) coordinates or not (the 'USE as texture coordinates' option). Another option specific to the (u,v) attribute is 'Pick texture coordinates', which automatically generates control points that create the least possible distortion in the texture.

5 Assistance tools

Up until now, we have focussed on describing the tools and menus that allow an user to define diffusion curves. The proposed GUI also contains other interface tools that do not directly enable users to manage diffusion curves, but are there to aid in the creative process.

Image navigation tools, for example, allow the user to easily navigate through the artwork, to set zoom levels and to move the visible parts of the image (Table A.7 (a)). Additional interface tools *toggle* different image views, such as the image of normals or planar map (Table A.7 (b)). And finally, drawings and textures can be saved and opened with *File* menu (Table A.7 (c)).

(a)	 Zoom forward	 Zoom backward	 Pan
(b)	 Toggle splines display.	 Toggle blur computation.	 Toggle normal map display.
(c)			

Table A.7: ***Assistance tools****: (a) Image navigation tools. (b) Various toggles. (c) Screen capture of the 'File' menu options*

appendix B

Texture Structural Definition

The term “visual textures” describes the perceived appearance of materials and support surfaces in the real or depicted world. The vastness of the visual texture types and their often contradictory properties make a unified description of textures difficult. The usual practice is then to define textures from a certain perspective of their nature. Categories of texture definition include statistical models, models based on spatial frequency filtering, and structural approaches.

In the context of vector graphics, texture representation models tend to be structural techniques. This is because vector graphics systems are strongly user-oriented, and the structural definition comes directly from the human interpretation of perceived patterns. This *structural approach*, originally proposed by Haralick in 1973 ([HSD73]), considers a texture as an “organized area phenomenon” which can be decomposed into “primitives” having specific spatial distributions. For instance, each texture in Figure B.1 is composed of particular texture elements, e.g. objects (bricks), shapes (jigsaw pieces), or simply color patterns. These primitives are organized in a particular spatial structure indicating certain underlying placement rules.

Yanxi Liu et al. [LLH04, LT05] and Wen-Chieh Lin et al. [LHW+04] have recently proposed a structural characterization that classify textures as:

1. *Regular textures*. This refers to periodic patterns where the color and shape of all texture primitives are repeating in equal intervals.

2. *Near-regular textures*. These textures, while having recognizable primitives and structure periodicity, depart slightly from regular tiling along different axes of appearance, and thus could have:

    - A regular structural layout but irregular color appearance in individual tiles (like the brick texture in Figure B.1).
    - A distorted spatial layout but topologically regular alterations in color (the bottom right image in the near-regular textures set in Figure B.1).

- Small deviations from regularity in both structural placement and color (the weave examples in Figure B.1).

3. *Irregular textures*. Here, the texture elements have individually discernable shapes, but they vary in appearance. The distribution, while irregular, follows coherent rules.

4. *Near-stochastic textures*. In near-stochastic textures, individual elements are less distinguishable. The general aspect is that of patches of color randomly distributed.

5. *Stochastic textures*. These are noise textures.

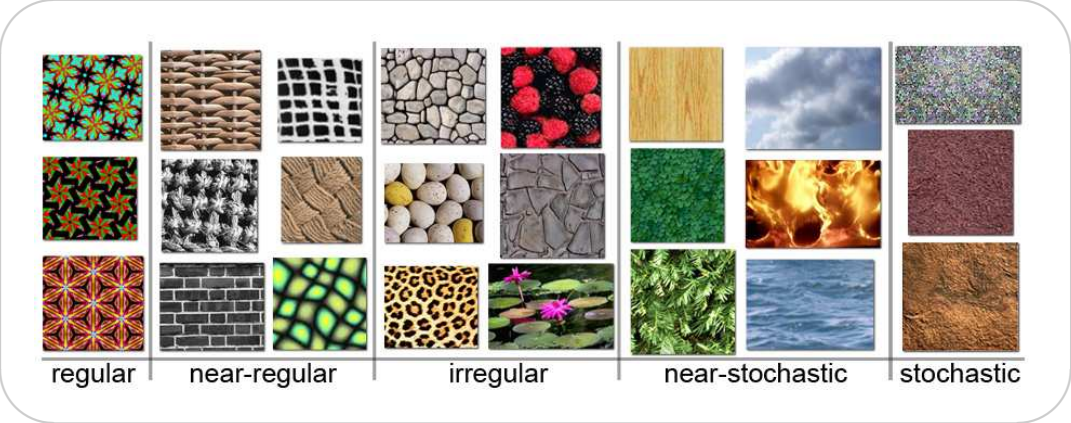


Figure B.1: ***Texture structural organization****: Texture types, considering the texture elements appearance and their planar distribution. Image taken from the "Near-regular texture analysis and manipulation" paper [LLH04]*

Bibliography

[ADA+04] Aseem Agarwala, Mira Dontcheva, Maneesh Agrawala, Steven Drucker, Alex Colburn, Brian Curless, David Salesin, and Michael Cohen. Interactive digital photomontage. *ACM TOG (Proceedings of SIGGRAPH 2004)*, pages 294–302, 2004.

[ASP07] Paul Asente, Mike Schuster, and Teri Pettit. Dynamic planar map illustration. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, 26(3), 2007.

[AWB06] Ken-ichi Anjyo, Shuhei Wemler, and William Baxter. Tweakable light and shade for cartoon animation. In *NPAR '06: Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, pages 133–139, New York, NY, USA, 2006. ACM.

[BA06] W. Baxter and K. Anjyo. Latent Doodle Space. In *Computer Graphics Forum*, volume 25, pages 477–485. Blackwell Synergy, 2006.

[BBT+06] Pascal Barla, Simon Breslav, Joëlle Thollot, François Sillion, and Lee Markosian. Stroke pattern analysis and synthesis. *Computer Graphics Forum (Proc. of Eurographics 2006)*, 25, 2006.

[Ber87] Fredrik Bergholm. Edge focusing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(6):726–741, 1987.

[BFSC04] Marcelo Bertalmio, Pere Fort, and Daniel Sanchez-Crespo. Real-time, accurate depth of field using anisotropic diffusion and programmable graphics cards. In *Proc. of 3DPVT*, pages 767–773, 2004.

[BG89] P. Baudelaire and M. Gangnet. Planar maps: an interaction paradigm for graphic design. *ACM SIGCHI Bulletin*, 20(SI):313–318, 1989.

[BGH03] J.A. Bangham, S.E. Gibson, and R.W. Harvey. The art of scale-space. In *British Machine Vision Conference*, pages 569–578, 2003.

[BHM00] William L. Briggs, Van Emden Henson, and Steve F. McCormick. *A multigrid tutorial (2nd ed.)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[BK04] Mario Botsch and Leif Kobbelt. An intuitive framework for real-time freeform modeling. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 630–634, 2004.

[BL91] James R. Bergen and Michael S. Landy. Computational modeling of visual texture segregation. *Computational Models of Visual Processing*, pages 253–271, 1991.

[BSCB00] Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, and Coloma Ballester. Image inpainting. In *Proc. of ACM SIGGRAPH 2000*, pages 417–424, 2000.

[BTM06] Pascal Barla, Joëlle Thollot, and Lee Markosian. X-toon: An extended toon shader. In *INPAR '06: Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*. ACM, 2006.

[Can86] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

[Car88] Stefan Carlsson. Sketch based coding of grey level images. *Signal Processing*, 15(1):57–83, 1988.

[CH03] John P. Collomosse and Peter M. Hall. Cubist style rendering from photographs. *IEEE Transactions on Visualization and Computer Graphics*, 9(4):443–453, 2003.

[CH05] John P. Collomosse and Peter M. Hall. Genetic paint: A search for salient paintings. In *Proc. of EvoMUSART*, pages 437–447, 2005.

[Che87] L. P. Chew. Constrained delaunay triangulations. In *SCG '87: Proceedings of the third annual symposium on Computational geometry*, pages 215–222, 1987.

[CR68] F. W. Campbell and J. G. Robson. Application of fourier analysis to the visibility of gratings. *The Journal of Physiology*, 197:551–566, 1968.

[DMSB99] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 317–324, 1999.

[DP73] David Douglas and Thomas Peucker. Algorithms for the reduction of the number of points required for represent a digitzed line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[DS02] Doug DeCarlo and Anthony Santella. Stylization and abstraction of photographs. *ACM TOG (Proceedings of SIGGRAPH 2002)*, 21(3):769–776, 2002.

[EB09] Inc. Encyclopædia Britannica. *Encyclopædia Britannica online – http://www.britannica.com/*. Encyclopædia Britannica, Inc., 2009.

[EG01a] J. H. Elder and R. M. Goldberg. Image editing in the contour domain. *IEEE PAMI*, 23(3):291–296, 2001.

[EG01b] James H. Elder and Richard M. Goldberg. Image editing in the contour domain. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(3):291–296, 2001.

[Eld99] James H. Elder. Are edges incomplete? *International Journal of Computer Vision*, 34(2-3):97–122, 1999.

[ELS08] Christian Eisenacher, Sylvain Lefebvre, and Marc Stamminger. Texture synthesis from photographs. In *Proceedings of the Eurographics conference*, 2008.

[FDL04] Graham D. Finlayson, Mark S. Drew, and Cheng Lu. Intrinsic images by entropy minimization. In *ECCV '04: Proceedings of the 7th European Conference on Computer Vision*, pages 582–595, 2004.

[FF87] M. A. Fischler and O. Firschein. *Intelligence: The Eye, the Brain and the Computer*. Addison-Wesley, 1987.

[FHD02] Graham D. Finlayson, Steven D. Hordley, and Mark S. Drew. Removing shadows from images. In *ECCV '02: Proceedings of the 7th European Conference on Computer Vision*, pages 823–836, 2002.

[FLW02] Raanan Fattal, Dani Lischinski, and Michael Werman. Gradient domain high dynamic range compression. *ACM TOG (Proceedings of SIGGRAPH 2002)*, pages 249–256, 2002.

[GDHZ06] Yotam I. Gingold, Philip L. Davidson, Jefferson Y. Han, and Denis Zorin. A direct texture placement and editing interface. In *UIST '06: Proceedings of the 19th annual ACM symposium on User interface software and technology*, pages 23–32, 2006.

[Gos94] Ardeshir Goshtasby. On edge focusing. *Image and Vision Computing*, 12(4):247–256, 1994.

[GWL+03] Nolan Goodnight, Cliff Woolley, Gregory Lewin, David Luebke, and Greg Humphreys. A multigrid solver for boundary value problems using programmable graphics hardware. In *Graphics Hardware*, pages 102–111, 2003.

[GZW03] Cheng-En Guo, Song-Chun Zhu, and Ying Nian Wu. Towards a mathematical theory of primal sketch and sketchability. In *ICCV 2003: Proceedings of the Ninth IEEE International Conference on Computer Vision*, volume 2, pages 1228–1235, 2003.

[GZW07] Cheng-En Guo, Song-Chun Zhu, and Ying Nian Wu. Primal sketch: Integrating structure and texture. *Computer Vision and Image Understanding*, 106(1):5–19, 2007.

[Hec86] Paul S Heckbert. Survey of texture mapping. *IEEE Comput. Graph. Appl.*, 6(11):56–67, 1986.

[Her98] Aaron Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In *SIGGRAPH '98*, pages 453–460, 1998.

[HLEL06] James H. Hays, Marius Leordeanu, Alexei A. Efros, and Yanxi Liu. Discovering texture regularity as a higher-order correspondence problem. In *9th European Conference on Computer Vision*, May 2006.

[Hod03a] Brenda Hoddinott. *Drawing for Dummies*. Wiley Publishing, 2003.

[Hod03b] Elaine R. S. Hodges. *Guild Handbook of Scientific Illustration*. John Wiley & Sons, Inc, 2003.

[HRRG08] Charles Han, Eric Risser, Ravi Ramamoorthi, and Eitan Grinspun. Multiscale texture synthesis. *ACM Trans. Graph.*, 27(3):1–8, 2008.

[HSD73] Robert M. Haralick, K. Shanmugam, and Its'Hak Dinstein. Textural features for image classification. *IEEE Transactions on Systems, Man and Cybernetics*, 3(6):610–621, 1973.

[IMIM08] Takashi Ijiri, Radomir Mêch, Takeo Igarashi, and Gavin Miller. An example-based procedural system for element arrangement. *Computer Graphics Forum (Proc. of Eurographics 2008)*, 27, 2008.

[Ink08] Inkscape. *Inkscape documentation: http://www.inkscape.org/doc/*, 2008.

[JC08] Pushkar Joshi and Nathan Carr. Repoussé: Automatic inflation of 2D artwork. In *SBIM'08*, pages 49–55, 2008.

[Joh02] Scott F. Johnston. Lumo: illumination for cel animation. In *NPAR '02: Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, pages 45–52, New York, NY, USA, 2002. ACM.

[Jul62] Béla Julesz. Visual pattern discrimination. *IRE Transactions on Information Theory*, 8(2):84–92, 1962.

[Kap05] Craig S. Kaplan. Islamic star patterns from polygons in contact. In *GI '05: Proceedings of Graphics Interface 2005*, pages 177–185, 2005.

[KCC06] Hyung W. Kang, Charles K. Chui, and Uday K. Chakraborty. A unified scheme for adaptive stroke-based rendering. *The Visual Computer (Proceedings of Pacific Graphics 2006)*, 22(9):814–824, 2006.

[KD79] J. J. Koenderink and A. J. Doorn. The internal representation of solid shape with respect to vision. *Biological Cybernetics*, 32(4):211–216, 1979.

[Koe84] Jan J. Koenderink. The structure of images. *Biological Cybernetics*, 50(5):363–370, 1984.

[Kov06] Tania Kovats. *The Drawing Book: A survey of drawing - The primary means of expression*. Black Dog Publishing, 2006.

[KS04a] Craig S. Kaplan and David H. Salesin. Dihedral escherization. In *GI '04: Proceedings of Graphics Interface 2004*, pages 255–262, 2004.

[KS04b] Craig S. Kaplan and David H. Salesin. Islamic star patterns in absolute geometry. *ACM Transactions on Graphics*, 23(2):97–119, 2004.

[KWT87] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1987.

[Lév01] Bruno Lévy. Constrained texture mapping for polygonal meshes. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 417–424, 2001.

[Lév08] Bruno Lévy. *Numerical geometry*. PhD thesis, INPL, France, February 2008. HdR Habilitation thesis.

[LG04] Michael S. Landy and Norma Graham. Visual perception of texture. In *In L. M*, pages 1106–1118. MIT Press, 2004.

[LHM09] Yu-Kun Lai, Shi-Min Hu, and Ralph R. Martin. Automatic and topology-preserving gradient mesh generation for image vectorization. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2009)*, 28(3), 2009.

[LHW+04] Wen-Chieh Lin, James Hays, Chenyu Wu, Vivek Kwatra, and Yanxi Liu. A comparison study of four texture synthesis algorithms on near-regular textures. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Posters*, New York, NY, USA, 2004. ACM.

[Lin91] Tony Lindeberg. *Discrete Scale-Space Theory and the Scale-Space Primal Sketch*. PhD thesis, 1991. PhD thesis.

[Lin93] Tony Lindeberg. Effective scale: A natural unit for measuring scale-space lifetime. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(10):1068–1074, 1993.

[Lin98] Tony Lindeberg. Edge detection and ridge detection with automatic scale selection. *International Journal of Computer Vision*, 30(2):117–154, 1998.

[LL06] Gregory Lecot and Bruno Lévy. Ardeco: Automatic Region DEtection and COnversion. In *Eurographics Symposium on Rendering*, 2006.

[LLH04] Yanxi Liu, Wen-Chieh Lin, and James Hays. Near-regular texture analysis and manipulation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2004)*, 23:368–376, 2004.

[LM99] Thomas Leung and Jitendra Malik. Recognizing surfaces using three-dimensional textons. *Computer Vision (Proceedings of ICCV 1999)*, 1999.

[LP00] Laurent Lefebvre and Pierre Poulin. Analysis and synthesis of structural textures. In *Graphics Interface 2000*, pages 77–86, May 2000.

[LT05] Yanxi Liu and Yanghai Tsin. The promise and perils of near-regular texture. *International Journal of Computer Vision*, 62:1–2, 2005.

[Mar82] David Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. W. H. Freeman & Company, 1982.

[MH80] D. Marr and E. C. Hildreth. Theory of edge detection. *Proceedings of the Royal Society of London. Series B, Biological Sciences*, 207:187–217, 1980.

[MP08] James McCann and Nancy S. Pollard. Real-time gradient-domain painting. *ACM TOG (Proc. of SIGGRAPH)*, 27(3), 2008.

[MTC07] Ankit Mohan, Jack Tumblin, and Prasun Choudhury. Editing soft shadows in a digital photograph. *IEEE Computer Graphics and Applications*, 27(2):23–31, 2007.

[MZD05] Wojciech Matusik, Matthias Zwicker, and Frédo Durand. Texture design using a simplicial complex of morphable textures. *ACM Trans. Graph.*, 24(3):787–794, 2005.

[OBBT07] Alexandrina Orzan, Adrien Bousseau, Pascal Barla, and Joëlle Thollot. Structure-preserving manipulation of photographs. In *NPAR*, pages 103–110, 2007.

[OBW+08] Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. Diffusion curves: A vector representation for smooth-shaded images. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2008)*, volume 27, 2008.

[ON95] Michael Oren and Shree K. Nayar. Generalization of the lambertian model and implications for machine vision. *International Journal of Computer Vision*, 14(3):227–251, 1995.

[Pal99] S. Palmer. *Vision Science : Photons to Phenomenology.* MIT Press, 1999.

[PB06] Brian Price and William Barrett. Object-based vectorization for interactive image editing. In *Visual Computer (Proceedings of Pacific Graphics '06)*, volume 22, pages 661–670, September 2006.

[PCPN07] Giuseppe Papari, Patrizio Campisi, Nicolai Petkov, and Alessandro Neri. A biologically motivated multiresolution approach to contour detection. *EURASIP Journal on Advances in Signal Processing*, 2007, 2007.

[PFWF00] Lena Petrović, Brian Fujito, Lance Williams, and Adam Finkelstein. Shadows for cel animation. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 511–516, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[PGB03] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. *ACM TOG (Proceedings of SIGGRAPH 2003)*, 2003.

[Pho75] Bui Tuong Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, 1975.

[PJP93] Ulrich Pinkall, Strasse Des Juni, and Konrad Polthier. Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics*, 2:15–36, 1993.

[PSK06] Darko Pavić, Volker Schönefeld, and Leif Kobbelt. Interactive image completion with perspective correction. *The Visual Computer: International Journal of Computer Graphics*, 22(9):671–681, 2006.

[PV95] Ken Perlin and Luiz Velho. Live paint: Painting with procedural multiscale textures. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive technique*, pages 153–160. ACM Press, 1995.

[RIY04] Ramesh Raskar, Adrian Ilie, and Jingyi Yu. Image fusion for context enhancement and video surrealism. In *Proceedings of NPAR 2004*, pages 85–152, 2004.

[RL93] A. Ravishankar Rao and Gerald L. Lohse. Identifying high level features of texture perception. *CVGIP: Graph. Models Image Process.*, 55(3):218–233, 1993.

[Rom03] Bart ter Haar Romeny. *Front-End Vision and Multi-Scale Image Analysis*. Kluwer Academic Publishers, 2003.

[Ros06] Robert Rosenblum. *Ron Mueck*. Thames & Hudson, 2006.

[SD04] Anthony Santella and Doug DeCarlo. Visual interest and npr: an evaluation and manifesto. In *Proceedings of NPAR 2004*, pages 71–150, 2004.

[Sel03] Peter Selinger. *Potrace: a polygon-based tracing algorithm*, 2003.

[SKvW+92] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. Fast shadows and lighting effects using texture mapping. *Computer Graphics (Proceedings of SIGGRAPH '92)*, 26(2):249–252, 1992.

[SL08] Yael Shor and Dani Lischinski. The shadow meets the mask: Pyramid-based shadow removal. *Computer Graphics Forum (Proceedings of Eurographics 2008)*, 27(2), 2008.

[SLWS07] Jian Sun, Lin Liang, Fang Wen, and Heung-Yeung Shum. Image vectorization using optimized gradient meshes. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, 26(3):11, 2007.

[Spa98] Anne Morgan Spalter. *The Computer in the Visual Arts*. Addison-Wesley Professional, 1998.

[Ste92] Saul Steinberg. *The Discovery of America*. Alfred A. Knopf, New York, 1992.

[Sut80] Ivan Edward Sutherland. *Sketchpad: A man-machine graphical communication system (Outstanding dissertations in the computer sciences)*. Garland Publishing, Inc., New York, NY, USA, 1980.

[TABI07] Hideki Todo, Ken-ichi Anjyo, William Baxter, and Takeo Igarashi. Locally controllable stylized shading. volume 26, page 17, 2007.

[TFA05] Marshall F. Tappen, William T. Freeman, and Edward H. Adelson. Recovering intrinsic images from a single image. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(9), 2005.

[Wan95] Brian A. Wandell. *Foundations of Vision*. Sinauer Associates, 1995.

[Wei01] Y. Weiss. Deriving intrinsic images from image sequences. In *Proceedings of ICCV 2001*, pages 68–75, 2001.

[Wel04] Terry Welsh. Parallax mapping with offset limiting: A per-pixel approximation of uneven surfaces. 2004.

[WLL+06] Fang Wen, Qing Luan, Lin Liang, Ying-Qing Xu, and Heung-Yeung Shum. Color sketch generation. In *Proceedings of NPAR 2006*, 2006.

[WOBT09] Holger Winnemöller, Alexandrina Orzan, Laurence Boissieux, and Joëlle Thollot. Texture design and draping in 2d images. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering 2009)*, 28(4):1091–1099, 2009.

[WOG06] Holger Winnemöller, Sven C. Olsen, and Bruce Gooch. Real-time video abstraction. *ACM TOG (Proceedings of SIGGRAPH 2006)*, 25(3):1221–1226, 2006.

[WSTS08] Tai-Pang Wu, Jian Sun, Chi-Keung Tang, and Heung-Yeung Shum. Interactive normal reconstruction from a single image. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2008)*, 27(5):1–9, 2008.

[WTBS07] Tai-Pang Wu, Chi-Keung Tang, Michael S. Brown, and Heung-Yeung Shum. Natural shadow matting. *ACM Transactions on Graphics*, 26(2), 2007.

[WXSC04] Jue Wang, Yingqing Xu, Heung-Yeung Shum, and Michael F. Cohen. Video tooning. *ACM TOG (Proceedings of SIGGRAPH 2004)*, 23(3):574–583, 2004.

[WZS98] Michael T. Wong, Douglas E. Zongker, and David H. Salesin. Computer-generated floral ornament. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 423–434, 1998.

[Zen86] Silvano Di Zenzo. A note on the gradient of a multi-image. *Computer Vision, Graphics, and Image Processing*, 33(1):116–125, 1986.