Technical Report

UCAM-CL-TR-862
ISSN 1476-2986

Number 862

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Surface modelling for 2D imagery

Henrik Lieng

October 2014

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

This technical report is based on a dissertation submitted June 2014 by the author for the degree of Doctor of Philosophy to the University of Cambridge, Fitzwilliam College.

Some figures in this document are best viewed in colour. If you received a black-and-white copy, please consult the online version if necessary.

Technical reports published by the University of Cambridge Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/techreports/

ISSN 1476-2986

Summary

Surface modelling for 2D imagery, Henrik Lieng

Vector graphics provides powerful tools for drawing scalable 2D imagery. With the rise of mobile computers, of different types of displays and image resolutions, vector graphics is receiving an increasing amount of attention. However, vector graphics is *not* the leading framework for creating and manipulating 2D imagery. The reason for this reluctance of employing vector graphical frameworks is that it is difficult to handle complex behaviour of colour across the 2D domain.

A challenging problem within vector graphics is to define smooth colour functions across the image. In previous work, two approaches exist. The first approach, known as diffusion curves, diffuses colours from a set of input curves and points. The second approach, known as gradient meshes, defines smooth colour functions from control meshes. These two approaches are incompatible: diffusion curves do not support the local behaviour provided by gradient meshes and gradient meshes do not support freeform curves as input. My research aims to narrow the gap between diffusion curves and gradient meshes.

With this aim in mind, I propose solutions to create control meshes from freeform curves. I demonstrate that these control meshes can be used to render a vector primitive similar to diffusion curves using subdivision surfaces. With the use of subdivision surfaces, instead of a diffusion process, colour gradients can be locally controlled using colour gradient curves associated with the input curves.

The advantage of local control is further explored in the setting of vector-centric image processing. I demonstrate that a certain contrast enhancement profile, known as the Cornsweet profile, can be modelled via surfaces in images. This approach does not produce saturation artefacts related with previous filter-based methods. Additionally, I demonstrate various approaches to artistic filtering, where the artist locally models given artistic effects.

Gradient meshes are restricted to rectangular topology of the control meshes. I argue that this restriction hinders the applicability of the approach and its potential to be used with control meshes extracted from freeform curves. To this end, I propose a mesh-based vector primitive that supports arbitrary manifold topology of the control mesh.

Acknowledgements

I thank:

- my principal supervisor Neil A. Dodgson for suggestions, intellectual contributions, proofreading, contributions to a couple of figures (Figures 4.11 and 4.12), generally ensuring that I have progressed towards a PhD dissertation, and for letting me use figures (Figures 3.2 and 6.3) from his lecture notes for the Advanced Graphics course;
- my second supervisor Jiří Kosinka for suggestions, intellectual contributions, proofreading, contributions to multiple figures (Figures 4.14, 6.4, 6.8, 6.11, 6.12, 6.16, and 7.11), and for letting me use his subdivision code;
- my hosts at the Max Planck Institute for Informatics (MPI), Erik Reinhard and Tania Pouli, for suggestions, intellectual contributions, proofreading, and other contributions (see the previous page);
- my other collaborators, including Flora P. Tasse, Jingjing Shen, Christian Richardt, James Tompkin, and Jan Kautz for suggestions and contributions;
- Chuong Nguyen at MPI for suggesting me to team up with Erik Reinhard at MPI over the 2012 summer and for other suggestions;
- the Norwegian government for giving me money;
- Search Press Limited for allowing me to use their images (Figures 5.3 and 5.6) from a book[17] on chiaroscuro.

Contents

Chapter 1

Introduction

Digital design and creation of 2D imagery targets many applications, ranging from industrial and medical problems to artistic applications. The research presented in this dissertation involves the investigation of novel approaches to creating 2D imagery with surface modelling technologies, specifically targeted towards artistic applications. Such modelling technologies have proven instrumental in 3D digital surface design, due to their flexibility and ease of use. My research question is whether this flexibility can be of advantage for applications within 2D image creation and processing.

The motivation of my work is spurred by the increasing popularity of desktop and mobile computers, making digital art creation potentially more accessible than traditional creation techniques. Practitioners of fine art, including artists such as David Hockney, are experimenting with mobile computers as a substitute for the traditional paint brush and the plain-woven canvas. For example, at an exhibition at the Fondation Pierre Bergé - Yves Saint Laurent, Paris (2010), Hockney exhibited dozen of Apple iPads displaying digitally painted depictions of flowers[34]. The novelty of the exhibition, in addition to using digital canvases, was that the paintings were sporadically updated, adding and replacing new flowers into the paintings.

Hockney uses iPads and iPhones for creating art work for several reasons. A particular advantage, Hockney points out, is the ease of sharing[34]: 'you can make a drawing of the sunrise at 6 AM and send it out to people by 7 AM . . . at 8 AM they look at a very fresh picture of the sunrise two hours earlier'. Of course, we see this daily through social media, such as Facebook and Instagram, where millions of novice 'artists' share their drawings and stylised photographs with their friends.

While the accessibility of art creation increases and sharing of art work has improved with digital technology, the creation process itself is far from perfect. Digital tools for creating graphics typically emulate traditional painting media and support additional features such as undo and layering. Whether such digital tools achieve this simulation is a hot debate among artists[i]. Those in favour for a digital revolution envision that computers will eventually take over all aspects of the fine arts and argue that new computer technologies essentially will make

[i]There are many online discussions on traditional vs. digital art creation. See for example: graphicssoft.about.com/u/ua/designandcreate/Digital-Art-Versus-Traditional.htm.

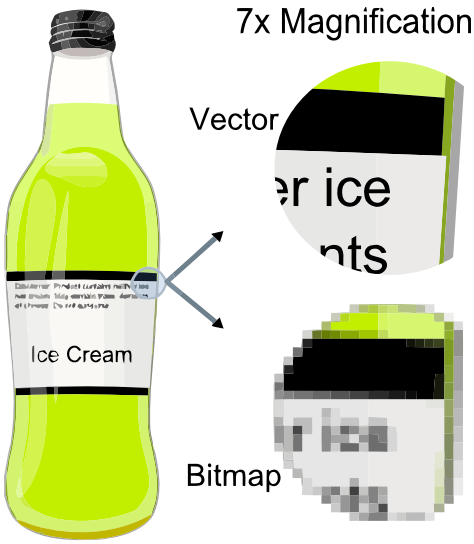


Figure 1.1: An advantage of vector graphics over pixel-based graphics is that vector graphics is scalable, producing crisp images at any resolution. Original image from Wikipedia (http://en.wikipedia.org/wiki/File:VectorBitmapExample.svg).

everybody artists. On the other side, the conservative argues that computers will never be able to simulate the physical nature of painting, such as feeling the bumps in the paint, the roughness of the canvas, working with real colours, and so on.

Whether the applicability of digital technology to art is only of passing interest to the hobbyist, it is of more importance to the digital professional, who uses computers for their artistic jobs. As digital distribution and communication have enjoyed a rapid increase of popularity in the past two decades, digital art creation and design have evolved into major industries. While professionals, who make their living from this demand, are able to efficiently create art with commercial products and tools like Adobe Photoshop and Illustrator, they are always looking for novel and interesting ways to solve their problems because many things that they would like to do are time consuming to achieve, even with these highly polished products.

Research in this area targets not only the problem of simulating traditional techniques, but also explores new ways in which artists interact with the digital framework. Such research aims to expand the suite of tools found in commercial products by improving hardware, software, or the underlying mathematical frameworks defining the image. My research falls largely in the last category.

Scope

I have focused my research on *vector* graphics. As opposed to a typical pixel image, defined by a grid of pixels, a vector image is defined by a sparse set of primitives. These primitives can describe specific mathematical functions (circles, curves etc.), along with user-defined attributes such as colour and shape. The final (pixel) image displayed on the screen is rendered by the software using rasterisation algorithms for each type of primitive.

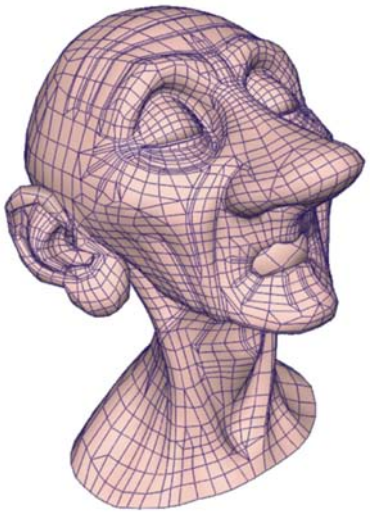


Figure 1.2: A subdivision surface defined by a control mesh. Image from ref. 23.

The major advantage of vector images is that they are scalable. That is, any given primitive can be scaled to fit any display. Thus, there is no need to resample the original image, as with pixel images, which produces visually different images compared to the original (Figure 1.1). Vector images are increasingly popular for this scalable property due to the rise of smartphones and tablets with different types of displays and image resolutions. Moreover, vector images better support the zoom operation, especially employed by digital artists, as the image can be rendered accurately at any level. By contrast, pixel images struggle with quantisation artefacts when the image size (number of pixels) is reduced.

As with most mathematical frameworks, there are also some drawbacks. The major disadvantage of vector images is that there is no direct mapping to the final pixels on the display. By contrast, pixel-based drawing supports more direct mapping as pixels can be directly 'painted' onto the pixel image. Thus, simulation of coloured brush strokes and plain-woven canvases is easier to achieve with a pixel-based framework due to the direct relationship between the strokes and the underlying model. For vector graphics, such strokes, including their associated colours, must be transformed into a compact, well-defined, solution which defines a rasterisation procedure that faithfully visualises the relationship between the original strokes and the canvas. Consequently, initial vector-based drawing suites were restricted to relatively simplistic colourings via directionally-fixed colour gradients.

The current technology supports two types of approaches for defining smooth colour gradients: diffusion curves and gradient meshes (see Section 2.2 for details). Diffusion curves require freeform curves as input that are associated with colour attributes. The rendering of diffusion curves is performed by diffusing the colours at the curves to the rest of the image domain. On the other hand, gradient meshes require *control meshes* as input. A control mesh is composed of control points and edges, forming a set of connected faces (Figure 1.2). In gradient meshes, control points are associated with colours and derivative constraints. The rendering of gradient meshes is performed by rendering a specific type of freeform surface. This surface is defined by Ferguson patches (see Chapter 3 for details).

Contributions

Diffusion curves and gradient meshes are incompatible. Diffusion curves only support freeform curves as input and are defined using a mechanism that, unfortunately, makes its behaviour difficult to predict (Chapter 5). By contrast, gradient meshes support local control (Section 3.1), meaning that colour edits propagate only to the local neighbourhood. Gradient meshes are related to technology originally developed for industrial design, where local control is important. For example, if a designer alters the front bumper of a car, the roof should not change. Such local control is not guaranteed with a diffusion process. On the other hand, gradient meshes are strictly defined with control meshes of rectangular topology. This restriction has limited their use as a tool predominantly employed by experienced digital artists (Chapter 6).

In this dissertation, I narrow the gap between diffusion curves and gradient meshes. I:

- propose a new curve-based vector primitive, similar to diffusion curves, with local control;
- use a curve-based vector-centric approach to propose several novel applications to image processing and manipulation;
- propose a new mesh-based vector primitive that is unrestricted to mesh topology.

My new curve-based vector primitive (Chapter 5) supports local control. The practical advantage of my method, compared to diffusion curves, is that the gradient of the colour function out from the curve can be explicitly specified without propagating the effect beyond the local neighbourhood of the edit. To achieve such behaviour, I render my primitive as a freeform surface. Thus, the specification of my primitive is akin to a diffusion curve, being related to freeform curves, but the rendering of my primitive is akin to gradient meshes, being defined as freeform surfaces.

To render my curve-based primitive, a map from curves (1D manifold) to surfaces (2D manifold) must be defined. In Chapter 4, I present solutions to create 3D control meshes from freeform curves. These control meshes are then used to render my curve-based primitive (Chapter 5) with surfaces. Furthermore, I present additional applications for these control meshes. In Chapter 7, I propose a framework, using a solution presented in Chapter 4, to automatically enhance the perceived contrast in photographs. The challenge of this problem is to enhance the perceived contrast whilst retaining the original look of the image. I show that there are inherent limitations in previous image processing techniques, where artefacts related to saturation are created depending on the image content. With the use of freeform surfaces, more local and explicit control of the contrast enhancement effect can be achieved. As a result, my method does not produce saturation artefacts.

While the contrast enhancement method is automatic, the framework presented in Chapter 7 is highly adaptable. In Chapter 8, I show that this framework can be employed in the setting of user-adjustable artistic image processing. I demonstrate novel imagery that would be challenging to create with traditional pixel-based techniques.

My mesh-based primitive (Chapter 6) is unrestricted to the topology of the input control mesh. I argue, as with many other researchers[53;57;58;72;110], that gradient meshes' restriction to rectangular grids hinders its full potential in many applications (Section 6.1). To this end, I propose

solutions that emulate the behaviour of gradient meshes in the regular setting and that extend this behaviour to the irregular setting. As well as having an immediate impact on the practical usage of gradient meshes, this contribution also stimulates future work towards a superset framework that both supports diffusion curves and gradient meshes.

In summary, I present a wide-ranging study of surface modelling for 2D imagery. I make the following contributions:

- Chapter 4:
  - I present a novel framework for creating control meshes from curves associated with attributes to control the shape of the resulting surfaces.
  - In contrast to related work, my solutions are robust to the topology of the input curves.
- Chapter 5:
  - A new vector primitive for defining smooth colour gradients out from curves.
  - A new method to control lighting and shading effects in vector graphics using boundary and slope curves.
  - Local control (via minimal support property) not demonstrated by previous diffusion curve methods.
  - Interactive performance is more easily achieved with the proposed framework compared to similar approaches that are based on partial differential equations.
- Chapter 6:
  - I present a new vector primitive for defining smooth colour gradients with control meshes of arbitrary manifold topology.
  - New interpolation schemes, using subdivision, that guarantee interpolation of original colours, do not stray outside the colour space, are smooth, and adapt colour gradients according to directional colour weights.
- Chapter 7:
  - I present the first approach to apply vector-centric image processing to contrast enhancement.
  - My solution does not produce saturation artefacts introduced by previous approaches.
  - In user trials, my solution is significantly preferred over a state-of-the-art contrast enhancement method.
- Chapter 8:
  - I demonstrate novel approaches for user-assisted artistic image processing to render several types of effects out from image edges.

My research builds on previous work on vector graphics (Chapter 2) and freeform-surface design (Chapter 3). The next two chapters outline the main ideas and theories in these two fields.

Chapter 2

Background

In this chapter, I present a brief overview of vector graphics and problems of current interest. In Section 2.1, I present a brief presentation of the history of vector-based technologies, ranging from vector displays, popular from the 1950s to the 1970s, to current applications such as graphic design. A major problem in vector graphics is to provide mathematical frameworks suitable for colouring. In Section 2.2, I present solutions to this problem.

2.1 Background of vector-based technologies

Early computer display and interaction systems were vector-based. A prominent example is the US SAGE air defence system[84] developed during the 1950s to track Russian planes. Using 'pens' (light guns), US Air Force officers would point, towards a cathode-ray tube (CRT) display, and click on a position that represented a potential target plane. A second click would update the trail of the target on the screen. The course and speed of the target could then be established and an interceptor aircraft would be vectored to meet and identify the target. This vector drawing process led to the name 'vector graphics' (ref. 91, Chapter 13).

In the early 1960s, Ivan Sutherland used a similar computer system (Lincoln TX-2) at the Massachusetts Institute of Technology, USA, to propose a revolutionary sketching system named Sketchpad[96] (Figure 2.1). Underlying primitives, such as lines and arcs, were drawn by clicking with a pen on the CRT display (as with SAGE). Once a drawing was created, the user could force equalised angles, make lines equilateral or parallel, or resize the drawing. Such interaction with the computer was revolutionary. At the time, human-computer interaction was mostly concerned with the creation of punch cards to be run on a mainframe system overnight. The idea of directly interacting with the computer, as demonstrated by Sutherland, was therefore novel. The system was also pivotal in the development of computer graphics and computer-aided design in the way the computer augmented the skill and performance of the designer.

Sutherland's system spurred companies to develop commercial vector display systems for vector drawing[47]. A vector display allows the CRT electron beam to be moved freely on the image plane. The electron beam follows a path defined by a given set of vectors, typically stored

Figure 2.1: Ivan Sutherland demonstrates his Sketchpad system. Image from the Massachusetts Institute of Technology.

in a display list on the computer's memory, to produce the entire image. The beam is turned off whilst skipping from the end of one vector to the start of the next, thus allowing the set of vectors to define a sequence of disjoint polylines. While the technology enthused general observers, the cost of such computer systems restricted their usage to larger companies and research laboratories. For example, the IBM 2250, a typical vector graphics system available in the mid-60s, cost around $280 000. The first low-cost vector graphics systems were released in the early 70s (for example, Imlac PDS-1 at $8 300 in 1970 and DEC GT40 at $11 000 in 1972).

A major disadvantage of vector display systems is the challenge of avoiding refresh flicker when many vectors are drawn on the screen. Additionally, vector graphics systems could only provide a limited type of shading (for example, via half toning). For these reasons, graphics systems started to employ raster displays when frame buffers became practically available in the late 70s.

While raster graphics received more attention in graphics research in the early 80s with the invention of advanced rendering algorithms like ray tracing, vector graphics continued to develop in commercial settings. In electronic printing, typography, and desktop publishing, inventing standardised ways of defining page images became important to address the development of the laser printer and the rise of personal computers. To this end, PostScript was established in 1982 as a programming language that was defined in terms of vector primitives. These could then be rendered to a pixel image at any desired resolution. The introduction of this language was important because it provided a unified framework to present documents containing lines, arcs, and (cubic Bézier) curves at any resolution.

With the growing popularity of PostScript in the mid-and-late 80s, software companies started to release vector-graphics editors to create and manipulate PostScript files. The intended purpose of these editors was not only to edit fonts, but also to provide general vector editing for applications like logo and product design. Notable editors include: Adobe Illustrator (1987), Corel CorelDRAW (1989), and Aldus FreeHand (1988; development discontinued in 2005).

Similar to the need for standardisation in printing, a comparable standardisation issue arose

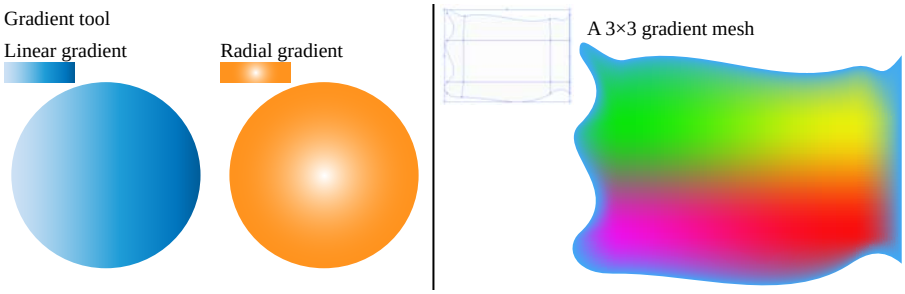


Figure 2.2: Tools available in Adobe Illustrator to manipulate colour propagation. With the gradient tool, the user manipulates a univariate colour function illustrated with a colour bar. This function is applied uniformly in a linear or radial direction. The gradient mesh tool enables users to manipulate colours at vertices of regular meshes to specify varying colour propagation in the object interior.

with the development of the Internet in the 90s: Internet browsers should follow the same guidelines when rendering vector graphics. To resolve this issue, the World Wide Web Consortium, the international standards organization for the World Wide Web, released the Scalable Vector Graphics (SVG) file format in 2001. SVG is more powerful than PostScript with support for additional features like web-specific scripting (similar to JavaScript) and animation.

Today, vector graphics tools enjoy substantial popularity among artists and designers. The scalable property of vector graphics makes this technology especially suitable for settings such as product design and cartoons. Advanced tools, described in the next section, enable experienced artists to draw photorealistic imagery. Additionally, the use of the SVG file format on the Internet is increasing. To this end, the latest smartphones and recent versions of Internet browsers now support the SVG file format.

2.2 Colouring vector-based graphics

The difficulty in defining detailed colourings with vector graphics is a major reason to the preference of raster graphics in many applications. For example, producing photorealistic imagery, with for instance a camera sensor, is more tractable with rectangular, dense, sampling (raster) rather than converting the signal into mathematical expressions (vector). Additionally, signal-processing theory is established and well-understood effects can be achieved with the wide suite of filters available. On the other hand, many difficult problems in image processing, like image re-sampling, are bypassed in vector graphics. For these and other[73] reasons, a wide range of research has been conducted to improve on colouring and imaging aspects of vector graphics.

2.2.1 Commercial solutions

One of the earliest methods for applying varying colour across enclosed vector objects was the gradient (Figure 2.2(left)). Such gradients are applied in a linear or radial direction. For example, the linear gradient illustrated in Figure 2.2 is applied along the path from left (white) to right (blue). Mathematically speaking, the gradient tool provides an editable univariate colour

Figure 2.3: Drawing coloured images with first-order diffusion curves (top); images from ref. 73, and second-order diffusion curves (bottom); images from ref. 31.

function (a curve in colour space). This colour function is manipulated by the user with colour points and weights. The function is illustrated with a colour bar, as shown in the figure, and is rendered, in a uniform manner, along a given path (linear or radial) inside the vector object.

The gradient mesh tool (in Illustrator; from version 9, 1998) and the mesh fill tool (in CorelDRAW; from version 9, 1999) enable users to manipulate colours at selected locations inside the object interior (Figure 2.2(right)). These locations are connected in a mesh, called a gradient mesh, where colours and first derivatives at the vertices of the mesh can be manipulated. Mathematically speaking, a gradient mesh is a rectangular grid with colour and derivative constraints associated to its vertices. The technical details of the tools have not been published and one can therefore only speculate on how they are rasterised. A reasonable assumption is that a spline surface, defined by Ferguson patches, is used to interpolate the mesh. I elaborate more on this assumption in Chapter 3.

2.2.2 Related work in research

In 2008 (ref. 73; Figure 2.3(top)), a novel vector primitive was proposed: it associates colours to each side of Bézier curves. To render the primitive, the curves are discretised as boundary conditions for a partial differential equation[i] (PDE). These boundary conditions define the value (Dirichlet conditions) the solution needs to satisfy at the given boundary. That is, the solution must be defined with the given colour values at the curve positions. The primitive is rasterised by solving the Laplacian equation ($\nabla^2 f = 0$), utilising Laplacian diffusion which produces a harmonic colour function (that is, a colour function that is 'as constant as possible'). Due to such diffusion of colours, the primitive is known as a *first-order diffusion curve*, or simply a diffusion curve.

[i] A PDE is an equation that contains an unknown multivariable function and its partial derivatives. A solution to a PDE is generally not unique and additional conditions are typically specified on the boundary of the domain.

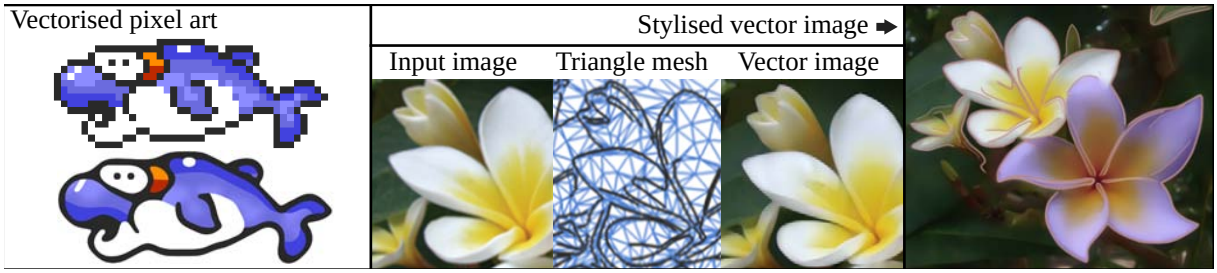


Figure 2.4: Vectorising raster graphics can have practical and artistic applications. Images adapted from refs. 51 (fish) and 58 (flowers).

The proposal of diffusion curves has inspired many researchers to either extend the primitive or to apply the primitive in novel settings[72]. Extensions to the primitive include improved texture design and draping[107], improved colour and texture parameter estimation[42], and diffusion constraints for control over diffusion strength, anisotropy and orientation, and diffusion barriers[4]. Diffusion curves have also been investigated in other settings than vector graphics, like geometric modelling of terrains and displaced surfaces[41;36] and volumetric modelling[98].

In 2011 (ref. 31; Figure 2.3(bottom)), the diffusion curve primitive was considerably altered by redefining it for bi-Laplacian diffusion (that is, solving the equation $\nabla^4 f = 0$, thus extending the primitive to *second-order diffusion curves*). In addition to value constraints, conditions on the first derivative can be defined (defining Neumann or Cauchy boundary conditions). 'Special' curves were introduced to constrain the first derivative to zero either along the curves or across the curves. This diffusion process defines a function that is 'as harmonic as possible'.

The second-order diffusion curve primitive is more involved than its predecessor, both representationally and computationally. In recent years, research in this topic has been concentrated around computational aspect of diffusion curves[10;40]. While progress has been made, the technology is not suitable for interactive applications, at least not on computers like smartphones, where performance is in the order of a few seconds[40].

A related problem within vector-based graphics is to map 2D pixel images to a vector-based representation (Figure 2.4). In the setting of photographic imagery, current solutions transform the image to 5D control meshes (2D in image plane; 3D in colour) and render the vector graphics with interpolation methods. The type of mesh relies on the interpolation method used. For example, if a gradient-mesh-like interpolant is used[95;53], then the meshes must be defined with rectangular topology. A related application is to vectorise pixel art[51] (Figure 2.4(left)).

In summary, there are two main approaches for rendering complex colourings for vector graphics. The first approach is related to control meshes and frameworks to interpolate colours over such meshes. Such interpolation is typically related to B-splines (introduced in the next chapter). The second approach is related to diffusion curves and the PDEs employed for rasterisation. My ambition is to create a superset framework that covers both of these two approaches. While I do not propose such a framework in this dissertation, my contributions close gaps between diffusion curves and mesh-based vector graphics. In Chapter 4, I propose solutions to creating meshes from curves for applications such as diffusion-curve-like vector editing (Chapter 5) and novel vector-centric image processing (Chapters 7 and 8). Additionally, I propose gradient meshes of arbitrary topologies to handle meshes extracted from curves (Chapter 6).

Chapter 3

Surface modelling technologies

Vector graphics rely on mathematical tools developed for computer-aided design (CAD). For example, the work presented in this dissertation both builds upon and extends these tools. One of the key challenges in CAD is to define surfaces that interpolate control meshes. Conversely, the purpose of control meshes is to provide a simple type of control mechanism that approximate surfaces with certain properties. In this chapter, I present a brief introduction to the tools employed in related work in vector graphics and in the work presented in this dissertation.

3.1 Popular types of parametric curves and surfaces

Coons patches

A well-known problem in CAD is: given four boundary curves, find a parametric surface where these curves form the boundary of that surface. A solution to this problem is to bi-linearly interpolate the curves. This solution was proposed by Coons[19] in 1964 and the resulting surface is known as a *Coons patch* (Figure 3.1(left)).

Coons patches have been employed in many applications within computer graphics. In vector-centric graphics, for example, Coons patches have been used to interpolate estimated normal vectors at curves in concept sketches[92]. The problem of extracting normal fields for vector graphics is discussed in Chapter 5.

Ferguson curves and patches

In 1964, Ferguson[29] proposed a framework to manipulate curves with control polygons, where derivative constraints at each control point can be edited. His interpolation framework, producing *Ferguson curves*, defines cubic parametric C^1 spline[i] curves. This framework has been widely adopted in vector graphics to edit curves. For example, Illustrator's curve editor uses

[i] A polynomial spline is a polynomial function that is defined in a piecewise manner.

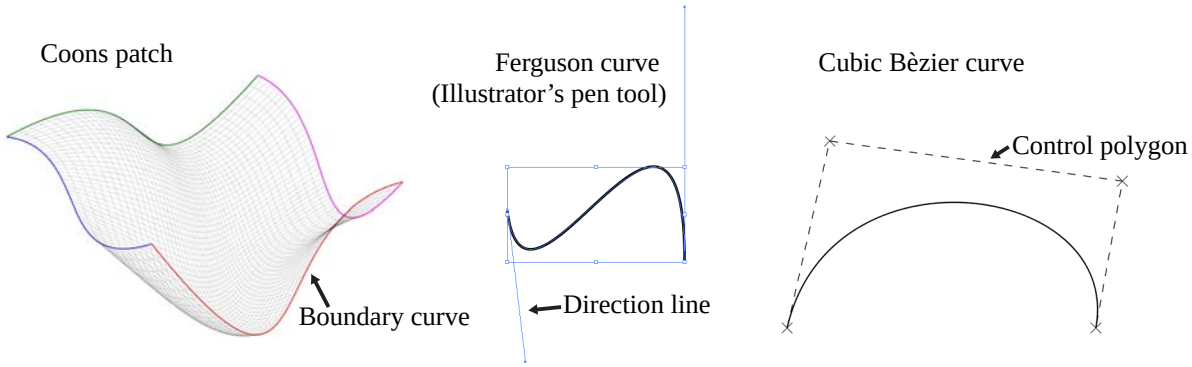


Figure 3.1: Three types of parametric curves and surfaces.

this approach. In its literature[ii], curve points are referred to as anchor points and derivative constraints are referred to as direction lines (Figure 3.1(middle)).

Ferguson extended the framework to surfaces by a tensor-product formulation of his curves[29]. Thus, *Ferguson patches* are defined with a rectangular grid, where each control point of the grid is associated with derivatives. The surfaces are defined as bi-cubic C^1 spline surfaces.

Ferguson's framework covers all aspects of the gradient mesh. In Illustrator, for example, vertices in gradient meshes are defined with the exact same definition as anchor points in curves, with the additional option of being associated with a colour. Moreover, the mesh is constrained in Illustrator's user interface to be rectangular. Thus, Illustrator's gradient mesh can be rasterised by employing 5D (2D in position, 3D in colour) Ferguson patches. This conclusion has also been made by other researchers[95;53].

Bézier curves and surfaces

As human-computer interaction revolutionised in the 1960s, research and development departments in the automobile and aeroplane industries started to build computer systems for industrial design. An important goal was to provide a framework that is intuitive for the designer, whilst supporting complex designs. An approach that arose from this work (for example, by Pierre Bézier at Renault and Paul de Casteljau at Citroën) was to provide intuitive mappings from the parametric space of the curve to the spatial domain. More specifically, let there be a map from the parameter space of the curve or surface, $\mathbb{R}^p$ ($p = 1$ for curves and $p = 2$ for surfaces), to the spatial domain $\mathbb{R}^n$. Then, a curve (or surface) can be constructed as linear combinations of curves (or surfaces) in $\mathbb{R}^p$ with a set of control points in $\mathbb{R}^n$. Such a framework establishes an intuitive synergy between the designer (working with control points) and the mathematical framework (producing parametric curves or surfaces).

Bézier curves (Figure 3.1(right)) and surfaces are prominent examples of such frameworks. In the setting of curves, the curve is defined by linear combinations of the control polygon with a set of basis polynomials, known as Bernstein polynomials. Bézier curves are defined as parametric polynomials with a degree equal to the number of control points minus one. Bézier

[ii] http://help.adobe.com

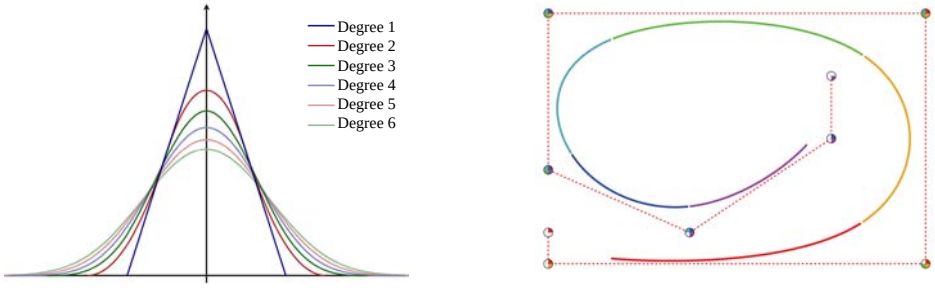


Figure 3.2: Influence of basis functions and control points of B-splines. Left: basis functions from degree 1 to 6 with uniform knot spacing. Right: a cubic B-spline curve. Each polynomial piece of the spline curve is shown in a different colour and the colours within each control point show which pieces of the curve are affected by moving that control point.

surfaces are defined as the tensor product of the definition of Bézier curves. An important difference to Ferguson's framework is that the resulting shapes of Bézier's framework, in general, approximate their control points (that is, the function does not 'pass through' the control points). By contrast, Ferguson's splines always interpolate their control points (that is, the function always 'passes through' the control points). Consequently, Bézier's framework is classified as an *approximating* method, while Ferguson's framework is an *interpolatory* method.

An important property of Bézier curves is that the start (and end) of the curve is tangent to the first (and last) edge of the control polygon. This means that a similar framework to Ferguson curves can be created with cubic Bézier curves. That is, a cubic Bézier curve will have two end control points, adjusting the position of the curve, and two internal control points, adjusting the first derivative at the end points. This property makes cubic Bézier curves suitable for curve editing, as cubic Bézier curves can be used as an alternative to Ferguson curves. Interestingly, many vector editing applications, like Illustrator, employ Ferguson's framework for editing; that is, 'direction lines' (derivatives) associated with control points are used instead of control polygons. However, the curves are probably evaluated as cubic Bézier curves, according to the various vector graphics standards like PostScript and SVG.

B-spline curves and surfaces

B-spline curves (B for 'basis') bear similarities to Bézier curves in that control polygons interact with basis functions to produce parametric curves. In contrast to Bézier curves, which are (single) parametric polynomials, B-spline curves are, in general, parametric splines (piecewise polynomials). Note that Bézier curves represent a subset of B-splines.

An important theorem of B-splines is: Any spline function of a given degree and continuity can be uniquely represented as a linear combination of B-splines of that same degree and continuity[22]. B-splines can therefore be referred to as a class of basis functions which are linearly combined to form splines of given degree and continuity. Figure 3.2(left) shows various basis functions for degrees 1 to 6. The degree of a B-spline can be increased, increasing the order of the polynomial pieces, to make each basis function have larger influence of the curve.

In addition to the control polygon and degree, other, more advanced, aspects can be specified.

Such aspects relate to the placement of the *knots* (the positions joining the polynomial pieces) in parameter space. B-splines are defined to make the resulting splines as smooth as possible across the knots: a degree d B-spline has $d - 1$ continuous derivatives across each knot. Knots can be placed in a non-uniform manner in parameter space, which can adjust the shape, and coincident knots adjust the continuity of the spline. Note that I did not employ the advanced aspects of knot placement and I therefore do not discuss the fine details of knots.

The use of splines gives rise to several advantages over Bézier curves. The main two advantages of B-splines relate to polynomial order and to curve behaviour. The first advantage relates to polynomial order; more specifically, the order of the individual polynomial pieces can be specified. By contrast, the order of a Bézier curve is determined by the number of the control points provided. The advantage of this additional degree of freedom is that many control points can be used to specify splines of low degree (like quadratics or cubics). The second advantage relates to curve behaviour; more specifically, any point on the B-spline curve only relates to a given subset of the control points. This means that when a control point is manipulated, this manipulation only influences a portion of the curve (Figure 3.2(right)). By contrast, manipulating a control point of a Bézier curve will influence the entire curve.

A property that makes B-spline editing suitable for designers is the property of *minimal support*[22]; that is, B-splines are mathematically constructed so that each control point affects the minimum amount of the resulting spline curve. In practice, this means that B-splines allow designers to construct long, smooth, curves whilst still being able to modify a small region at a time. Note that diffusion curves, rendered with Laplacian diffusion, do not achieve such local control, which motivates the use of B-spline-related solutions in vector graphics. In Chapter 5, I discuss this minimal support property further.

B-spline surfaces are defined with the tensor product of the definition of B-spline curves. Thus, rectangular control meshes define B-spline surfaces. The advantages given above for curves also apply for B-spline surfaces, which make them popular in CAD.

Non-uniform rational B-splines (NURBS)

A problem with B-splines in the setting of CAD is that they cannot represent conic sections exactly. In 1975, Versprille[103] observed that *rational* B-splines can deal with this issue, as rational polynomials can represent conic sections without approximation. From this observation, Tiller[100] (1983) proposed the NURBS framework, extending B-splines to rational B-splines.

The practical addition of this extension is that a weight is associated with each control point, where larger weights of a control point attracts the curve or surface closer to that control point. This feature is used in Chapters 7 and 8 to adjust various effects in vector-centric imaging.

3.2 Subdivision curves and surfaces

Subdivision offers an alternative way to generate curves and surfaces: Given a control polygon or mesh, subdivide this input into a denser polygon or mesh (Figure 3.3). Such subdivision can

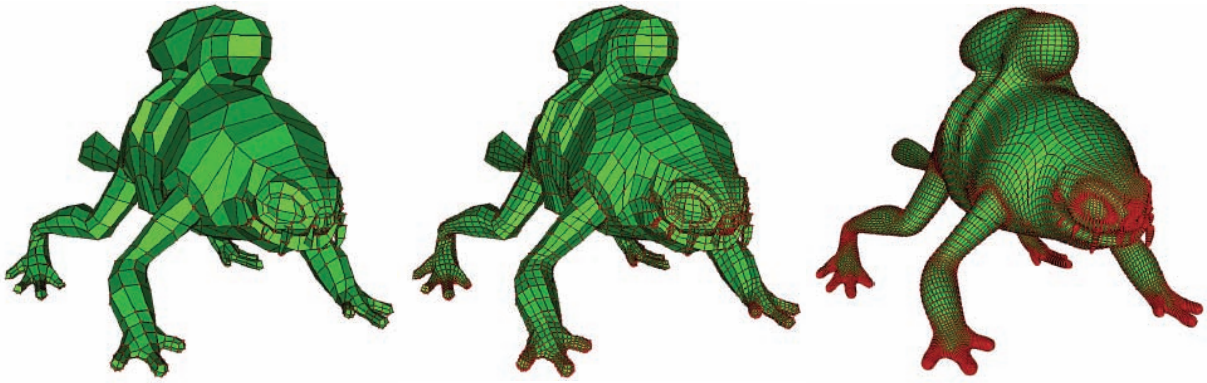


Figure 3.3: Iterative subdivision of a control mesh, using the rules of Catmull and Clark.

be performed recursively to produce smooth-looking curves and surfaces. Many subdivision schemes have been proposed to produce curves and surfaces of certain properties, which relate to the *limit* curve or surface of the scheme; that is, the curve or surface defined by an infinite number of subdivision steps. In the setting of regular meshes, Doo and Sabin[24] (1978) proposed a scheme to produce quadratic B-spline surfaces (scheme known as Doo-Sabin). The scheme of Catmull and Clark[12] (1978) produces cubic B-spline surfaces (scheme known as Catmull-Clark) and the scheme of Loop[59] (1987) produces smooth surfaces from triangle meshes.

The fundamental difference to the methods presented in Section 3.1 is that there is no explicit reference to the closed-form parametric representation at all. Instead, schemes can implicitly produce certain parametric curves and surfaces (this can be established by proof[24;12]). Since subdivision is not restricted to a parametric representation, subdivision rules for meshes with irregular connectivity can be used. For example, all of the schemes mentioned above, using 'special' rules at irregular mesh elements, produce smooth limit surfaces with irregular meshes. By contrast, methods based on tensor-product constructions, like B-splines, are restricted to rectangular meshes.

While subdivision provided obvious advantages related to freedom in topology, the technology was initially only employed in research. Instead, companies developing products for CAD and engineering decided to employ NURBS. Subdivision was not as attractive as NURBS because it did not provide the same freedom related to surface parameterisation and smoothness. These aspects were therefore deemed more important than freedom in mesh topology. In 1998, DeRose and colleagues[23] at Pixar Animation Studios demonstrated that in animation and film, freedom in topology can provide multiple practical advantages like planning times and sparser meshes (these advantages are discussed more in Chapter 6). DeRose et al. made their subdivision-based tool practical for animation by extending the Catmull-Clark scheme to support a smoothness parameter at edges, controlling the visual 'sharpness' across surfaces.

Subdivision covers a surprisingly small space in the vector-graphics solution space. The only method I am aware of employing subdivision is the method of Liao and colleagues[58] (2012) for vectorisation of images. The work presented in the subsequent chapters contribute to this space with a subdivision-based diffusion-curve-like framework (Chapter 5) and a subdivision-based gradient-mesh-like framework (Chapter 6). To achieve the former framework, meshes from curves must be defined. This problem is discussed in the next chapter.

Chapter 4

Creating 3D control meshes from curves

This chapter presents research described in the following papers:

Cornsweet surfaces for selective contrast enhancement
Shading curves: vector-based drawing with explicit gradient control

---

Surfaces defined by 3D control meshes can be intuitively designed in CAD applications for creation and manipulation of 3D models. However, such 3D modelling environments are not suitable for 2D image creation and editing. Control meshes defining surfaces must be abstracted in order to maintain the simplicity of a 2D manipulation interface. That is, we would like to abstract control meshes via typical input for 2D drawing interfaces, like curves or brush strokes.

In this chapter, I present solutions to the problem of creating 3D control meshes from curves defined on the 2D canvas. These control meshes are, in subsequent chapters, used to render surfaces for several vector-centric applications. The first two coordinates at a mesh point correspond to coordinates in the image plane and the third coordinate relates to the specific application. For applications within 2D imagery, this coordinate typically refers to modification in colour or luminance space. Figure 4.1 shows a simple curve, the resulting 3D control mesh, the corresponding surface, and the resulting image (modification in the green colour channel). To add degrees of freedom to the surface modelling framework, attributes can be associated with the curve. In this chapter, two attributes are used: *extent* defines the extent of the control mesh in the image plane in the perpendicular direction to the curve and *height* defines the distance of the control mesh in the perpendicular direction to the image plane.

I argue that there are two reasonable approaches to solving the problem of creating 3D control meshes from curves. The first approach involves a complete tiling of the given image domain. That is, the goal is to define a single mesh, or multiple connected meshes, defined across the entire domain. The second approach defines control meshes for each curve separately and does not guarantee a tiling of the domain. The first approach, meshing the entire domain is well studied. In contrast, the second approach is novel. To my knowledge, this particular problem has not been presented in the literature. However, closely related problems, targeted towards other applications not related to surface modelling, have been addressed (Section 4.1).

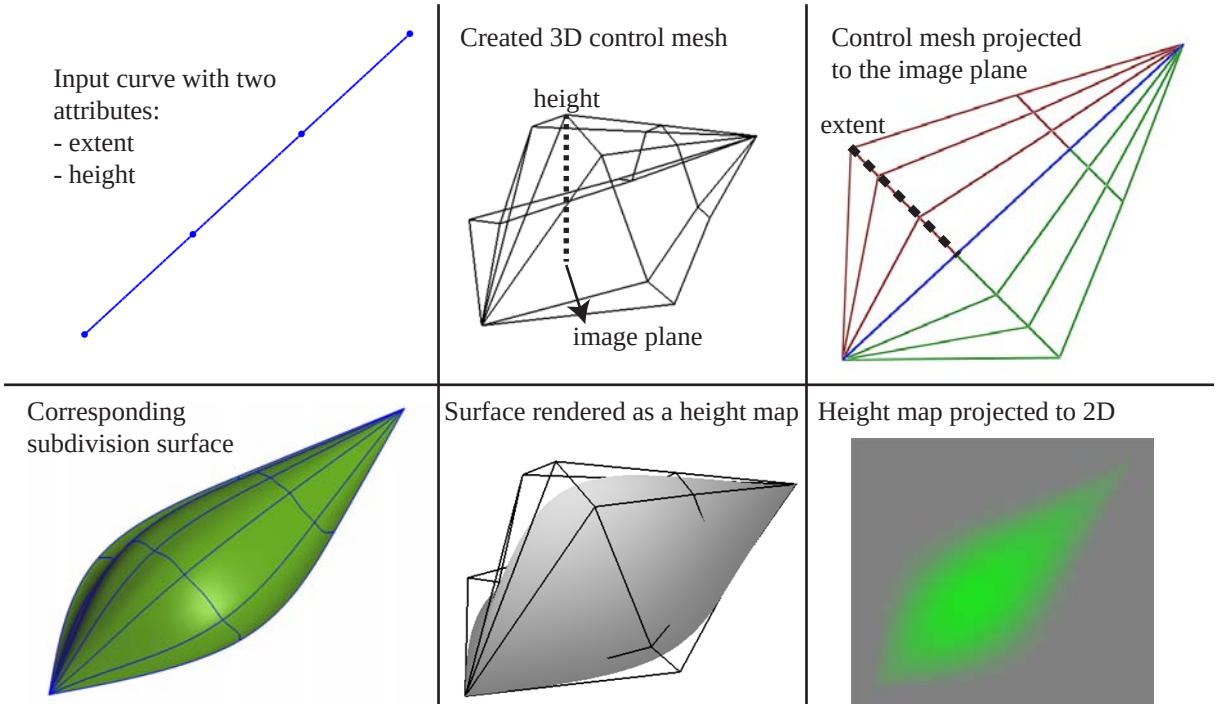


Figure 4.1: Creating a 3D control mesh from a curve. Attributes used to control the shape of the corresponding surface are associated with the curve. In this example, an extent parameter is associated with the extent of the mesh perpendicular to the curve and a height parameter is associated with the height of the mesh perpendicular to the image plane. The problem, presented in this chapter, is to define 3D control meshes from this set of input. In subsequent chapters, the corresponding surfaces are used to create and manipulate 2D imagery.

The main focus of this chapter is related to the second approach: creating 3D control meshes for each input curve separately, producing multiple disjoint meshes. In Section 4.1, I describe the problem in more detail and present related work. I have investigated two solutions to this problem, where I make slightly different assumptions for each solution. These two solutions are presented in Section 4.2. Finally, I discuss the potential of employing current state-of-the-art meshing methods, generally aiming to model 3D objects, in the setting of vector graphics (Section 4.3).

4.1 Background and related work

The specific problem of creating a 3D control mesh for each input curve has not, to my knowledge, been previously addressed in the literature. In this section, I present the motivation to this problem (Section 4.1.1), describe the problem in more detail (Section 4.1.3), and discuss related work (Section 4.1.4).

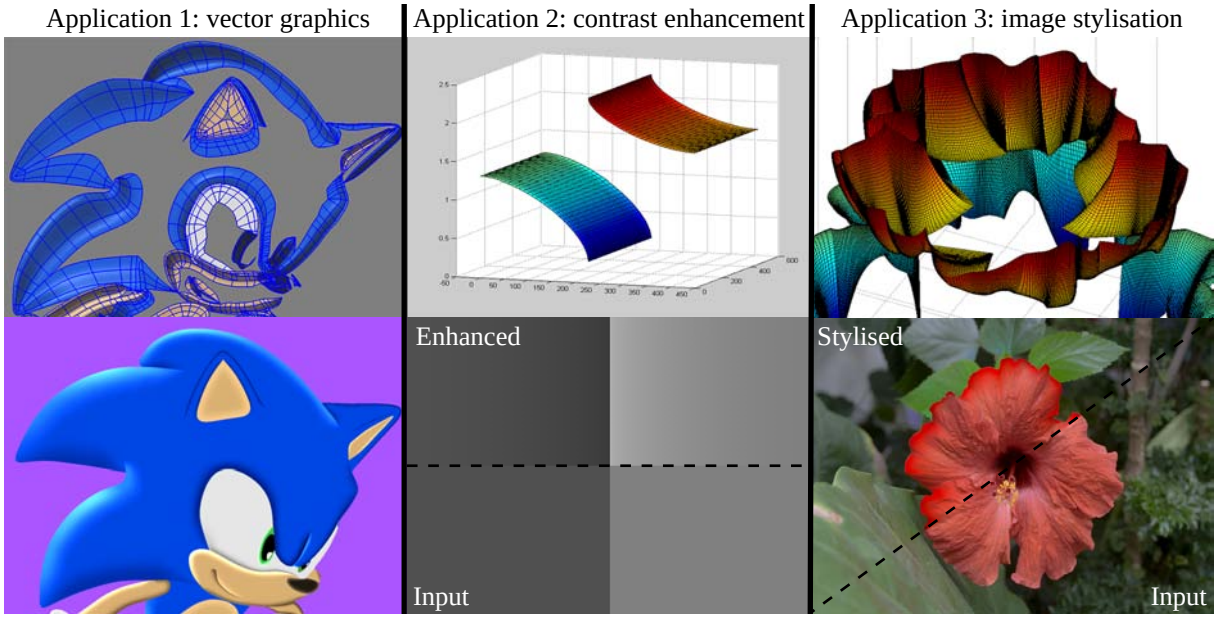


Figure 4.2: Applications I have investigated for modelling surfaces from curves. Top: surfaces rendered from the control meshes created with solutions presented in this chapter. Bottom: Images supplemented with these surfaces for various effects.

4.1.1 Motivation

The main motivation of the approach presented in this chapter is to provide a framework where end users, researchers, and developers can explicitly define and manipulate control meshes in the setting of vector-based graphics. To achieve this aim, I propose to attach attributes to the input curves. These attributes represent various aspects of the shape of the final surface. In contrast to this approach, standard meshing methods only employ curves as constraints to the meshing problem; other frameworks, such as vector or cross fields, act as the principal constructors of the mesh (see Section 4.3 for more discussion).

The overall aim of my approach is a surface modelling framework that is as direct as possible, without forcing users of the framework (that is, researchers, software developers, or artists) to be concerned with additional dimensions not related to the 2D image plane. That is, the 2D imagery should not be 'modelled' in a 3D environment, but instead be created and manipulated with input parameters tuned towards the 2D setting. As already mentioned, my solution to the problem of defining a framework that is as direct as possible is to associate attributes to the curves. In subsequent chapters, I demonstrate multiple ways to interact with images using such attributes (Figure 4.2). In short, I demonstrate vector graphics creation (Chapter 5) with additional sliders and curves, automatic contrast enhancement (Chapter 7) with fixed global parameters and pre-defined attributes, and artistic image processing (Chapter 8) with user-defined strokes influencing the attributes.

Finally, I note the main practical difference between my approach and a general meshing approach. As previously mentioned, the former produces a given set of disjoint meshes associated with the input curves, while the latter typically produces a single mesh covering the given domain. Since my approach does not guarantee a complete tiling of the domain, it is more

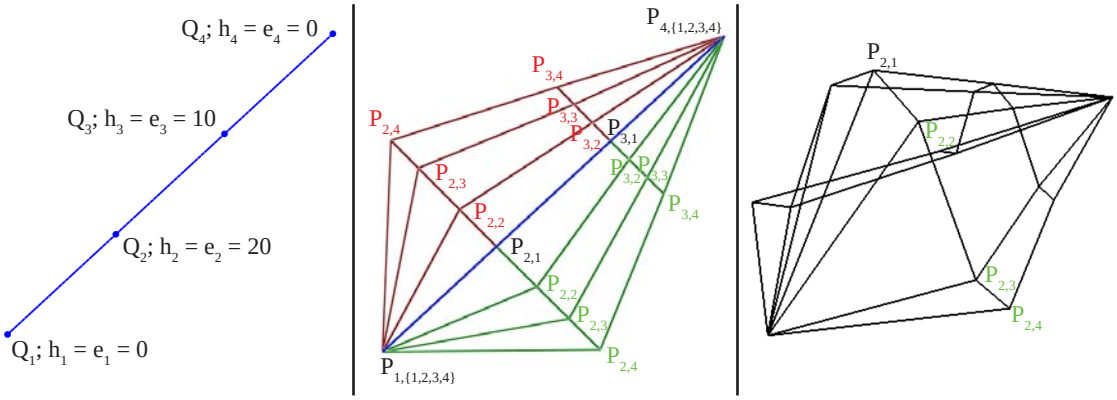


Figure 4.3: Notational labels added to the example given in Figure 4.1. There are no notational difference between the two control meshes associated with the curve. In this illustration, they are separated with red and green colours.

suitable for applications which do not require the entire image plane to be tiled with meshes. In subsequent chapters, I demonstrate that my approach is particularly useful when images are *supplemented* with surfaces to produce a desired effect.

Before discussing the meshing problem in detail, we need formal definitions of the input curves, including their attributes, and the output control meshes. These definitions are presented next.

4.1.2 Definitions

An input B-spline curve is defined by a sequence of n control points $Q_i = (x_i, y_i)$, $i = 1, \ldots, n$ (Figure 4.3(left)). The 2D unit normal vector of the curve at the position related to Q_i, computed from the curve's first derivative[22], is denoted N_i. It is assumed that the normal vectors are consistently oriented. Two attributes, height and extent, are attached to each control point:

- extent, $e_i \in \mathbb{R}_0^+$: defines the extent of the mesh in the direction N_i from Q_i.
- height, $h_i \in \mathbb{R}$: defines the height of the mesh in the perpendicular direction to the image plane at Q_i.

This set of attributes is not an exhaustive set and other attributes can be used if this is needed. In Chapters 5 and 7, an additional set of shape attributes are defined to manipulate the shape of the surface.

With the input curves and attributes defined, the output 3D control meshes are now described. The 3D control points derived from and associated with Q_i are defined as $P_{i,j} = (x_{i,j}, y_{i,j}, z_{i,j})$, $i = 1, \ldots, n; j = 1, \ldots, m$, where n is the number of control points associated with Q_i (Figure 4.3(middle)). In Chapter 5, $m = 4$ to model bi-cubic surfaces and in Chapter 7, $m = 3$ to model quadratic surface profiles. A rectangular grid of size $n \times m$ is initially assumed to be created out from the curve. However, some quads will, in Section 4.2, be degenerated to triangles.

The above definitions are only concerned with a mesh created in the direction N_i. A mesh in the opposite direction $-N_i$ can also be created. I do not include notations to algorithmically separate the two meshes as I have not found such explicit treatment necessary in the description of the solutions. That is, it is assumed that the same treatment is performed when constructing the mesh on the other side of the curve (thus redefining N_i to $-N_i$). Consequently, two sets of attributes are typically associated with Q_i (that is, one set of attributes on each side of the curve). These two sets can have different values and can therefore be manipulated separately. Note that there is no requirement that both meshes are created. In the framework presented in Chapter 5, for example, the user, via a prototype user interface, can manually enable or disable the meshes on each side of the curve.

The z, or 'height', coordinates are now defined. The control points along the original curve, $P_{i,1}$, are set according to the corresponding height parameter. We can assume that the effect of the surface, being adjustment in luminance, colour, or any other type of adjustment, will fair out to have zero effect at the 'other end' of the control mesh at $P_{i,m}$. That is, surface values of $z = 0$ do not alter the image. Given the related curve control points $Q_i = (x_i, y_i, 0)$, the following coordinates can now be defined:

$$P_{i,1} = (x_i, y_i, h_i); \tag{4.1}$$

$$z_{i,m} = 0. \tag{4.2}$$

Note that I have not seen any practical advantage of defining non-zero values of $z_{i,m}$ and I therefore constrain it to zero in this definition.

The coordinates $(x_{i,m}, y_{i,m})$ represent the 'outermost' control point of the mesh. A naïve definition of this point is:

$$(x_{i,m}, y_{i,m}) = Q_i + N_i e_i. \tag{4.3}$$

This solution, however, can give rise to artefacts when the surface is projected to 2D. In Section 4.1.3, I illustrate such artefacts and discuss why a robust, artefact-free, placement of $(x_{i,m}, y_{i,m})$ is challenging. Placing $(x_{i,m}, y_{i,m})$ is therefore the principal problem of this chapter. I present my solutions to this problem in Section 4.2.

The control points defining the rows 2 to $m - 1$ in the control mesh $(P_{i,\{2,\ldots,m-1\}})$ define the shape of the surface. These should be placed on the plane defined by $P_{i,1}$, $(x_i, y_i, 0)$, and $P_{i,m}$. The definition of $P_{i,\{2,\ldots,m-1\}}$ is described in Chapters 5 and 7 as this definition depends on the application.

The description so far has only been concerned with a single B-spline curve as input. Multiple curves are naturally supported and should be initially treated separately. The solutions presented in Section 4.1.3 do take all input curves into account when defining $(x_{i,m}, y_{i,m})$. If the set of curves are disjoint, no further treatment is necessary. However, if curves are joined at junctions, adjacent control meshes should be merged. Such merging is described in Section 4.2.

4.1.3 Problem description

As mentioned earlier, the principal problem faced in this chapter is to define $(x_{i,m}, y_{i,m})$, the 'outermost' control points of the mesh. To discuss this problem, we need some idea to what I

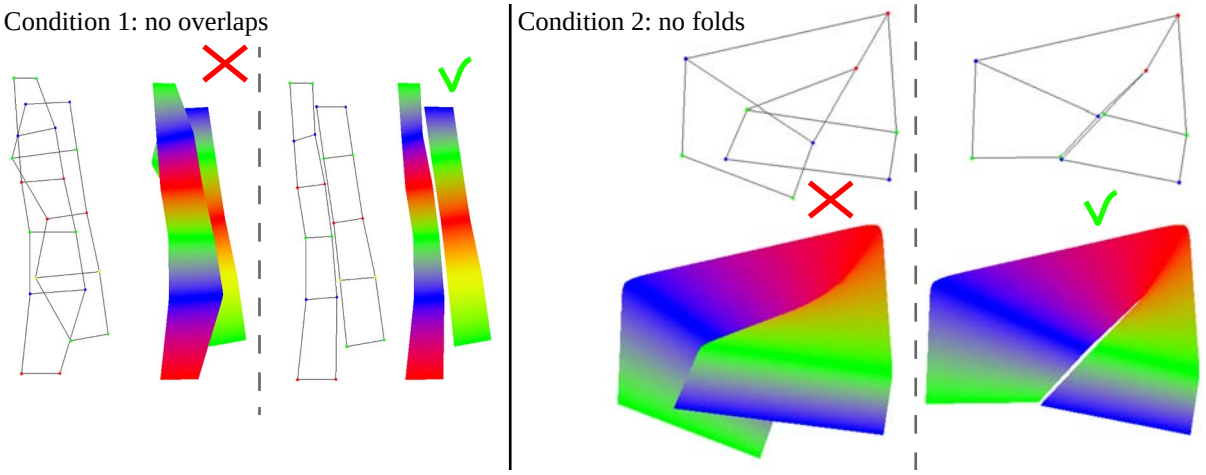


Figure 4.4: In this chapter, we assume that, when the 3D control meshes are projected to the (x, y)-plane, overlaps between meshes and folds are unwanted.

mean by a 'good' solution. Informally speaking, the control meshes should behave naturally and should not give rise to visual artefacts when applied to images. In the applications I have investigated, the two conditions below should be satisfied. When the control meshes are projected to the 2D image, they should:

- **Condition 1**: not overlap each other;
- **Condition 2**: not fold.

Figure 4.4 illustrates the visual artefacts related to the two conditions. Mathematically, when either of these conditions are unsatisfied, the resulting function defined by the projected surfaces gives rise to C^{-1} jumps. Such jumps should be avoided since they represent sharp transitions in the image, like image edges. Instead, the resulting function should be smooth across the image, unless jumps are specifically specified.

Note that we are only concerned with control meshes that have been projected to the 2D plane. This projection is defined by simply neglecting the z coordinate. To this end, I will, in the remainder of this chapter, refer to the mesh control points $P_{i,j}$ as 2D points with the coordinates $(x_{i,j}, y_{i,j})$. Thus, we aim to define the points $P_{i,m}$. Additionally, let the line $P_{i,1}$–$P_{i,m}$ be L_i and the cubic B-spline curve defined by $P_{i,m}$ be the approximate *offset curve* of the input curve. The offset curve is *open uniform*, meaning that it passes through its two end points $P_{1,m}$ and $P_{n,m}$.

The problem can be described as follows. Define offset curves that do not intersect with other offset curves (Condition 1) and do not self-intersect (Condition 2). Additionally, L_i lines should not intersect (Conditions 1 and 2).

The main issue with this problem is to define $P_{i,m}$ in regions where a naïve solution would produce (self)intersections between offset curves. I refer to these regions as *narrow regions*. Figure 4.5(bottom-left) illustrates a sensible solution for a relatively simple configuration of a narrow region: simply narrow the extent of the offsets to prevent the overlap.

Figure 4.5(bottom) illustrates an inherent difficulty related to narrow regions: they give rise to two types of scenarios. The first scenario is the normal case, where the placement of $P_{i,m}$ can

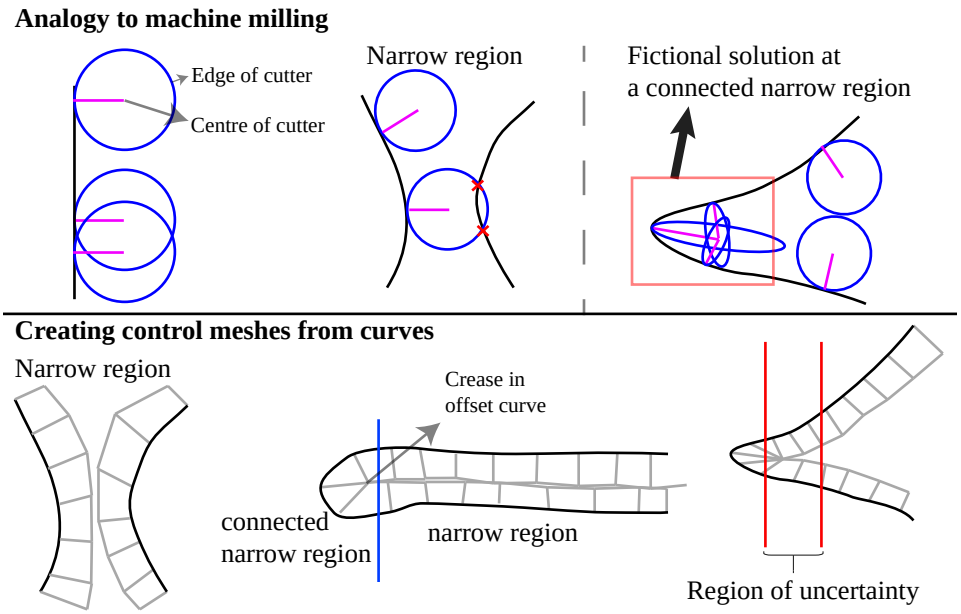


Figure 4.5: Top: the process of machine milling employs cutters to form shapes. If the cutter is too large for a narrow region, it will not be able to create the shape and a smaller cutter is needed. The fictionary solution shows the solution required by our problem at connected narrow regions. Such 'adaptive' ellipsoid-shaped cutters do not exist in reality. Bottom: control meshes at narrow regions. In connected narrow regions, triangular fans are created. This meshing problem is ill-posed since it is not clear how to differentiate between narrow regions and connected narrow regions.

be treated separately for each Q_i. The second scenario relates to regions where the curve is defined with high curvature. An alternative solution in such regions is to extend the placement of $P_{i,m}$ from Q_i. This will extend the offset curve to a point where L_i lines can intersect. Such intersections can be avoided by placing multiple $P_{i,m}$ at a single point, producing a triangular fan in the mesh out from this point. Due to these coincident $P_{i,m}$ at triangle fans, the continuity of the offset curve at those $P_{i,m}$ points is C^0. Since we connect multiple mesh points at such narrow regions, I will refer to these as 'connected narrow regions' (the regions from the first scenario are referred to as 'narrow regions'). To treat a narrow region as a connected narrow region has the advantage that $P_{i,m}$ is extended further, meaning that the related effect can be extended further than a solution related to a narrow region. This difference is discussed in more detail in Section 4.2.

It is not clear how to differentiate between narrow regions and connected narrow regions. More specifically, a narrow region can transform into a connected narrow region. This is illustrated in Figure 4.5(bottom-right), where, in the 'region of uncertainty', the region can be both treated as a narrow region and as a connected narrow region. This problem is therefore ill-posed since we are not guaranteed a unique solution. In Section 4.2, I deal with this issue by using various parameters to control the solution at connected narrow regions.

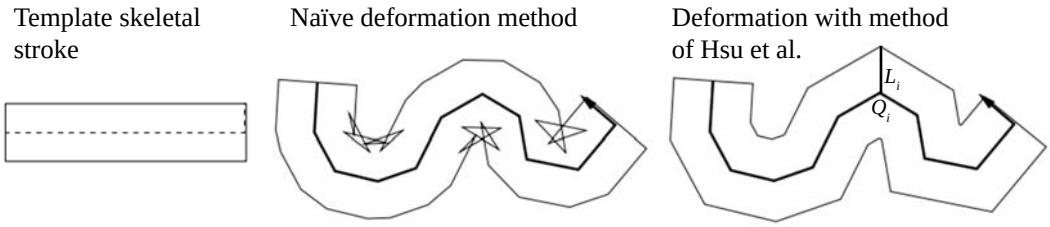


Figure 4.6: The problem of deforming skeletal strokes is similar to our problem. Naïvely deforming the stroke, according to Equation 4.3, can produce intersections between L_i lines. The deformation method of Hsu et al., using radii of curvatures, is more robust. Images from ref. 38.

Analogy to machine milling

An interesting analogy to this problem is machine milling: the process of using cutters to remove material of a workpiece to create machine parts in given sizes and shapes. Such machine milling faces similar problems because any given circular cutter might not 'fit' certain shapes (Figure 4.5(top)). This issue is dealt with automatically by using algorithms that both approximate the offset curve sufficiently accurately and identify regions where the given cutter does not fit. Thus, a narrow region corresponds to a region where the cutter is too large to fit the given shape. A solution is to use a sufficiently small cutter. A cutter at a connected narrow region would correspond to an imaginary cutter which is deformed to both fit the shape and be able to follow the offset path, as shown in Figure 4.5(top-right). See ref. 48 for an example of a robust solution to this milling problem (of course, not considering our 'special' connected narrow regions).

The problem faced in this chapter is not directly applicable to machine milling, meaning that the offset path algorithms constructed for machine milling are not suitable for the definition of $P_{i,m}$. Machine milling requires a certain accuracy of the offset curve since the offset curve defines the path of the cutter. As a consequence, the algorithms related to such path estimation are unnecessarily involved because our problem does not require an offset curve to be found at all. Instead, the points $P_{i,m}$ can simply be used to approximate an offset curve. This approximated offset curve does not have to be used explicitly to solve our problem. It could, however, be used to analyse the solution, by, for example, verifying whether the solution satisfies Conditions 1 and 2.

Before presenting my solutions, I discuss work related to the placement of $P_{i,m}$. While these methods are not related to the problem of creating control meshes, they are targeted towards various aspects of vector graphics. We will see that these solutions are relatively naïve and are oblivious to narrow regions.

4.1.4 Related work

Recall the naïve solution from Section 4.1.2: $P_{i,m} = Q_i + N_i e_i$. This solution has been employed to place additional offset curves to an existing second-order diffusion curve[31]. These offset curves are associated with boundary conditions, like a 'standard' (second-order) diffusion curve.

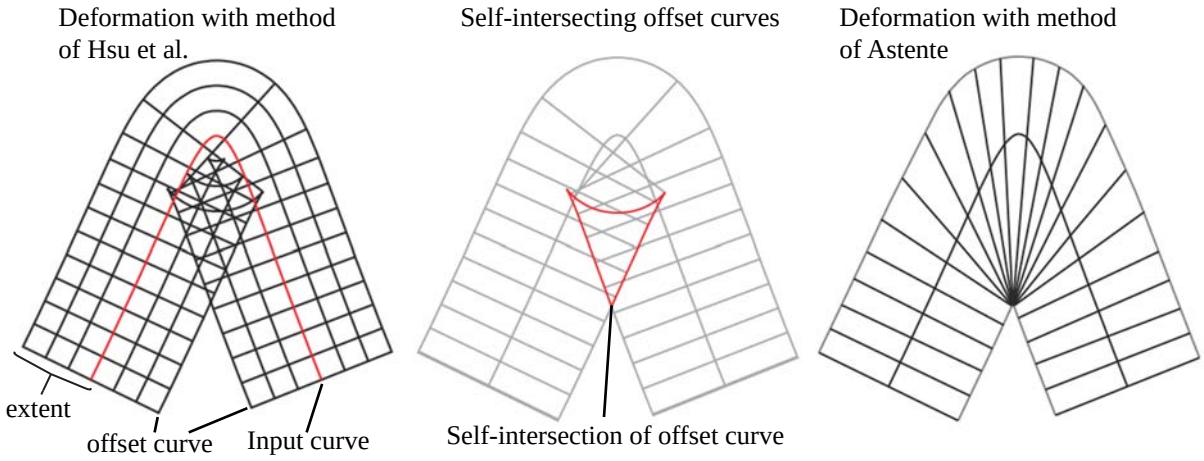


Figure 4.7: Folding avoidance with the method of Asente. Folds, produced by the method of Hsu et al., are resolved in two steps. First, areas of modification are identified along the curve where the offset curve self-intersects (area along the offset curve highlighted in red in the centre image). Then, L_i lines in the identified areas are rotated so that its related $P_{i,m}$ point is placed on the intersection point of the offset curve. Images from ref. 1.

This means that several types of boundary conditions can be associated with a single diffusion curve by offsetting additional conditions. While this solution works for sufficiently straight curves and smaller offsets, it can make the offset curve self-intersect.

The deformation model of *skeletal strokes*, by Hsu and colleagues[38], requires a solution to a similar problem. The skeletal strokes framework provides a way to transform a pre-defined image along a path. Such paths can be defined as polygons or curves (Hsu et al. assumes cubic Bèzier curves). The transformation of the pre-defined image is performed both along the given path and in the normal direction to the path. The latter deformation problem requires a definition of points along given extents in the normal direction of the path and can therefore be solved with Equation 4.3. Hsu et al. noted the limitation of such a naïve solution and proposed to constrain the extent, e_i, to the radius of the curvature of the curve point related to Q_i (Figure 4.6). The rationale for this restriction is that the radius of curvature is defined as the intersection point of two infinitely close normal lines to the curve. Thus, nearby lines L_i defined in this fashion will most likely not intersect.

While nearby L_i lines probably do not intersect, the solution is not robust to arbitrary configurations of curves (Figure 4.7(left)). This is because the solution does not take more global aspects into account, such as interactions between L_i of curve control points further away or of curve control points defined on other curves. Asente[1] has improved on this folding issue for skeletal strokes with a two-step approach: by identification of intersections and by resolving those intersections. The first step, identification, is performed by identifying the self-intersections of the offset curve defined by Hsu et al. Curve segments that contain intersections are defined by closed loops formed by intersection points in the offset curve (Figure 4.7(middle)). The second step, resolving intersections, is performed by repositioning the points $P_{i,m}$ to the related intersection point found in the previous step. This solution therefore degenerates the quads defined by neighbouring $P_{i,1}$ and $P_{i,m}$ to triangles, as shown in Figure 4.7(right).

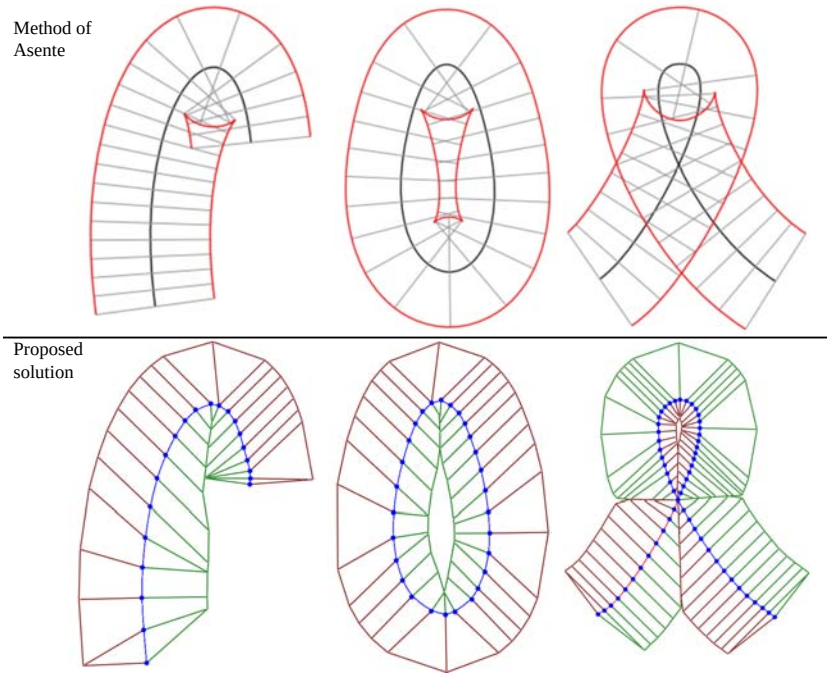


Figure 4.8: The method of Asente is not robust to the topological layout of the input curves. In contrast, the solutions presented in Section 4.2 *are* robust. Topmost images from ref. 1.

While the approach by Asente improves on the folding issue, his solution is not robust in multiple situations. Figure 4.8 illustrates folding issues impossible to resolve with his method. This conclusion was also noted by Astente (Section 4 in ref. 1): 'While [my] technique improves folding in many cases, [I] unfortunately cannot completely eliminate folding because there are cases where [my] method cannot be applied...Some completely new adjustment technique is needed when [my] method fails, either because it cannot find a centre of crossing or because the ribs are not in order. [I] believe that a method based upon preventing the mapped artwork from crossing the medial axis of the area inside the curve could give good results, but [I] have not yet implemented it.' In contrast to the method of Asente, I provide an analysis in Section 4.2 which results in several solutions that are guaranteed to avoid folding (see Figure 4.8 for acceptable solutions where Asente's method fails). Note that I have found the medial axis, or similar constructions, pivotal in achieving such robust solutions.

4.2 Defining the coordinates related to extent

In this section, I present two solutions for defining the coordinates of $P_{i,m}$. As hinted above, these solutions build on mathematical tools like the medial axis. These tools are now described.

4.2.1 Relevant mathematical tools

In the following, I describe the medial axis, the distance transform, and Voronoi partitioning.

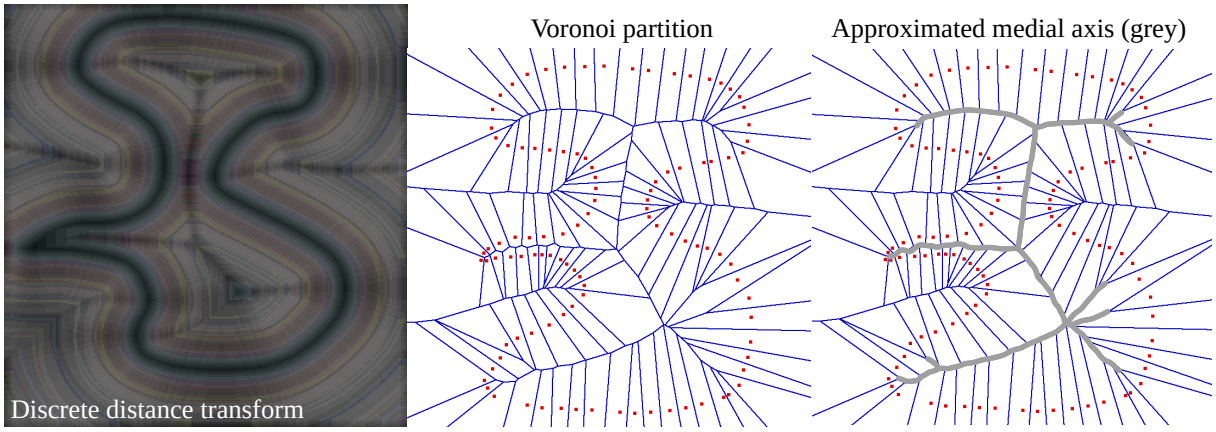


Figure 4.9: Visualisations of the mathematical tools employed in this section. Left: visualisation of the discrete distance transform. Original edge pixels are black. Middle: control points (in red) of a closed B-spline curve define the Voronoi partition. Right: edges completely contained in the region bounded by the curve approximate the medial axis.

The medial axis (MA) of a set of curves is, informally speaking, the set of all points having more than one closest point to any curve (Figure 4.9(right)). Formally, the MA of a planar closed curve C, forming a bounded domain S, is the locus of the centres of circles that are tangent to C in two or more points, where all such circles are contained in S. This was originally referred to as the topological skeleton, introduced by Blum[6] (1967). In the setting of simple polygons, the MA is a tree with leaves that are defined at the vertices of the polygon, since the circle (used in the definition) collapses to a point (the vertex). The medial axis transform (MAT) is the medial axis augmented with the radius of the circle (that is, MAT is the set of circles in S that are tangent to C in two or more points).

The distance transform[88] (DT) is a map defining the distance for each point in a domain X to a subset Y∈X. If Y is a curve (i.e. Y=C) in the plane P, the DT defines a surface with a gradient, except at Y and at the MA, with a direction normal to Y and with a slope of 45 degrees with respect to P. The slope of the gradient is undefined at Y and is less or equal to 45 degrees at the MA.

The discrete distance transform, DDT (Figure 4.9(left)), of an (pixel) image is defined similarly. In the setting of B-spline curves, the curve pixels in the image are set to 0 and all other pixels are set to 1. With this configuration, the DDT of the image produces a new image of the same size as its original, where each pixel in this new image is given its distance (typically Euclidean distance) from the set of 0's in the original image.

A Voronoi partition of the point set Q_i is a partition of the domain X where every point in X in each Voronoi *cell* has the same point in Q_i as its closest point (Figure 4.9(middle)). Formally, the Voronoi partition is the tuple of cells R_i, where:

$$R_i = \{x \in X \mid d(x, Q_i) \leq d(x, Q_j) \text{ for all } j \neq i\},$$

where d is a distance function (typically Euclidean distance). Note that this definition is not restricted to point sets, but also supports any non-empty subset of X like curves or segments.

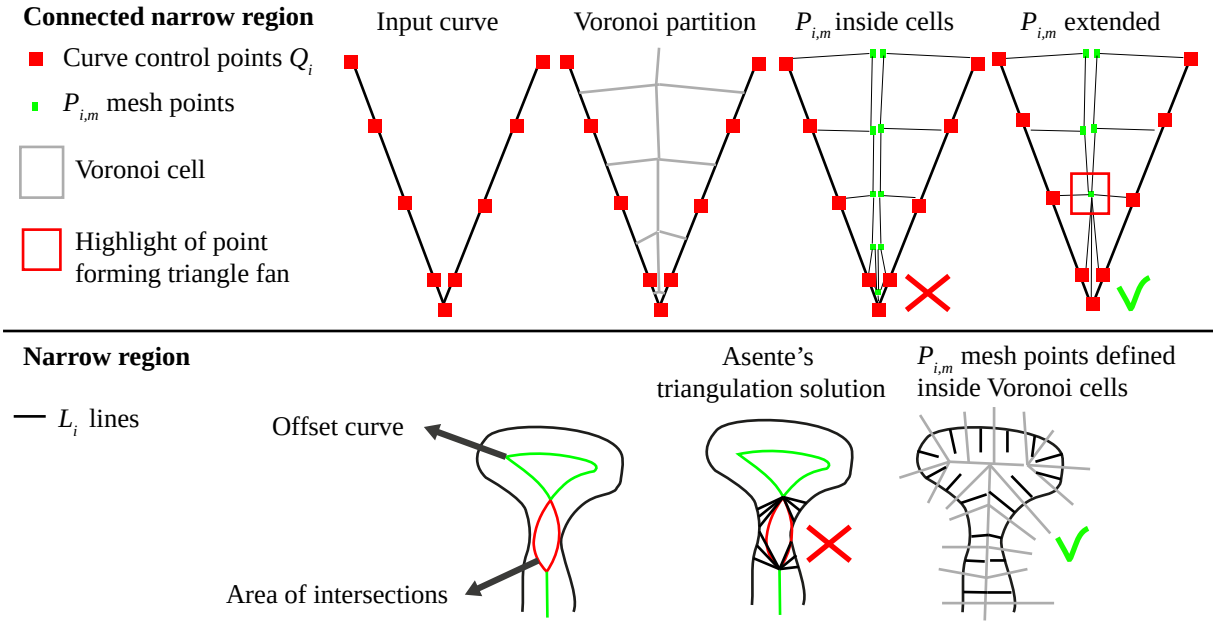


Figure 4.10: Several solutions at (connected) narrow regions. Top: placing mesh points $P_{i,m}$ only inside the Voronoi cell of a curve control point can make the extent of the L_i line unnecessarily short. Creating triangle fans, as shown in the rightmost example, is an acceptable solution. Bottom: creating triangle fans in narrow regions, as with Asente's solution, can stretch the control mesh and the resulting surface. In this configuration, placing mesh control points only inside related Voronoi cells *is* acceptable.

4.2.2 Observations

I start my explanation of the solutions with the following observation. From the definition of a Voronoi partition, each Voronoi cell for any given curve control point is unique with respect to cells of other control points. Furthermore, the cells do not overlap with each other. This means that one can place $P_{i,m}$ at any point inside the Voronoi cell of a Q_i and guarantee to satisfy Conditions 1 and 2.

This approach raises the following question: what should we do if the spatial extent of a Voronoi cell is unsatisfactorily small with respect to the input extent? Figure 4.10(top) illustrates this issue for a corner, with sufficiently close neighbouring curve control points to force the Voronoi cells unsatisfactorily small. Figure 4.10(top-right) further illustrates a proposed solution, similar to Asente's solution, that degenerate all nearby quads to triangles, forcing a satisfactory extent of $P_{i,m}$. This case therefore corresponds to a connected narrow region described in Section 4.1.1.

On the other hand, such a solution is unwanted at narrow regions since connecting adjacent neighbouring $P_{i,m}$ with triangles can stretch the control mesh and the resulting surface. This behaviour is unwanted because the direction of related effect is unnecessarily shifted away from the normal direction. These shifts do not appear with a Voronoi-based treatment. Thus, Voronoi-based treatment is preferable in narrow regions (Figure 4.10(bottom)).

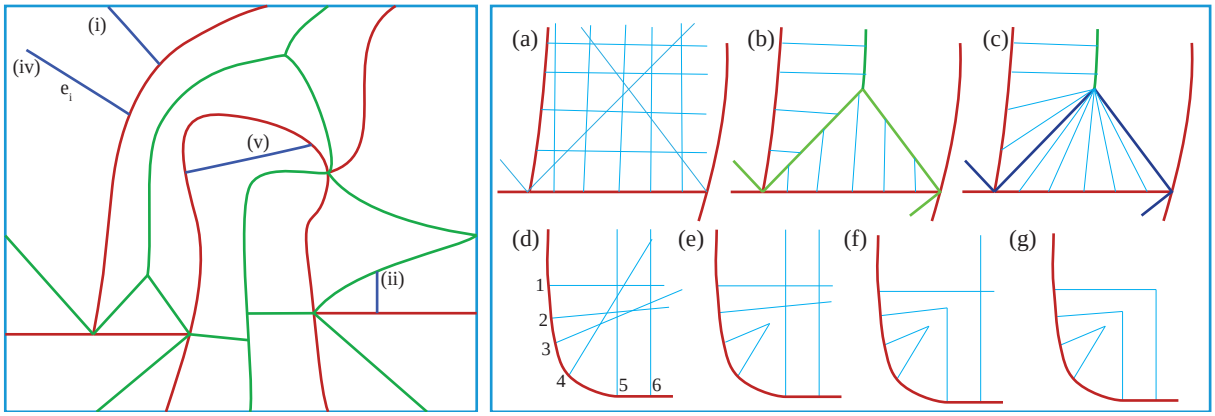


Figure 4.11: Overview of Solution 1. Left: input curves (red), Voronoi partition of the curves (green), and four termination cases (blue) for ray tracing out from control points. Right: (a) any arbitrary set of control points with their associated normal vectors (thin blue lines) may give a complex set of intersections. (b) The Voronoi partition, employed in Step 1, plays an important role by avoiding intersections to other curves. (c) Intersections related to rays $\hat{r}$ (thick blue lines) associated with corners will form triangular patches. (d–g) Remaining intersections are solved by iteratively pairing rays with low intersection count.

Based on these observations, I present two solutions to the placement of $P_{i,m}$. The first solution, Solution 1 (Section 4.2.3), assumes that curve control points relating to connected narrow regions are tagged as input. In Chapter 7, I present a method to identify such curve control points by thresholding the curvature of the curve. The second solution, Solution 2 (Section 4.2.4), relies on sensible values of the extent attributes. Consequently, Solution 2 is not as robust as the first solution in terms of larger extents, but is suitable for interactive applications where the user can alter the extents manually (Chapter 5).

4.2.3 Solution 1: meshing with tagged corners

The first solution I present resolves the connected narrow region vs. narrow region ambiguity by assuming that curve control points related to connected narrow regions are tagged as input. Such tagged points are defined as $\hat{Q}_i$. For simplicity, I refer to these points as *corners* (mathematically, they are not corners). Corners are used to define sensible coordinates for the related $\hat{P}_{i,m}$, which will give rise to triangular fans.

In this solution, I use a Voronoi partition of curve segments, instead of points. Figure 4.11(green edges) shows the Voronoi partition defined by the curve segments delimited by curve-end points and corners. In this partition, all points within a Voronoi cell have the same curve segment as their closest curve segment. Let $A = \{b_k\}$ be the MA approximated by this Voronoi partition, where the MA contains all edges b_k of the partition.

The solution is involved with two steps: (1) find the maximum possible extent for each curve control point Q_i by tracing rays inside Voronoi cells and (2) fix intersecting L_i lines inside each Voronoi cell. Additionally, an optional third thresholding step can be performed for robustness.

Step 1 finds the maximum possible extent c_i for each curve control point Q_i in the direction of N_i. These extents will be bounded by the MA. The maximum extent of a given Q_i is found by shooting a ray, r_i, from Q_i in the direction of N_i.

First, if the end points of an edge b_k coincides with a corner $\hat{Q}_i$ or a curve-end point, this edge is deleted from the current MA (that is, $A_i = A \backslash b_k$). The related ray is tagged as $\hat{r}_i$. Consequently, the two Voronoi cells related to the corner or the curve-end point are merged into a single cell.

There are five cases which terminate the ray r_i (Figure 4.11(left)):

(i) r_i strays outside the image domain (if defined),

(ii) r_i hits the current MA, A_i.

(iii) r_i hits any other barrier that constrains mesh extents in the image,

(iv) the length of r_i is equal to the extent attribute: $||r_i|| = e_i$,

(v) r_i hits the current curve segment.

The extent c_i is then set to either $||r_i||$ in the cases of (i)–(iv) or to $||r_i||/2$ in the case of (v).

Step 2 handles intersections between rays r_i inside a Voronoi cell. First, if such an intersection lies on a $\hat{r}_k$ found in Step 1, the corresponding quadrilateral patch degenerates to a triangular patch (Figure 4.11(c)):

$$P'_{i,m} = \begin{cases} Q_k + c_k N_k & \text{if } r_i \cap \hat{r}_k \neq \emptyset; \\ Q_i + c_i N_i & \text{otherwise.} \end{cases} \tag{4.4}$$

The remaining intersections are dealt with by truncating the rays (and thus the profile extents c_i) to the furthest point possible, without having any intersection to any other ray. Given a pair of intersecting rays $r' \cap r'' = I$, the number of intersections with other rays between Q' (Q'') and I is counted. The pair with the lowest intersection count is truncated to this intersection point. This is performed iteratively until there are no intersections left.

Figure 4.11(d–g) shows an example of iteratively dealing with such intersections. For example, the intersection count between r_1 and r_6 is 4. The lowest intersection count in this example is between r_3 and r_4 (0). These rays are therefore truncated to the intersection point between them. Two more iterations are then needed to resolve the remaining intersections.

Optional step 3. After ensuring that there are no self-intersections, neighbouring extents may be very different (Figure 4.12). Variations in neighbouring extents can be reduced so that they remain below a threshold T (evaluated in both directions of the list of c_i):

$$c_i = \begin{cases} c_{i-1} + T & \text{if } c_i - c_{i-1} > T; \\ c_i & \text{otherwise.} \end{cases} \tag{4.5}$$

Results

This solution was implemented as a framework for modelling surfaces in photographs and is discussed in Chapters 7 and 8. See Appendix A for details of my MATLAB implementation. Some results of the final rays produced with the three-step solution are shown in Figure 4.13.

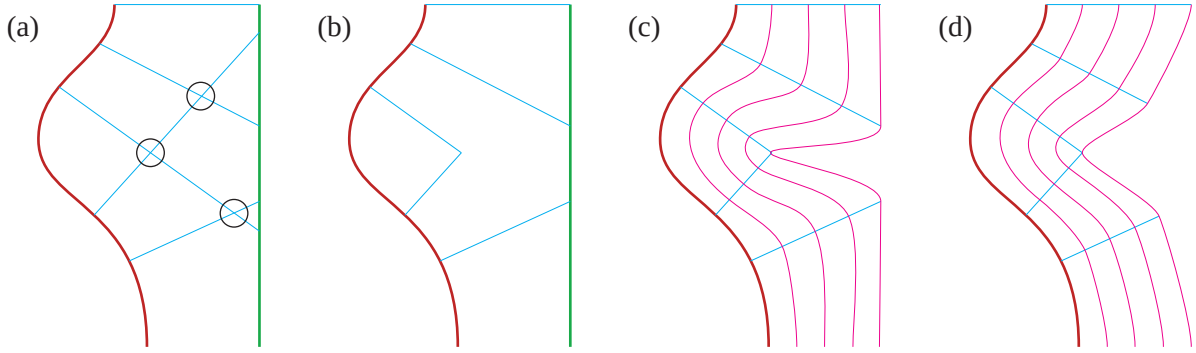


Figure 4.12: Large extents may lead to inappropriate profiles. In this example, intersections have been resolved (b), but neighbouring extents are too different for a smooth-looking profile in the image plane (c). The optional thresholding step improves such profiles (d).

In contrast to previous methods discussed in Section 4.1, my solution is guaranteed to satisfy Conditions 1 and 2 for any configuration of the input curves, as long as they do not intersect (in such cases, the curves can be trivially split). This is guaranteed by the use of the Voronoi partition, ensuring no overlaps between meshes of different curve segments, and by resolving intersections of L_i lines within a Voronoi cell. Finally, the behaviour of the solution can be controlled by the user by adjusting the threshold used in the third step, the extent attribute, and the labelling of corners.

4.2.4 Solution 2: avoiding intersections assuming sensible extents

The previous solution assumed knowledge of connected narrow regions and is robust to larger, or infinite, values of the extent attribute. In the following, I describe a single-step solution that satisfies Conditions 1 and 2. The extent attribute is the only parameter needed for this solution. The cost of such a 'simpler' solution is that it is not as robust with respect to large extents. Later in this section, I discuss optional parameters to achieve a robust solution akin to Solution 1. Solutions 1 and 2 are compared in Section 4.2.6.

The solution performs a single step of tracing along the DT surface (coordinates of a DT surface point: (x, y) position in image plane; z: distance to the closest curve point at (x, y); see Figure 4.14 for a visualisation). This surface has many useful properties:

- The gradient at any point, not positioned at an input curve or at the MA, is defined in the direction normal to the curve;
- The slope of this gradient is 45 degrees;
- C^0 creases in the surface relate to the input curves and the MA;
- The slope along the MA is less than or equal to 45 degrees;
- Stationary points (not related to curve points) and local extrema on the surface lie on the MAT.

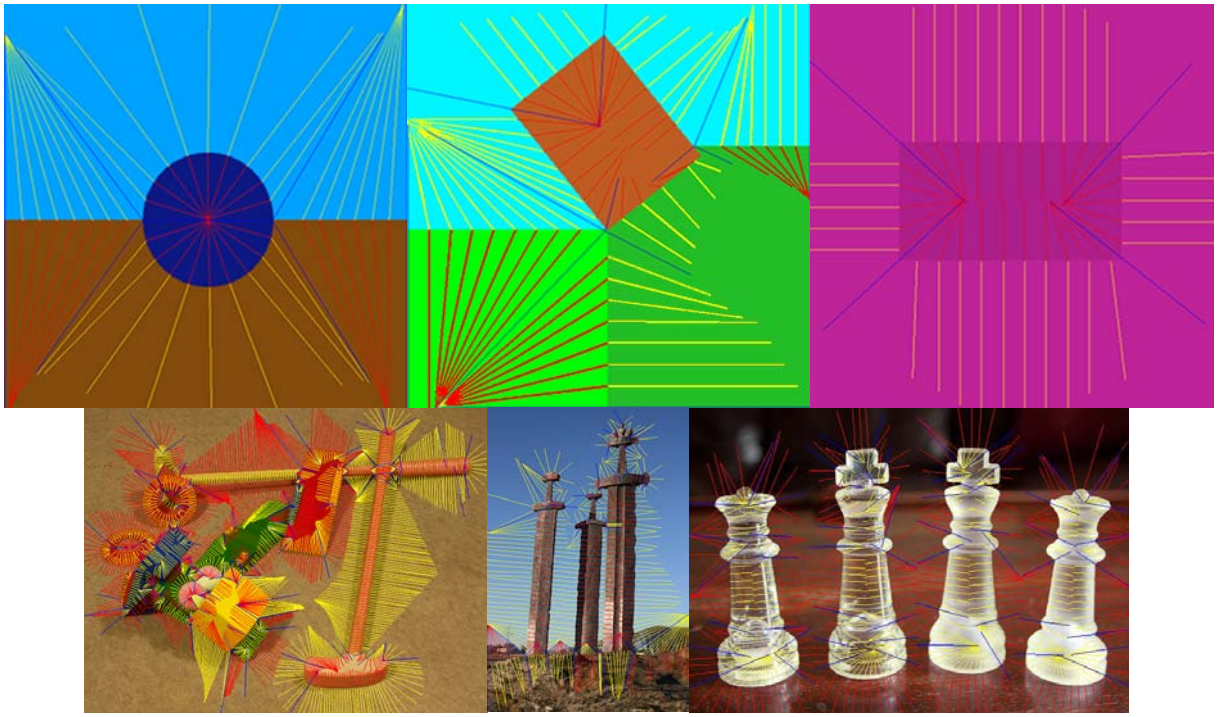


Figure 4.13: Rays shot from curve points extracted from images using the framework presented in Chapter 7. Rays related to corners and junctions, $\hat{r}_i$, are drawn with blue lines. The red and yellow lines relate to 'normal' rays r_i.

A conceptual solution is now described. Imagine a particle dropped on the DT surface for each Q_i along N_i. Such a particle is dropped infinitely close to the curve point associated with Q_i in the direction of N_i (that is, it is dropped on the correct side of the curve). These particles are then attracted by the gradient of the surface.

According to the properties of the DT surface, the particle behaves as follows. As time increases, they move 'upwards' along the direction of the gradient, towards the MA. When the MA is reached, they will continue to move along the MA until they reach a local extremum (that is, there is no more attraction and they remain stationary). In the limit, all particles will reach the end of the domain (e.g. the image border) or a local extremum. A particle also becomes stationary if its z coordinate equals the extent attribute. The coordinates of a $P_{i,m}$ are then defined as the image (x, y) position of its related stationary particle.

Informally, a particle goes hiking on the DT surface. It aims to reach the peak of the closest 'mountain' (local extremum) by finding the shortest route to the mountain ridge (the MA) and then follow this ridge to the peak. The shortest route to the ridge is found by following the direction with the steepest slope (the gradient). The particle also brings a device that calculates its elevation (z coordinate). The particle stops moving when this elevation reaches a pre-defined value (extent attribute), when it reaches a peak, or when it realises that there are no peaks (end of domain).

Figure 4.14 shows the behaviour of the DT tracing for three particles. The tracing follows the normal direction until the MA is reached. When the MA is reached, the tracing continues along the MA towards a local extremum. The tracing stops either when a DT value equals the given

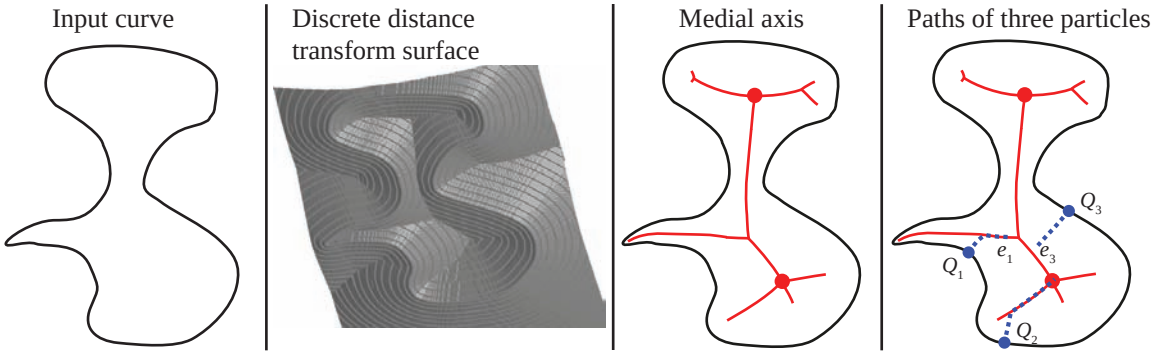


Figure 4.14: Solution 2 traces particles on the DT surface. The particles are attracted by the gradient of the surface. Consequently, they are implicitly traced along the MA.

extent (Q_1 and Q_3 in Figure 4.14) or when an extremum is reached (Q_2).

The control meshes are directly created from the control points $P_{i,m}$ defined from the particles. Quads related to Q_i and Q_{i+1} (that is: the quad defined by $P_{i,1}$, $P_{i+1,1}$, $P_{i+1,m}$, and $P_{i,m}$) are degenerated to triangles if their related particles coincide (that is: if $P_{i,m} = P_{i+1,m}$, they merge into a single control point).

Results and discussion

This solution was implemented as a framework for modelling surfaces in the setting of vector-based drawing (see Figure 4.15 for visualisations of created control meshes). See Appendix A for details on two C++ implementations (tracing the discrete DT and employing a Voronoi partition). The solution satisfies Conditions 1 and 2: Control meshes of neighbouring curves do not overlap (Condition 1) as the meshes are constrained by the MA. Surfaces do not fold (Condition 2) as the paths created by the particles only merge; they do not cross. However, larger extents may force the particle to travel along the MA to a point where artefacts can occur.

Figure 4.16 demonstrates that the proposed solution is not robust to larger extents: the mesh becomes stretched if the particle has travelled too far away on the MA. In the setting of drawing, it can be sensible to provide as much freedom to the user as possible and to not use any thresholds. If folds are created, due to different extents between curve control points (Figure 4.17), a warning message can be displayed (which can be clicked and the program can then automatically resolve the folding issue by decreasing the extents). Additionally, Figure 4.17 illustrates further issues that should be considered when implementing the solution in the commercial setting.

Thresholds should be employed in automatic applications to avoid the problems in Figure 4.17. There are two viable types of thresholds: limiting the extents globally, based on the resolution of the image or similar, or limiting the maximum angle from the normal vector N_i and the line L_i. It is not obvious how to define such thresholds in a general case due to the ambiguity described in Section 4.1.3. It therefore depends on the application and what kind of surfaces one wishes to define. Informally speaking, I do not see why angles between N_i and L_i should be above 20 degrees (or thereabouts): any effect is supposed to be modelled in the normal direction N_i out from a curve and not in the direction 20 degrees (or above) to N_i.

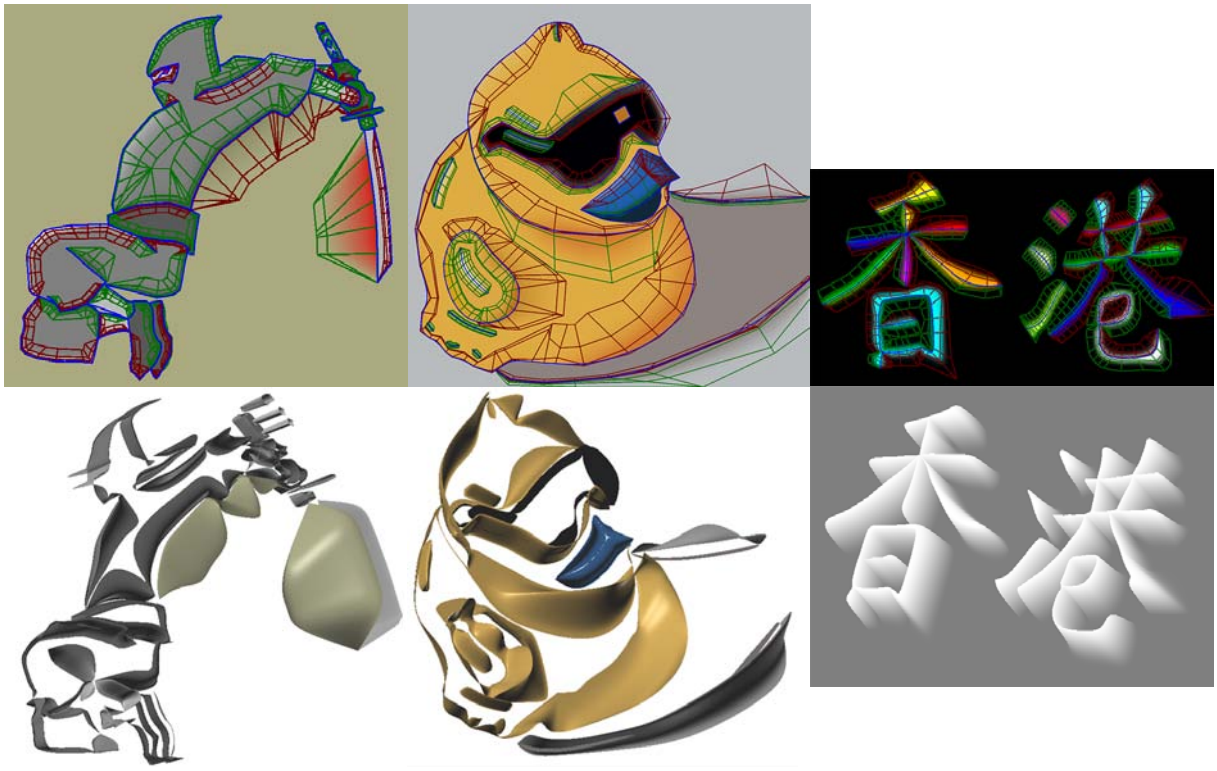


Figure 4.15: Solution 2 was implemented in the setting of vector-based drawing (Chapter 5). These visualisations show computed control meshes superimposed onto the resulting drawings (top) and visualisations of the resulting surfaces, which are hidden from the user (bottom).

4.2.5 Additional considerations

In the following, I discuss three special cases: junctions, high curvature points, and curve-end points.

Junctions

A junction defines a transition from one curve to another. Neighbouring curves may relate to the same object and the effect will typically continue between these curves. To this end, adjacent meshes with the same orientation in terms of the height attribute are merged (Figure 4.18). For two neighbouring curves the two overlapping control points, with attributes, A_1 and A_2, define this transition:

$$A = \begin{cases} (A_1 + A_2)/2 & \text{if } h_1 h_2 > 0, \\ 0 & \text{otherwise,} \end{cases}$$

where $A = \{e, h\}$ represents the attributes (extent and height) at the junction point. The control meshes are merged when height signs are the same. Otherwise, they remain separate. This will continue the effect if the two profiles have the same orientation and will make a smooth transition between them if they are different, as shown in Figure 4.18.

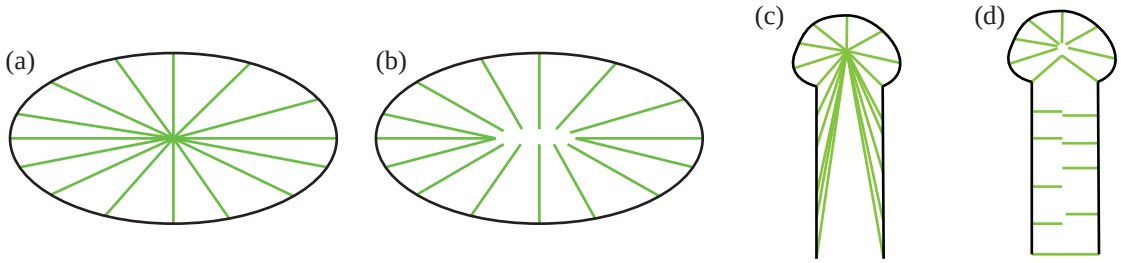


Figure 4.16: Influence of the extent parameter (large extents: a,c; low extents: b,d). Large extents can lead to stretched meshes (c).

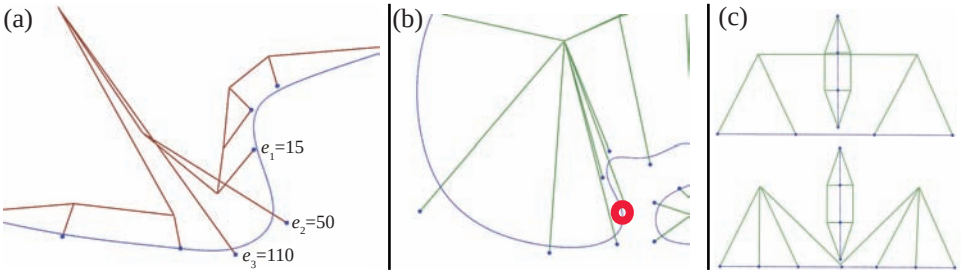


Figure 4.17: Cautionary matters: (a) Having different extents between neighbouring curve control points may result in folds. (b) Large extents may cause control meshes to cross input curves if they form a concave boundary. (c) Overlaps can occur if the solution is not sampled sufficiently.

High curvature mesh points

Having only a single set of $P_{i,m}$ control points out from high curvature curve points can be unsatisfactory since the offset curve is poorly approximated. To improve on this offset curve approximation, an additional set of control points can be added at curve points of high curvature (Figure 4.19):

$$P^*_{l-1,m} = P_{l,1} + \hat{N}d, \tag{4.6}$$

where P^* denotes the new control point that is added to the existing control mesh, l is the index of a high-curvature control point and d defines the extent of $P^*_{l-1,m}$. High-curvature mesh points can be detected by computing the angle between the control point's normal and the normal of the previous control point. If this angle is above a given threshold, the control point is a high-curvature point. A sensible threshold can be anything from 45 degrees and above. I use 90 degrees in my code.

The outer point, $P^*_{l-1,m}$, can be traced along a rotated 'normal' vector:

$$\hat{N}_l = \frac{1}{2}\left(P_{l-1,m} - P_{l,m}\right) - P_{l,1}.$$

The tracing, which will define d, is performed according to the given solution (1 or 2). The extent attribute should be defined as the extent of the related control point (that is: $e_l = e_{l-1}$).

Finally, I note that this issue stems from poor sampling of the solutions. An alternative solution

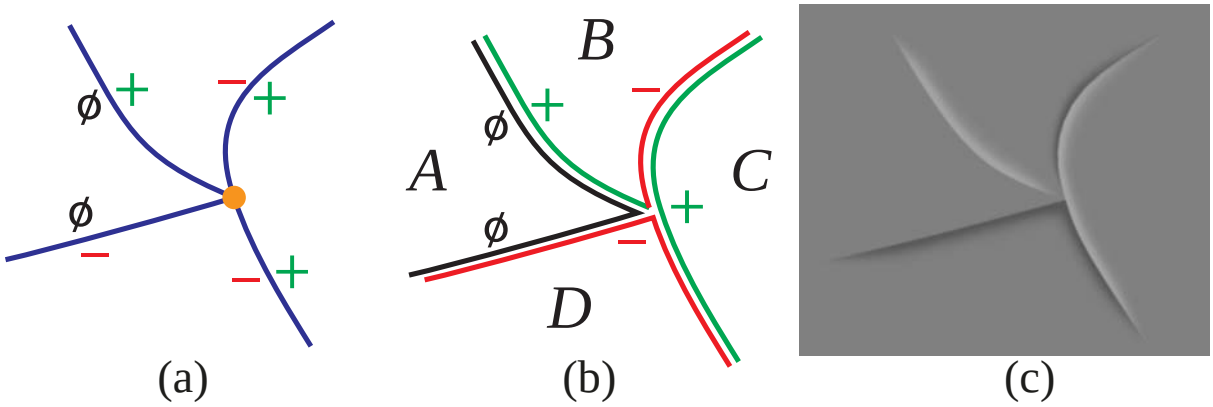


Figure 4.18: An example junction. (a) Four curves meet at a point. Each side of each curve has an orientation (negative or positive). (b) Two of the pairs of curves are merged (C and D). For visualisation, the curves have been separated slightly. (c) After merging. Regions C and D are each a single surface, B has two surfaces that blend smoothly into one another, A has no control meshes enabled.

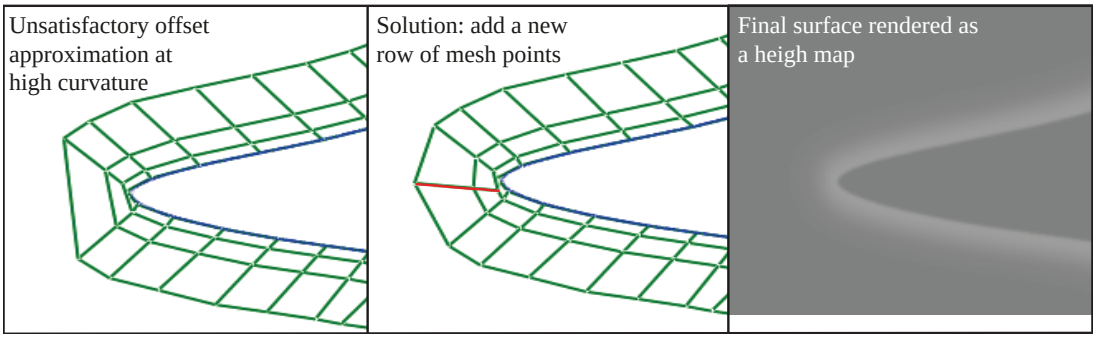


Figure 4.19: Improved mesh at a high-curvature point: An additional set of control points (red) are added at a high curvature control point so that the surface follows the offset curve more closely.

to ensure sufficient sampling in high-curvature regions is to sample such regions more densely with more curve control points.

Curve-end points

For open curves, special treatment at end points can be defined to make the effect of the curve smoothly fade out to the end of the curve. Alternatively, end points can be treated as any other curve control point, which makes a sharp transition if the effect is modelled in the image plane. Figure 4.20 illustrates the visual difference between these two approaches.

To define a smooth transition, the height and extent attributes can be set to zero. This creates triangular patches at the end of the curve, as shown in Figure 4.20(right).

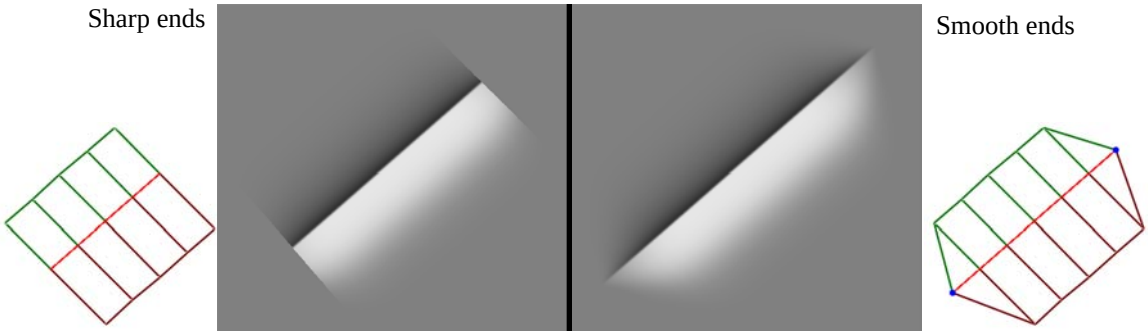


Figure 4.20: Two ways of treating the ends of a curve. Smooth transitions at the end of the curve can be achieved by zeroing the height and extent attributes at the end curve control points, resulting in triangular patches in the mesh.

4.2.6 Summary of solutions

I now summarise the differences between Solution 1 (S1) and Solution 2 (S2). S1 places $P_{i,m}$ inside Voronoi cells defined by curve segments delimited by corners (assumed as input) and curve-end points. Additionally, corners give rise to explicit modelling of triangular fans which is the desired behaviour at such points (Section 4.1.3). Intersections are resolved to ensure the satisfaction of Conditions 1 and 2. In contrast to S1, S2 does not require corners to be labelled. Instead, S2 approximates the offset curve defined by $P_{i,m}$ with the use of the DT surface. Conditions 1 and 2 are satisfied by tracing paths of particles attracted by the gradient of this surface. These paths only merge and do not intersect which guarantees the satisfaction of the two conditions. The ambiguity between connected narrow regions and narrow regions is implicitly dealt with by assuming that the given extent attribute is defined as the desired offset.

S1 is particularly well-suited in applications where there is knowledge of 'corners' and when correct modelling at such corners is more important than the satisfaction of the extent attribute. In Chapter 7, I demonstrate such an application: edges in photographs are enhanced automatically, typically 'as far out as possible' (that is, using infinite or very large extents). Corners can be extracted by measuring and thresholding the discrete curvature of the edges. It is important that such enhancement follows the shape of the edges rather than stretching the surfaces, which can happen with S2 at large extents (for example, see Figure 4.16).

On the other hand, S2 is useful for extent-centric applications, without the requirement of labelling corners. In Chapter 5, I demonstrate such an application: user-manipulation of shading out from curves for vector graphics. In this setting, the user manually defines the extent of the shading via a user interface. That is, the user is concerned with the extent attribute directly and not indirectly via corners. The system should therefore behave according to such input by approximating the given offset as best as possible. Such behaviour is best achieved with S2 because the particles will travel to the furthest point possible (if the extent is not reached), whilst ensuring that Conditions 1 and 2 are satisfied.

S1 and S2 can be employed in other applications not presented in this dissertation. Such applications might aim at different aspects than the applications I have investigated. For example, one might wish to employ S2 for an automatic application. That is, an application with automatic behaviour where it is deemed unnecessary to identify corners. To this end, several thresholds

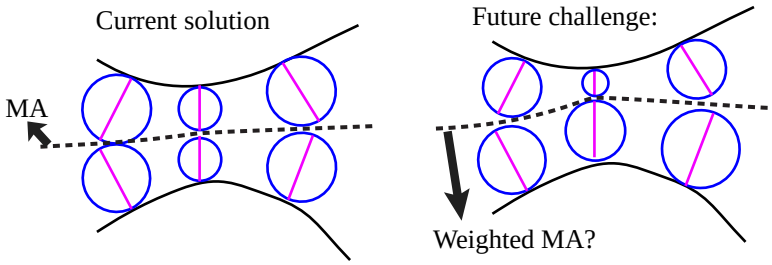


Figure 4.21: Future challenge: can the weighted distance transform be used to define a weighted MA? A weighted MA can enable curves to influence each other's meshes differently.

can be included to constrain the solution according to a target behaviour. Two viable thresholds are a global threshold on extent and a threshold on the deviation from normal vectors.

Similarly, the two solutions can be combined. For example, one might wish to control the behaviour at certain corners whilst approximating the given offset as best as possible elsewhere. Such behaviour can be achieved by initially employing S1. Then, instead of resolving intersections, which will limit the physical extent of the surfaces, S2 can be employed to curve points related to intersections to extend the surface to the given offset. Note that S1 and S2 have shown sufficient for their respective applications. I have therefore not experimented with such alternative configurations of the solutions.

In terms of computational efficiency, both solutions are efficient and suitable for interactive applications. Algorithms to evaluate the employed mathematical tools (DT, MA, and Voronoi partitioning) have all been proven efficient[88] ($O(n)$ complexity). On my laptop, with an Intel SU4100 1.3 GHz CPU, computing the discrete DT, using MATLAB, takes about 50 milliseconds on a 1 mega pixel image. Voronoi partitioning, using the Boost C++ library, takes about 5 milliseconds with 1000 points. Similar tracing operations are employed by S1 and S2, by tracing normal vectors (S1) and tracing paths of particles (S2). I have achieved better performance with S2, due to a better implementation (see Section 5.4 for timings; DT tracing ranges from 1 millisecond to 20 milliseconds depending on the number of curve control points).

4.2.7 Future work

The meshes' extent is restricted by the MA. However, one might wish to extend the meshes further, as illustrated in Figure 4.21. This would involve the extension of the set of attributes to include weights. An interesting avenue for future improvements of the solutions is to investigate whether a weighted distance transform[49] can be employed for this purpose.

4.3 Meshing the entire domain

Mesh generation methods do typically not rely on B-spline curves as input. Instead, they generate meshes from triangle meshes, point clouds, or iso-surfaces. In this section, I briefly summarise state-of-the-art quad generation methods. The types of surfaces produced, including

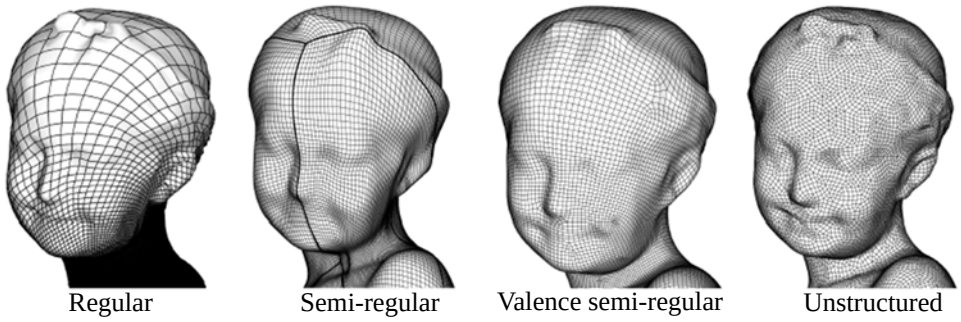


Figure 4.22: Classes of quad meshes. Image adopted from ref. 7.

frameworks to manipulate those surfaces, are usually intended for applications such as 3D character animation, industrial surface modelling, and the finite element method. This discussion is based on the review in ref. 7.

The purpose of this summary is to provide sufficient background to discuss the potential of employing quad meshing methods in the setting of vector graphics. While previous approaches in vector graphics have employed similar meshes for intermediate representations (for example, to employ the finite element method to render diffusion curves[10]), I am not aware of approaches aiming to produce general meshes that can be directly manipulated by users. In the setting of gradient mesh manipulation, previous methods have been proposed to automatically generate meshes from images[95;110;53]. However, such meshes must be defined with rectangular topology, which can limit the usability of the framework to certain applications (this limitation is discussed in detail in Chapter 6).

A solution with support for other types of control meshes, in particular quad meshes, can provide many advantages for manipulation of vector graphics. In general, quad meshes can be preferred because they support better alignment with the underlying shapes. Most geometry has two dominant local directions, typically associated with directions of principal curvature. While quads can be aligned with these two directions, triangle meshes, the popular alternative to quad meshes, give rise to an additional arbitrary direction.

In the remainder of this section, I present a brief summary of state-of-the-art quad-mesh generation methods (Section 4.3.1) and discuss the potential of employing such methods in the setting of vector graphics (Section 4.3.2).

4.3.1 Overview of state-of-the-art quad meshing

Quad meshes can be classified into several classes based on the degree of regularity (Figure 4.22). A quad mesh is:

- *regular* if it can be globally mapped to a rectangular subset of a square tiling;
- *semi-regular* if it consists of regular sub-meshes defined side by side;
- *valency semi-regular* if most of its vertices have valency 4;
- *unstructured* if a large fraction of its vertices are irregular.

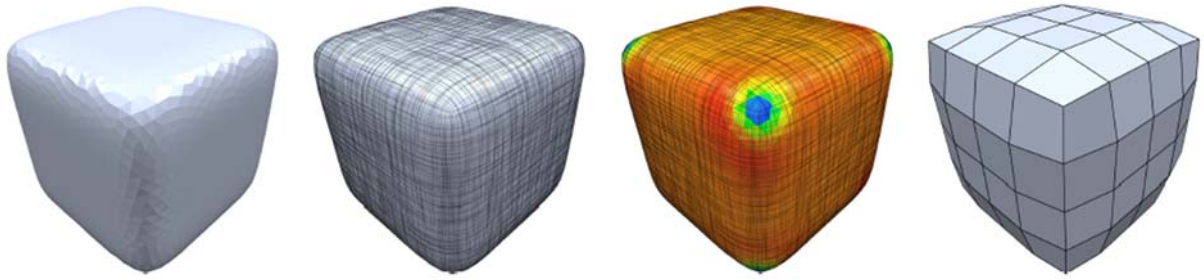


Figure 4.23: In cross-field-based methods, it can be instructive to separate the angular components (orientation) and length components (size) of the cross field. Image from ref. 7.

Additionally, a quad mesh is *pure* if it only consists of quads, is *dominant* if it mostly consists of quads, and is *mixed* if the number of quads is roughly equal to faces of valency other than four.

There is a natural relation between quad meshes and cross fields. A cross field is an assignment of a pair of directions to each surface point (ref. 7, Section 1.1) and may be regarded as a quad mesh with infinitely small quads with edges aligned with cross field directions. In the context of quad meshes, cross field singularities correspond to the irregular vertices of quad meshes. Finally, integral lines starting at singularities are defined as *separatrices*. Separatrices define a natural partition of the surface if all separatrices also end at singularities.

A naïve approach to convert a mesh to a pure quad mesh is to perform Catmull-Clark subdivision. Two major drawbacks of this approach are that it increases the number of elements and that it can introduce an unnecessary large number of irregular vertices. Consequently, Catmull-Clark subdivision is typically a practical option only if the initial mesh is quad-dominant and an increase in density is acceptable.

More advanced methods on triangle-to-quad mesh conversion combine a sequence of local operations on connectivity to produce meshes that are as regular as possible. A main operation is to fuse two original triangles into one quad. The choice of triangles is important to achieve high quality. For example, a current state-of-the-art method employs the blossom algorithm to optimally compute the best possible shape of the quads[85]. In general, one can only expect such methods to produce unstructured meshes[7].

My solutions presented in Section 4.2 are similar to certain meshing techniques. Specifically, direct meshing[i] can be performed by first defining patches at single offsets along the boundary and then advance out from the boundary. Due to this advancing behaviour, such methods are typically referred to as *advancing front methods*. Some of these methods employ the same mathematical tools used in Section 4.2: the MA has been used for triangular mesh[67], quad mesh[99;82], and hex-dominant mesh[89] generation and Voronoi partitioning has been used to create mixed triangle and quad meshes[94]. Note that all of these methods are involved and rely on separate treatment at different topologies, resulting in a tree of solutions for a given set of pre-defined topological cases. Thus, my problem, as described in Section 4.1.3, is not as challenging as general meshing because I produce a set of disjoint meshes. These disjoint meshes are not merged

[i]Direct meshing: methods that do not require an initial, typically triangular, mesh.

and are treated separately, without any disadvantages, in the applications I have investigated.

To achieve explicit control over local properties of quad elements in the mesh, constructions based on cross fields can be employed. The advantage of using cross fields is that the problem can be decomposed into a set of simpler sub-problems. This means that different aspects of the quad mesh can be optimised individually, which can be more tractable than optimising all aspects simultaneously. Note that the representation of a cross field can be split into an orientation field (angular components) and a sizing field (length components). See Figure 4.23 for a visualisation of these fields. A typical field-guided method consists of three steps (e.g., see ref. 8): orientation field generation, sizing field computation, and quad mesh synthesis.

In summary, approaches to meshing ranges from naïve (Catmull-Clark), to local optimisation based on shape (triangle-to-quad), to parametrisation methods via global optimisation (for example, via cross fields). Note that this list of methods is not exhaustive. For a broader view on quad-mesh generation methods, see ref. 7.

4.3.2 Future work

The above discussion is based upon meshing of 3D objects. A question that arises is whether these methods can be applied for applications within vector graphics. For example, given a set of freeform curves defined in the 2D image plane, can meshes be automatically created with sufficient quality to be manipulated by users or by software?

The summary above indicates that the specific application should dictate the choice of method and the analysis of its applicability. I will therefore not comment on whether current meshing approaches are suitable for vector graphics in general. However, let us assume the specific application of gradient mesh editing. That is: what type of mesh would a user prefer to manipulate colours with, given that this mesh is extracted from a set of boundary and constraint curves?

One can imagine that high-quality meshes (for example, semi-regular quad meshes) should be required in this setting. Additionally, the mesh should be relatively sparse so that users do not have to make too many edits to achieve a desired effect. Finally, we should assume the input curves to be of any shape, giving rise to meshes of a wide range of different topologies. Given these constraints, one can imagine that a more advanced method, such as a field-guided method, would be needed. This is because there are strict conditions on both quality and on topology, making other methods less attractive.

A significant problem that would arise is how to guide the field (for example, a cross field). In the setting of 3D objects, using the curvature of the underlying surface is a sensible option because we would get a denser mesh around high-curvature surface areas. However, in the setting of 2D, with only curves as input, there is no such surface curvature available.

Several previous methods employ cross and vector fields in the 2D setting, all of which use constraints on the boundary of their domain to create the field. Vector fields defined by the method in ref. 74 have been used to transform vector graphics to *calligrams*[63]. A calligram is a poem or phrase that is arranged in a way so that it creates a visual image. The boundary conditions dictate whether text is aligned with the boundary or not. Cross fields have also been employed to create 2D urban layouts from a polygonal domain associated with constraints like roads,

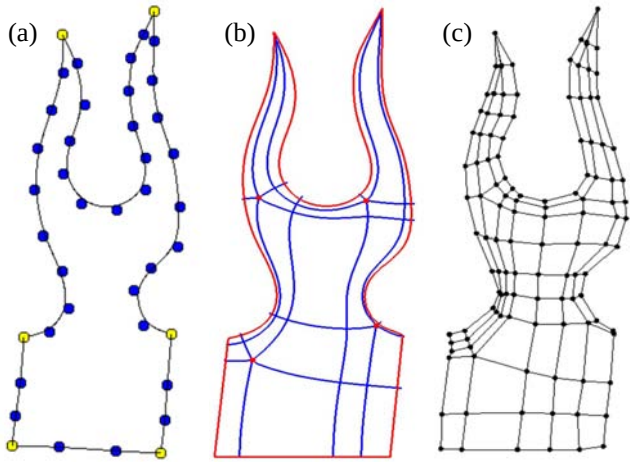


Figure 4.24: Meshing a challenging domain bounded by a set of curves. (a) Input curves; junctions are illustrated with yellow disks. (b) Partition of the domain with separatrices defined by a cross field[8]. (c) Mesh manually defined, guided by the separatrix partition.

lakes, parks, and buildings[111]. Finally, temporal painterly rendering has been demonstrated with vector fields[14], where the user can adjust the temporal 'flow' of the image by specifying field properties such as singularities.

Figure 4.24(a–c) shows a typical input in the setting of drawing and a type of result one can expect from a quad meshing method. The intermediate image (b) represents a partition of the domain using sepatrices of a cross field computed with the method in ref. 8. The constraints for the cross field generation method were defined as the tangent directions along the input curves. A quad mesh was manually created, with ShetchUp, using the sepatrix partition as a guide. This result and the wide range of results demonstrated for the 2D setting, as described above, indicate that such a method can be sufficiently robust for mesh generation for gradient meshes. An interesting avenue for future research is to validate (or invalidate) this claim to a greater extent.

4.4 List of contributions

– I have presented a novel framework for creating control meshes from curves associated with attributes to control the shape of the resulting surfaces.

– In contrast to related work, my solutions are robust to the topology of the input curves.

Chapter 5

Curve-based surface modelling for vector graphics

This chapter presents research described in the following paper:

Shading curves: vector-based drawing with explicit gradient control

---

Solution 2 presented in Chapter 4 is in this chapter applied in the setting of vector graphics creation. I propose a new primitive for vector graphics, which provides an alternative type of control of shading and lighting effects than previous primitives proposed in the literature (diffusion curves) and methods implemented in commercial software.

My motivations are a desire to develop concepts that are easy to understand and a desire to develop methods that allow the creation of vector graphics to resemble traditional drawing techniques as close as possible. To adapt current vector graphics primitives towards these goals, I argue that the artist should to be able to control the propagation of colours across the image domain. Mathematically speaking, the artist should be able to control the gradient of the colour function.

The principal prior work, against which I compare my new method, is diffusion curves (DCs). The primitive proposed in this chapter has more explicit control over the colour gradient compared with diffusion curves. Second-order DCs with gradient control employ additional 'special' curves that add pre-defined gradient penalties along a given direction. In contrast, the proposed primitive embeds gradient control to the colour curves directly.

Figure 5.1 illustrates the proposed primitive for manipulation of colour gradients. In addition to the colour attribute associated with the input curves, as employed by DCs, I propose to also associate *shading profiles*. A shading profile is defined as a curve that represents how the colour at the related point in the input curve is propagated in the perpendicular direction of the curve. Due to these shading profiles, I refer to my primitive as a *shading curve*.

The problem of rendering shading curves is solved by modelling and rendering subdivision surfaces. Figure 5.1(top-middle) visualises an intermediate image that is extracted from the

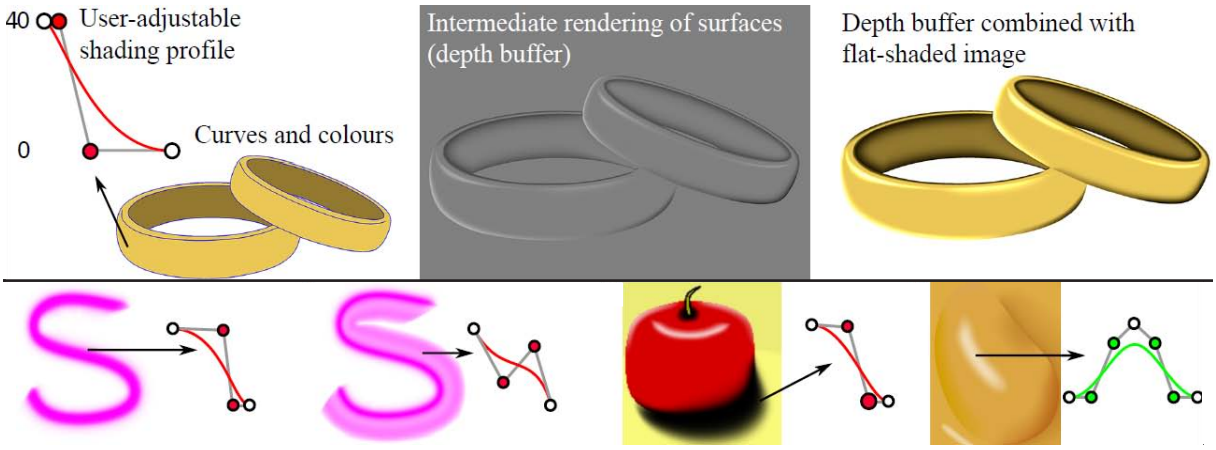


Figure 5.1: Drawing with shading curves. Top, left: input curves, colours, and a shading profile at the given curve location. In this example, the shading profile represents difference in luminance to the colour of the related region. That is, the luminance at the curve in the resulting image is the luminance of the yellow-ish colour plus 40. Top, centre: intermediate image extracted from the depth buffer of the rendered surfaces. The shapes of the surfaces are dictated by the shading profiles. Top, right: combining the flat colour image with the luminance modification produces the final result. Bottom: influence of the shading profile on several shading curves.

depth buffer of the rendered surface. This depth image is used to alter the original flat-shaded colour image in Figure 5.1(top-left) to produce the final result.

The main challenge of this problem is to construct control meshes that define the subdivision surfaces that are projected to the image plane. This problem is described in detail in Chapter 4. The other sub-problems faced when rendering the proposed primitive, such as surface rendering and interactive performance, are all well-studied problems. Standard solutions to these problems have shown to be sufficient for this application. Thus, I do not present any new technical solutions in this chapter. Instead, I present a framework to create, manipulate, and render shading curves (Section 5.3). Additionally, I compare the proposed primitive and the proposed framework to related methods in vector graphics (Sections 5.2 and 5.5).

The shading curve primitive has been inspired by traditional methods for depicting light and shade. This inspiration originates from the fact that previous primitives do not closely resemble traditional methods. I present a brief background of light and shade in art and demonstrate a common shading method in traditional drawing in Section 5.1. Additionally, shading is a well-studied problem in computer graphics, with well-defined methods in 3D environments. In the setting of 2D, however, automatic shading of images is not trivial, thus justifying the use of manual methods (Section 5.2).

5.1 Light and shade in art

Depicting light and shade is an important aspect of the visual arts. Sophisticated techniques have consequently been developed over the history of modern civilisation. In this section, I

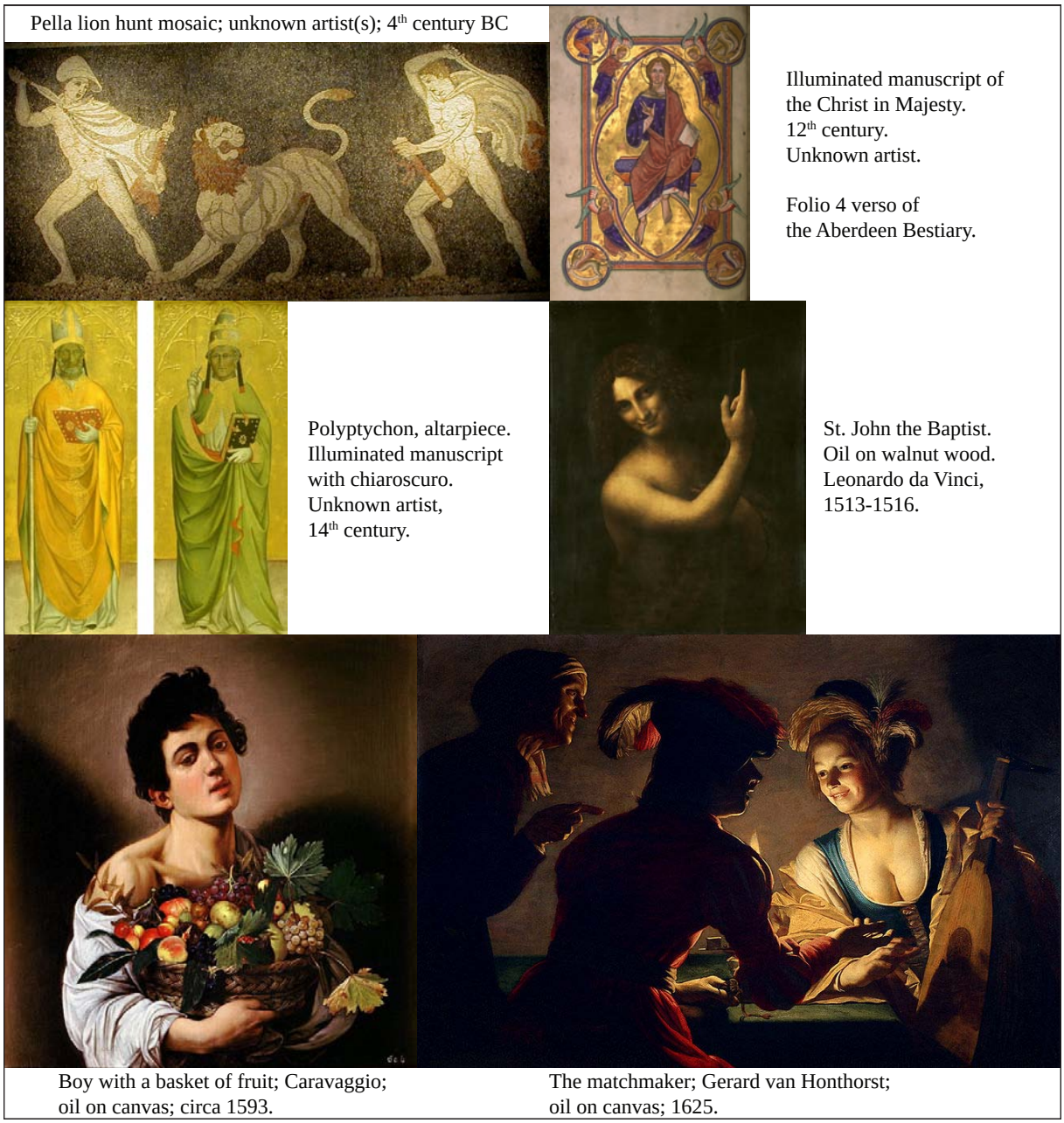


Figure 5.2: The development of chiaroscuro, depiction of light and shade, in the Western history of art. Shading techniques first arose in Ancient Greece with skiagraphia. This technique developed over the centuries through the Middle Ages with illuminated manuscripts. In the late Middle Ages, Italian artists started to employ ideas from Ancient Greece in illuminated manuscripts. Such shading became the start of chiaroscuro, a term coined by Cennino Cennini[13] in a drawing handbook. Chiaroscuro has developed over the centuries and today refers to the use of strong contrast between light and dark. It was during the Renaissance an important aspect of depiction of volume (e.g. St. John the Baptist) and later in the Baroque period for depicting emotional states (e.g. boy with a basket of fruit). The candlelight scene (e.g. the matchmaker) is in particular efficient with the use of chiaroscuro.

briefly present the history of shading in art and I discuss how such techniques have inspired the shading curve. Note that the focus of this discussion lies within the boundary of Western art and is in particular focused on the development of *chiaroscuro*. Figure 5.2 visually summarises this section. See refs. 108;17 for a broader view on this topic.

Formal techniques for light and shade can be traced back to Ancient Greece. The first shading technique, skiagraphia ('shadow-painting'), has been attributed to Apollodorus, an Ancient Greek painter of the 5th century BC. Unfortunately, none of his works remain. His technique, however, was highly influential, a conclusion which is drawn from ancient Roman historians such as Pliny the Elder and Plutarch. Early works of this technique are visible in 4th century BC mosaics of Pella, Macedonia. Compared to modern techniques, skiagraphia is similar to crosshatching; that is, variation in density corresponds to variation in tone.

In the early 15th century, the Italian painter Cennino Cennini[13] coined the term chiaroscuro as a technique for drawing light and shade (Italian: light–dark). At the time, chiaroscuro drew inspiration from skiagraphia and illuminated manuscripts. Illuminated manuscripts, the dominant form of art during the Middle Ages, typically refer to illustrated manuscripts decorated with layers of gold or silver. In the late Middle Ages, techniques from Ancient Greece, such as skiagraphia, were then incorporated to give these manuscripts relief and volume.

Chiaroscuro has, over the centuries, been developed into a sophisticated technique of depicting volume and mood. It was a major factor in the development of more realistic depictions during the Renaissance, with influential artists such as Leonardo da Vinci pushing the development of chiaroscuro. The evolution was further continued throughout the Baroque period. Rembrandt and Caravaggio are in particular well known for dramatic use of lighting, creating a sense of the human state, both physical and emotional. Such dramatic use of chiaroscuro is also referred to as tenebrism (Italian–tenebroso: dark/gloomy).

Chiaroscuro is today typically referred to as the use of strong contrast between light and dark tones. For example, the common scene lit by candlelight in the Baroque period is a typical chiaroscuro lesson in painting and photography. In film, this technique was adopted in Stanley Kubrick's *Barry Lyndon*, where numerous sequences were shot without electric lighting. Such careful use of lighting has shown to be one of the most powerful artistic tool for depicting both the 3D environment and mood.

Drawing chiaroscuro

Chiaroscuro drawing involves depicting tonal changes across the image/painting. This typically includes outlining a few areas of uniform tone and then shading off the boundaries. Based on these shading 'hints', the viewer will imagine the object's 3D shape and the direction of light. According to the professional[17], 'good chiaroscuro defines shadows and shade as decisive, quite sharp but not hard, with a certain transparency and able to suggest [the three-dimensional space]'.

Figure 5.3(top) illustrates a typical lesson in chiaroscuro, where drawing shade consists of three stages[17]:

1. outline areas of uniform tone,

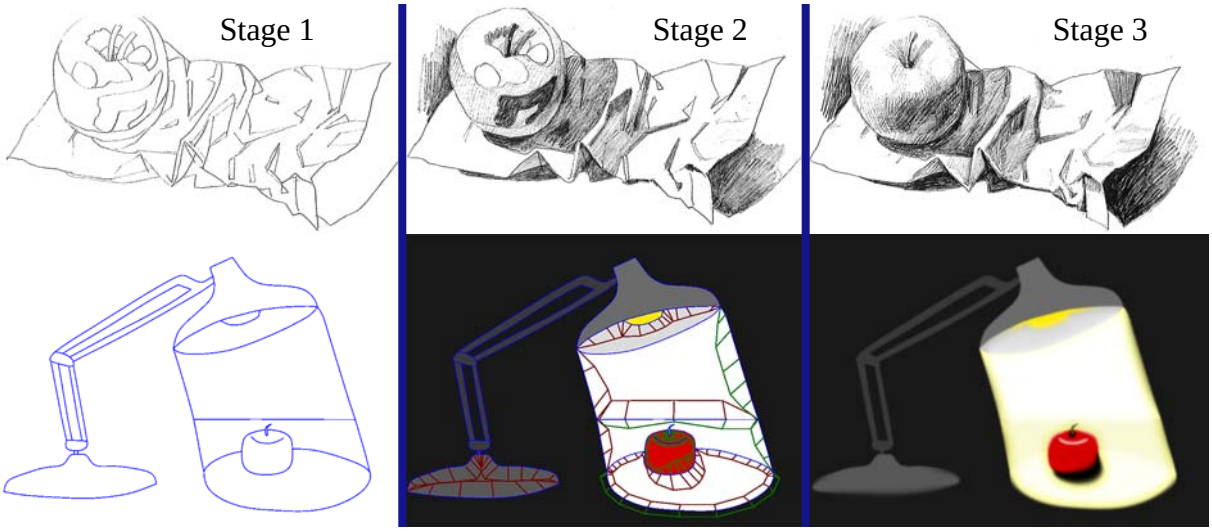


Figure 5.3: Drawing chiaroscuro with a traditional method (top; images from ref. 17) and with the shading curve (bottom). Such drawing can be performed in three stages. Stage 1: outline object boundaries and main tonal areas. Stage 2: fill in each area with a tone or colour, and with the shading curve: define the extent of the propagation of colours (green and red quads and triangles). Stage 3: smooth out the colours or tones, and with the shading curve: adjust shading profiles both globally and locally.

2. fill in each individual area with constant colour or tone,

3. smooth out the colours or tones to gradually blend with other areas.

The shading curve is inspired from such chiaroscuro drawing (Figure 5.3(bottom)). I suggest the following approach to drawing with shading curves:

1. Draw areas of constant tone with curves, including object boundaries and main tonal areas.

2. Fill in each individual area with constant colour and select the influence of that colour to adjacent areas.

3. Smooth out the colours with shading profiles. Refine colours and tones locally by adjusting the attributes of the shading curves.

In addition to the shading curve primitive, I have found it helpful to treat boundary curves and interior curves differently. Thus, I have implemented two types of shading curves with different behaviour. The first type of curve, the *boundary curve*, defines a sharp transition across the curve. It is therefore suitable for object boundaries since they are typically defined as hard edges. By contrast, the second type of curve, the *slope curve*, defines a smooth transition across the curve. Slope curves are useful for shading highlights and transitions within the object interior, such as specular highlights and cast shadows from other objects.

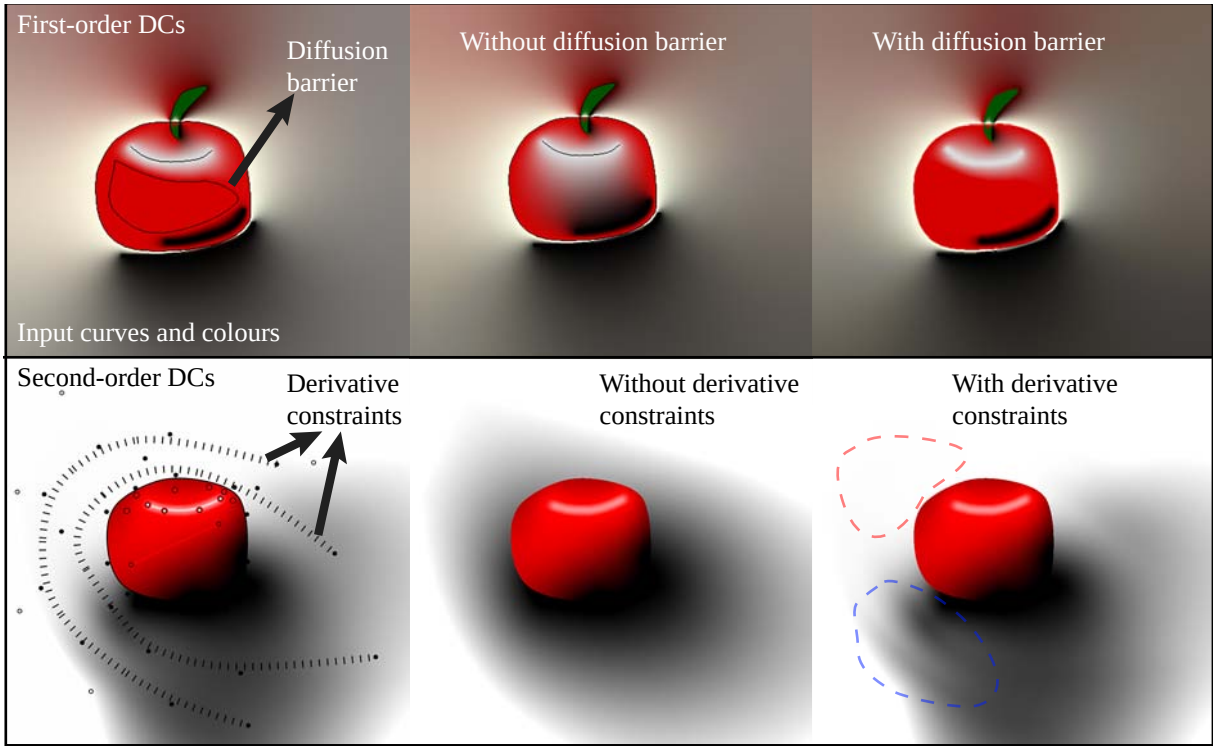


Figure 5.4: Influence of DCs on colour propagation. The results with first-order DCs and second-order DCs were drawn with prototype software provided in ref. 73 and ref. 31, respectively. The dashed regions (bottom right) illustrates that second-order DCs' derivative constraints give rise to different types of behaviour depending on the local configuration of value constraints.

Comparison with diffusion curves

A motivation of the shading curve and the shading profile is that DCs are not well suited for chiaroscuro-style drawing. Figure 5.4 demonstrates how the colour gradient can be manipulated with DCs.

First-order DCs are rasterised via Laplacian diffusion which does not natively support manipulation of the colour gradient. However, several tricks have been suggested to achieve some degree of manipulation: Gaussian blurring can be used to smooth the sharp transition across the curve[73], weights for rational harmonic functions can be utilised to alter the relative influence of a boundary condition[4], and diffusion barriers have been proposed to limit the extent the colour propagation[4] (Figure 5.4(top)).

In contrast to first-order DCs, second-order DCs *do* support manipulation of the colour gradient. Recall from Section 2.2.2 that 'special' curves were introduced to constrain the first derivative to zero either along the curves or across the curves (Figure 5.4). While the zero-derivative constraints provide a way to manipulate the gradient of the colour function, it does not achieve *explicit* gradient control. We have found these constraints to have two types of behaviours, as demonstrated in Figure 5.4(bottom). The first type of behaviour *suppresses* the propagation of a colour. This can be seen in the top portion of the example (red-dashed region in the figure), where the black colour is suppressed. This is because there are no black constraints normal

to the derivative curve in those regions. The colour of the (white) point constraint normal to the curve is therefore propagated to the curve. In the bottom portion of the image, however, both black and white constraints are located normal to the curve. In this scenario (blue-dashed region), the derivative curves give rise to a wavy-shaped colour profile (with no suppression). These derivative constraints can give rise to unwanted behaviour because their influence on the colour gradient depends on the spatial configuration of the neighbouring colour constraints. By contrast, our approach lets the user manipulate the colour gradient out from curve locations more explicitly without being influenced by neighbouring curves, as illustrated in Figure 5.1.

5.2 Light and shade in 2D computer graphics

As previously mentioned, shading is an instrumental aspect of the visual arts. As a consequence, sophisticated shading techniques have been developed for computer graphics. In this section, I present a brief introduction to local shading in computer graphics (Section 5.2.1). The main two messages from this discussion are that shading profiles are not fixed functions, as implicitly assumed by first-order DCs, and that being able to control such shading profiles can aid artists when drawing shade and light. Additionally, related methods aim to automatically apply shading to 2D images. While such methods can give rise to helpful tools, they rely on certain priors that make them unsuitable for general drawing and design (Section 5.2.2).

5.2.1 Mathematics of local shading

The mathematics of shading relates to the transportation of light and how light interacts with objects in our (real or virtual) world. In computer graphics, multiple simplifications are carried out to make this simulation problem computationally feasible. For example, the rendering equation[45], used as a basis for many graphics rendering algorithms, represents a computationally-feasible approximation to the transportation of light in the real world.

In this discussion, we are interested in aspects of *local* shading. By local shading, I mean that the shading, or light, emitted from a surface point is approximated by taking external light sources and internal material properties of that surface point into account, but the interaction of light between objects and other types of media is disregarded. A *global* shading model *does* take the latter types of interaction into account. To discuss local shading in the setting of 2D drawing is sufficient because one cannot assume, in general, that the spatial relationships between drawn 'objects' are known. Such objects are only defined in the image by curves and brush strokes, outlining the object boundaries projected to the 2D canvas. Thus, the spatial relationships between objects are not available. See ref. 97 for more discussion on this assumption.

A popular local shading model, often employed in interactive graphics rendering, is the Phong reflection model[78]. In addition to only considering local illumination, Phong also assumes the object to not be a light source, the light sources to be defined as point lights, and the material of an object to be constant. The Phong reflection model can be written as:

$$L(P, \boldsymbol{d}) = f_a L_a + \sum_{m \in \Theta} \left(f_d \left(\boldsymbol{d}_m \cdot \boldsymbol{n} \right) L_{m,d} + f_s \left(\boldsymbol{r}_m \cdot \boldsymbol{d} \right)^{\alpha} L_{m,s} \right); \tag{5.1}$$

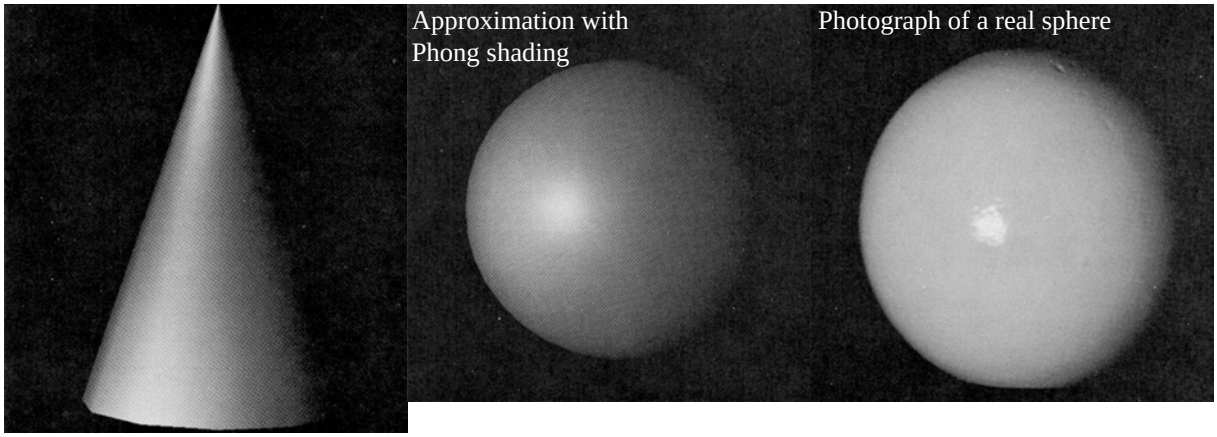


Figure 5.5: Shading of a cone (left) and a sphere (middle) rendered with the Phong reflection model. The comparison with the photograph of a real sphere (right) demonstrates that the model can be visually accurate in the setting of local illumination. Images from ref. 78.

where $L(P, \boldsymbol{d})$ is the light emitted from the surface at P in the direction $\boldsymbol{d}$. This reflection model defines shading as a combination of the ambient (subscript a), diffuse (subscript d), and specular (subscript s) shading components from P towards $\boldsymbol{d}$. The material properties f are defined as single scalars and are constant for each type of material. Similarly, the contribution from a light source m in the set of light sources Θ is defined as a single intensity or colour L_m. In relation to the light sources in the scene, the model is a combination of the perfect diffuse reflection, modulated by f_d and the incident angle of $\boldsymbol{d}_m$ to the unit normal vector $\boldsymbol{n}$, and the perfect specular reflection, modulated by f_s and the incident angle to the perfectly reflected vector $\boldsymbol{r}_m$. This vector $\boldsymbol{r}_m$ is the perfect reflection of $\boldsymbol{d}_m$ in relation to the normal vector $\boldsymbol{n}$. The ambient term lightens the objects in the scene and crudely compensates for global illumination effects. Finally, specular highlights are modelled with α, where smaller, more shiny-looking, highlights are defined with larger values of α.

Figure 5.5 demonstrates that the Phong reflection model is relatively accurate when illuminating a sphere. In general, the diffuse component accurately reproduces real-world Lambertian reflection of light. The specular component is a good approximation of specular reflection. More accurate models (for example, the model of Cook and Torrance[18]) can be used, but are unnecessary for this analysis since the rendering of shading curves does not require usage of such models.

According to Equation 5.1, the normal vector of a surface point P is the only intrinsic property of the surface that needs to be defined. While 3D coordinates (2D in image plane; 1D in depth) typically define the surface, disregarding the depth coordinate has shown to be sufficient for many types of shadings (see Section 5.2.2 for such methods). All other parameters: the direction to the observer, the ambient lighting term, material terms, definition of lights, and the specular highlight term, can all be defined by the artist in a drawing interface. Thus, to achieve Phong-like shading in the setting of 2D drawing, the normal vectors of the underlying surface must be estimated from the drawing.

Figure 5.6 shows shading of real objects with approximately constant diffuse-looking material; that is, according to Phong's assumption. In the head example, notice how the curvature of the

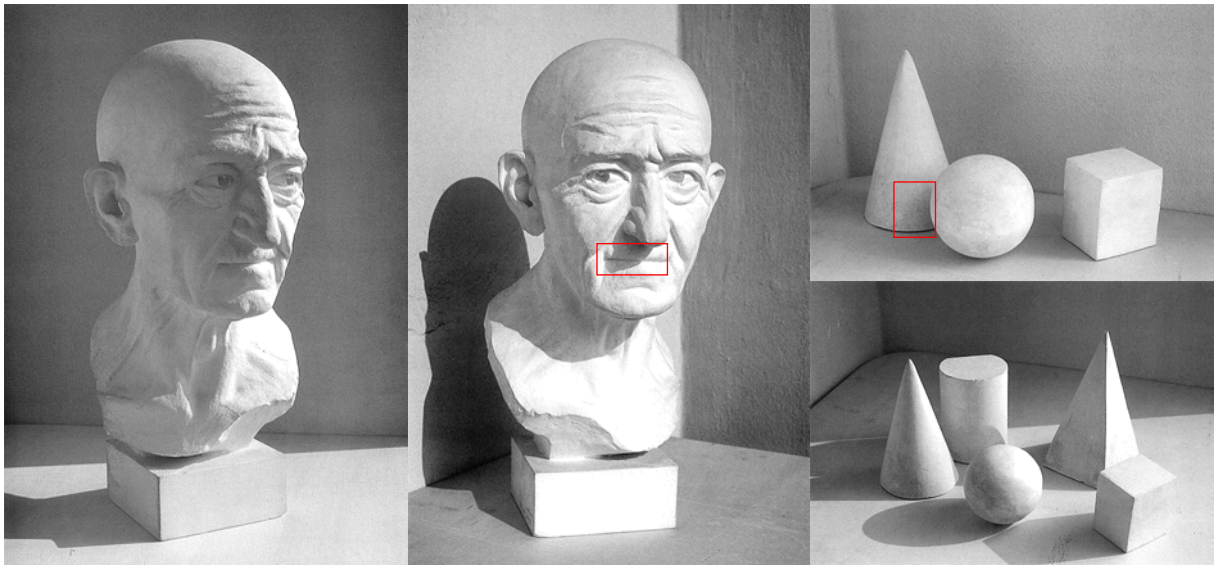


Figure 5.6: Shadings of diffuse-looking objects. In this setting, the shading mainly relies on the attributes of the light sources, such as strength of the light and position of the source, and the shape, or curvature, of the surface. Examples of cast shadows are highlighted in red rectangles. These require knowledge of which lights are visible from the point in question. Images from ref. 17.

surface influences the shading, bolstering the spatial understanding of the object by emphasising wrinkles and main parts of the face (mouth, eyes etc.). Cast shadows, which are not possible to capture with Phong's local illumination model, are highlighted in the red rectangles in the figure. These require knowledge of which lights are visible from the point in question. If a light is not visible then it does not contribute. Methods to determine visibility include ray tracing and shadow mapping.

In summary, the point that is most important to my work is that methods in 2D graphics aiming to accurately model local illumination, according to models such as Phong shading, require knowledge of the normal fields of the objects depicted. While 2D drawing systems should not necessarily require the artist to define such normal fields, developers of such systems should be aware of the fact that different object shapes can give rise to a wide range of shading profiles across the image domain. Thus, the artist should, somehow, be able to not only adjust smooth colour functions, as in basic first-order DCs, but also be able to control the propagation of those colours, as demonstrated by second-order DCs and our shading curve. In this way they simulate the effect of realistic local illumination.

5.2.2 Shading in 2D graphics

There are two approaches to drawing shading effects in 2D computer graphics: draw shading directly on the 2D canvas, like our method and most related methods in vector graphics, or draw shading indirectly by estimating and employing normal fields. Related work for the former approach is presented in Chapter 2. I now briefly discuss previous work for the latter approach and I discuss why I decided not to pursue this line of research.

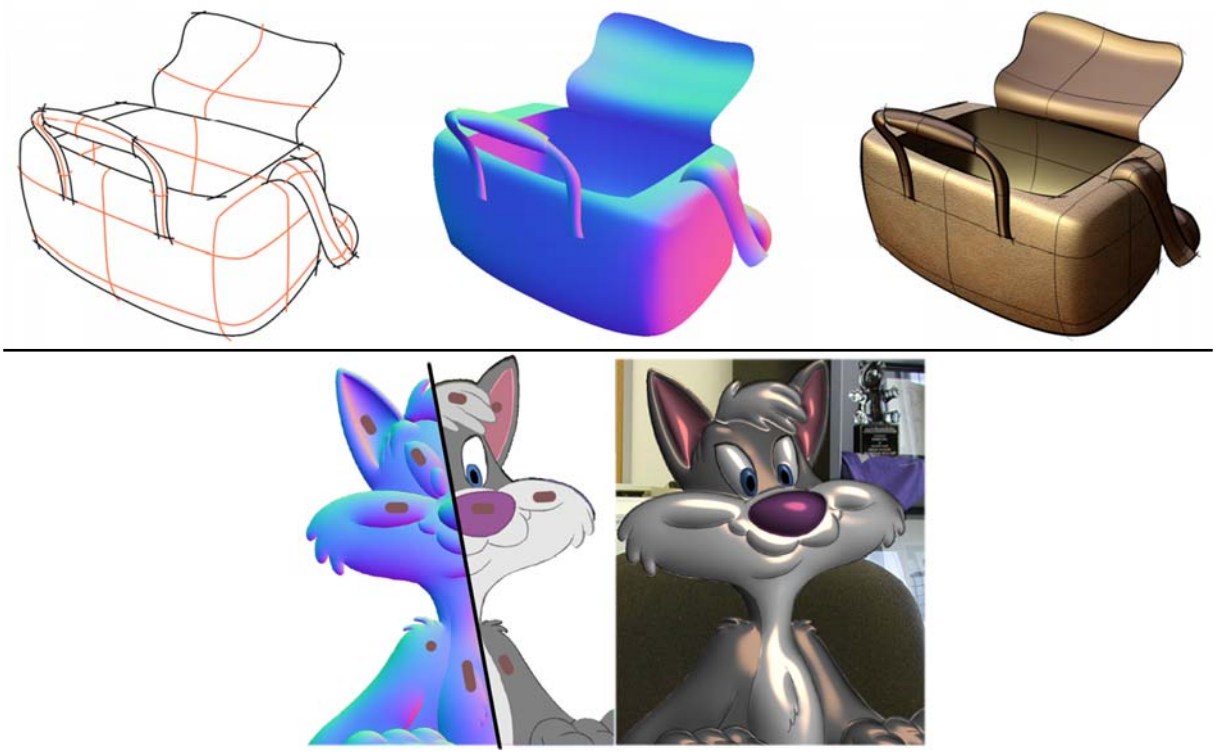


Figure 5.7: Shading in 2D with normal fields. Top: normal field (middle) extracted from a concept sketch with cross sections (left). Image rendered with a non-photorealistic shading model (right). Images from ref. 92. Bottom: complex effects achieved with a material-shading model based on the rendering equation. Inputs are a normal field, the original image, and strokes for manipulating highlights. Images from ref. 102.

A wide range of research has aimed to extract normal fields from 2D imagery. For example, given an input photograph or painting and a set of user-defined constraints defining a sparse set of known normal vectors, successful results have been demonstrated by propagating these normal vectors using various optimisation procedures[112;71;109]. Applications include relighting, embossing, and shading.

Researchers have also aimed to extract normal fields automatically, without any user input. To do so, various assumptions must be made. Given that normal vectors on object silhouettes in the image are aligned with the image plane, one assumption is that the normal field inside the domain of the silhouettes are defined as smooth functions. Such functions include harmonic functions[107] and bi-harmonic functions[43]; thus, solutions similar to DCs. While this assumption does not apply for general objects, artistic applications have been demonstrated, including shading of fonts and strokes, drawing of drapery, and shading of simple shapes (like smiley faces etc.).

In a similar manner, recent work has aimed to extract normal fields from concept sketches, typically used for product design[92] (Figure 5.7(top)). Such sketches are drawn with curves forming cross sections. A normal field is extracted from such a sketch with Coons patches, where the boundary for each patch is defined by the cross sections of the input sketch. The shaded image, using the extracted normal field, is rendered with the non-photorealistic shading

technique of Gooch and colleagues[33], which can be derived from Phong shading.

Assuming that an accurate normal field is already defined, a recent method has been proposed to manipulate material, as well as shading[102]. Since Phong's reflectance model does not incorporate spatially-varying materials, an alternative reflection model, assuming isotropic materials, was derived from the rendering equation. Image deformation operators were constructed from this model for shading and complex material editing (Figure 5.7(bottom)).

All of the above methods assume the objects to be defined in the image plane (that is, no depth coordinate). As mentioned in previously (illustrated in Figure 5.6), global illumination aspects, like cast shadows, cannot be achieved with such local models. Thus, to model more global aspects of illumination, the spatial relationship between objects, via depth information, must be defined. A recent method demonstrates that such aspects can be achieved in the 2D setting by additional user interaction to establish and manipulate geometry, occlusions, layers, inflation, and stitching between layers[97].

For the problem presented in this chapter (drawing shade and light in 2D), I did not aim to estimate the normal vectors required for Phong-style shading or depth information for global illumination effects. The rationale for this decision is that estimating normal vectors and depth information in the general setting of 2D images is challenging and therefore prone to give a result other than that desired by the artist. Alternatively, solutions to the problem of having the artist define a normal field could be improved. However, I am motivated by a desire to develop concepts that are easy to understand and a desire to develop methods that resemble traditional drawing techniques as close as possible. Following the discussion in Section 5.1, the shading curve primitive relates more closely to these motivations compared to the approaches presented above. Additionally, I note that our primitive is only inspired by the problem of creating shading effects for 2D vector graphics. However, our primitive is also adaptable to more abstract scenarios where smooth colour gradients are suitable. By contrast, vector-field-based methods aim specifically to achieve 3D illusions and shading effects in their results.

5.3 Vector-based lighting and shading with shading curves

In this section, I present the framework for defining and rendering shading curves. Figure 5.8 shows our prototype user interface for drawing shading curves. The artist draws the curves in a main window. This main window can be configured to view visualisations of the control meshes, produced with Solution 2 described in Chapter 4, the resulting image, or intermediate surfaces; note that viewing the intermediate surfaces is more suited for research and debugging purposes than for drawing. On the left hand side, the attributes of the shading curve, including the shading profiles, are edited. In the main editing window, the artist can select curves for global edits or control points for local edits. The purpose of the green and red colourings is to separate the treatment of the two sides of the shading curves, which are independent of one another except that they share the same base curve.

The framework for creating and rendering the shading curve primitive consists of three steps. First, input shading curves are manually created, along with their attributes (Section 5.3.1). The shading curve and profile are both defined as cubic subdivision B-spline curves and are created

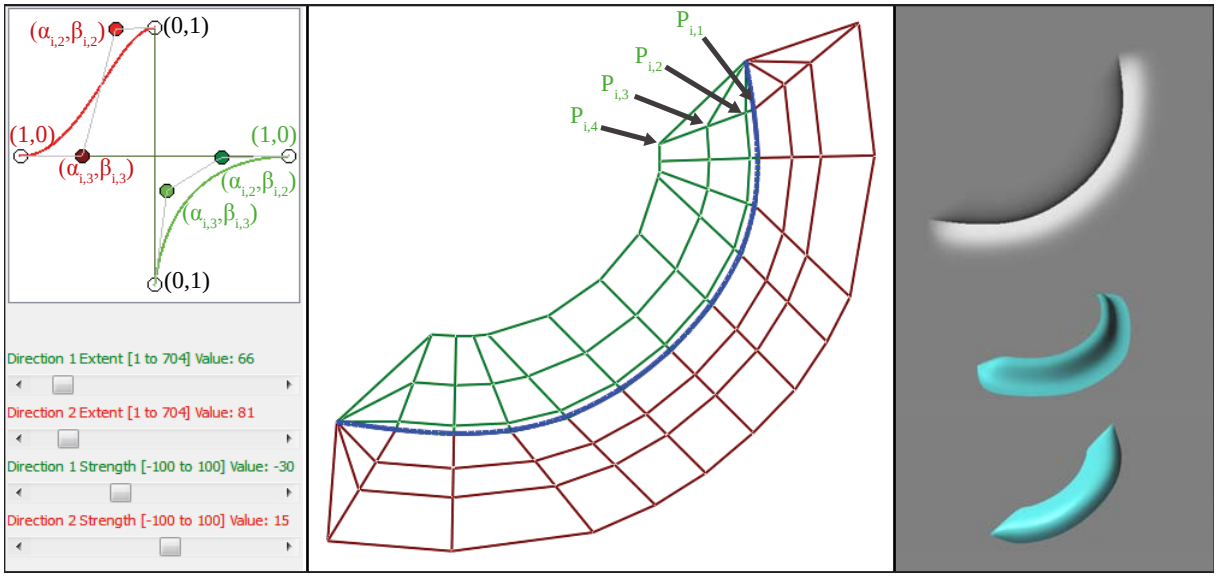


Figure 5.8: A simple shading curve in the prototype user interface. Left: The extent and height (labelled as Strength) are set with sliders and the shape attributes are extracted from the control points of a cubic B-spline curve defined in a normalised coordinate system. The user can change the shading profile by moving $(\alpha_{i,2}, \beta_{i,2})$ and $(\alpha_{i,3}, \beta_{i,3})$. Middle: The actual curve is edited in a typical curve editing interface. Right, top to bottom: the resulting shading and the two subdivision surfaces (bright and dark). The surfaces are rendered off screen and are not visible to the user.

with the prototype user interface shown in Figure 5.8. Second, the surfaces, used to interpolate the shading profiles in the image plane, must be automatically defined (Section 5.3.1). These surfaces are defined as Catmull-Clark subdivision surfaces. Finally, Section 5.3.2 presents the procedure of rendering the surfaces, producing the final image.

The framework is computationally efficient and the user is able to refine the input interactively, also after the surfaces are generated for the first time. Note that all of the steps related to rendering must be performed if the shading curves are moved in the image plane. In contrast, if the attributes of the shading curves, including the shading profile, are edited, the topology of the curves remains and only rendering the surfaces is sufficient; thus, it is not necessary to invoke the meshing procedure in Section 5.3.2 in this case.

5.3.1 Definition of control meshes

Recall the following definitions of the input curves and attributes from Section 4.1.2: An input B-spline curve is defined by a sequence of n control points $Q_i = (x_i, y_i)$, $i = 1, \ldots, n$. A set of attributes, height and extent, are attached to each control point:

- extent, $e_i \in \mathbb{R}_0^+$: defines the extent of the mesh in the perpendicular direction in the image plane from the curve at Q_i.
- height, $h_i \in \mathbb{R}$: defines the height of the mesh in the perpendicular direction to the image plane at Q_i.

The set of attributes is extended, in this chapter, with a shape attribute to support shading profiles: $((\alpha_{i,2}, \beta_{i,2}), (\alpha_{i,3}, \beta_{i,3})); \alpha_{i,j}, \beta_{i,j} \in [0,1] \subset \mathbb{R}$. Figure 5.8(left) illustrates that these shading profiles can be edited in a user interface by manipulating curves (rendered as cubic B-spline curves) defined in a normalised coordinate system.

As default, the height attribute defines changes in luminance using the LAB colour space. To support coloured profiles, the RGB colour space can optionally be used. Each curve therefore has an optional colour attribute C on each side. I did not found it necessary to associate colours to control points to demonstrate the results provided in this chapter.

Recall the following definitions of the output control meshes from Section 4.1.2:
$P_{i,j} = (x_{i,j}, y_{i,j}, z_{i,j})$, $i = 1, \ldots, n; j = 1, \ldots, m$, where n is the number of control points associated with Q_i. Having considered various alternatives, I chose to give the control mesh four rows (i.e. $m = 4$): one for the curve itself, one for the maximum extent of the curve (where the shading curve's influence drops to zero) and two to control the internal shape. The following coordinates can then be defined, given $Q_i = (x_i, y_i)$:

$$
\begin{aligned}
&P_{i,1} = (x_i, y_i, h_i);\\
&P_{i,\{2,3\}} = \alpha_{i,\{2,3\}} \left(P_{i,1} - Q_i\right) + \beta_{i,\{2,3\}} \left(P_{i,4} - Q_i\right);\\
&z_{i,4} = 0.
\end{aligned}
$$

The control points related to the shape attribute ($P_{i,\{2,3\}}$) are therefore placed on the plane defined by $P_{i,1}$, $(x_i, y_i, 0)$, and $P_{i,4}$, ensuring that the profile is indeed modelled in the direction towards $P_{i,4}$. The coordinates $(x_{i,4}, y_{i,4})$ are defined by Solution 2 described in Section 4.2.4. The two solutions presented in Chapter 4 are compared in Section 4.2.6, where I provided reasons to why Solution 2 should be employed, rather than Solution 1, for the application presented in this chapter. In short, Solution 2 is more useful for extent-centric applications, where the user directly manipulates the extent attribute, without the requirement of labelling corners.

Luminance clipping avoidance

In the setting of luminance enhancement in the LAB colour space, the z coordinates should be defined such that the surfaces do not produce luminance values that are outside the range of the luminance channel. In LAB, values in the luminance channel ranges from 0 to 100. Out-of-range values will give rise to clipping artefacts and the shading profile will not be correctly rendered.

To avoid clipping, $z_{i,\{1,2,3\}}$ is reduced, if necessary, by comparing with the related luminance value I_L of the underlying image. In our implementation, I_L is part of the definition of the underlying image and can be looked up for each curve. The following test is performed to keep luminance values below 100:

$$
z_{i,j} = \begin{cases} 100 - I_L & \text{if } I_L + z_{i,j} > 100; \\ z_{i,j} & \text{otherwise.} \end{cases}
$$

A similar calculation is performed to avoid negative luminance values.

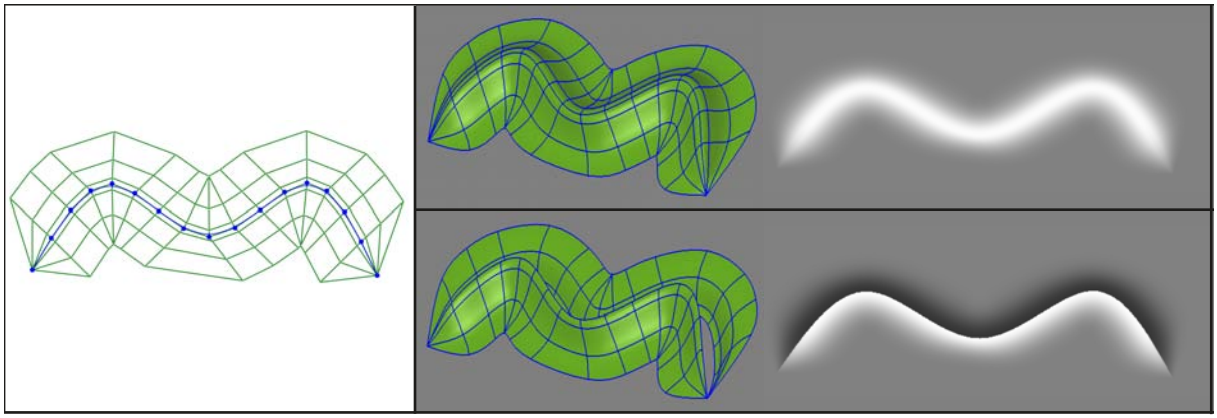


Figure 5.9: Slope (top) and boundary (bottom) curves. Slope curves compose single smooth surfaces and boundary curves compose two separate surfaces.

Boundary and slope curves

I have so far only treated the control meshes on each side of the curves as separate meshes. Thus, the transition across a curve will typically be discontinuous. However, one might wish to model a smooth transition across the curve. To this end, I implemented two types of curves: boundary curves for discontinuous transitions and slope curves for continuous transitions.

Figure 5.9 illustrates the difference between boundary and slope curves. The boundary curve associates one surface with each side of the curve. There is no requirement that both surfaces are enabled. By contrast, the slope curve creates a single smooth surface defined on both sides of the curve.

The only technical difference between these two types of curves is that, for each i, the $P_{i,1}$ on both sides of a slope curve are merged; that is, they appear only once in the control mesh. Note that boundary curves can implicitly be raised to slope curves. This happens when the height and shape attributes, defining $P_{i,\{1,2\}}$, are the same for both sides of a curve. Thus, the tangent planes for both $P_{i,1}$ are the same and the continuity across the curve is at least C^1.

5.3.2 Rendering

The surfaces, defined from the control meshes, are rendered as Catmull-Clark subdivision surfaces. Subdivision was chosen because the meshes can contain triangles. Frameworks requiring rectangular meshes, due to their tensor product definition (e.g. NURBS and Ferguson patches), model such triangular patches with degenerate (coincident) vertices which give rise to C^0 creases. By contrast, Catmull-Clark subdivision incorporates triangles with no difficulties. The resulting surfaces are guaranteed to be C^2 everywhere other than at isolated points where the continuity is C^1. Note that these isolated 'C^1-only' points cause no visible artefacts in 2D renderings as the surface is still smooth.

The process of creating the final image from the control meshes is straightforward: subdivide the control meshes, render the surfaces off screen, and then apply these surfaces to the original

image.

First, this 'original' colour image needs to be defined. In the prototype system, we chose to shade each region with a uniform colour where image regions are defined by closed curves. The colour defined in each image region is defined by the user. Thus, the effect produced by a shading curve can be viewed as an effect that is supplemented, in a single image layer, to any image. We chose to define such original images with regions of uniform colour since this is the simplest type of input, meaning that the contribution of shading curves are best studied when they are applied to such simplistic input.

The surfaces are defined with Catmull-Clark subdivision with the boundary subdivision rules developed by DeRose's team at Pixar[23]. The boundary rules ensures that the boundary of a surface matches its associated curve. Note that the semi-sharp features proposed by DeRose et al. are not used.

The number of subdivision steps is the same for all the images in this chapter (2). The control meshes are relatively simple and a sparse set of control points are used. From observation, the visual quality converges on two recursions. In some cases, where the surface is stretched over a large area, three subdivision steps can produce a visually smoother result. Note that our system remains interactive with three subdivision levels.

The layer representing the shading, S, is created by rendering the Catmull-Clark surfaces off-screen. In our system, this is achieved by using OpenGL, where the shading image is extracted from the depth buffer of this rendering. The camera parameters are set so that depth values directly correspond to the height attributes; thus $S \in [-100, 100]$ for luminance and $S \in [0, 1]$ for colour.

Finally, this shading image is applied to the original image I producing the resulting image R. Figure 5.10 shows the various options available. In the setting of luminance adjustment in the LAB luminance channel I_L, the shading image is simply added to this channel:

$$R_L = I_L + S. \tag{5.2}$$

The two other colour channels are set by the corresponding channels in I.

In the setting of colour adjustment, the result is created by linearly interpolating between the original image and a colour image, I_C (Figure 5.10(c-d)):

$$R = S * I_C + (1 - S) * I. \tag{5.3}$$

In our system, I_C is created by colouring the pixels of a surface with its associated colour C (Figure 5.10(c)). This produces an image with constant colours in regions defined by surfaces, with a black colour at pixels not related to surfaces. Alternatively, I_C can be produced with a gradient mesh rendering if the colour attribute is associated with curve control points Q_i and not to the entire curve (Figure 5.10(d)). This means that all mesh control points $P_{i,j}$ associated with Q_i are given the colour C_i associated with Q_i. Since the mesh contains triangles the gradient mesh must be rendered with the solution presented in Chapter 6.

The resulting image R is well suited to be used in a composition of images using layering. Commercial tools, such as Illustrator, support a wide range of layering modes. We found the

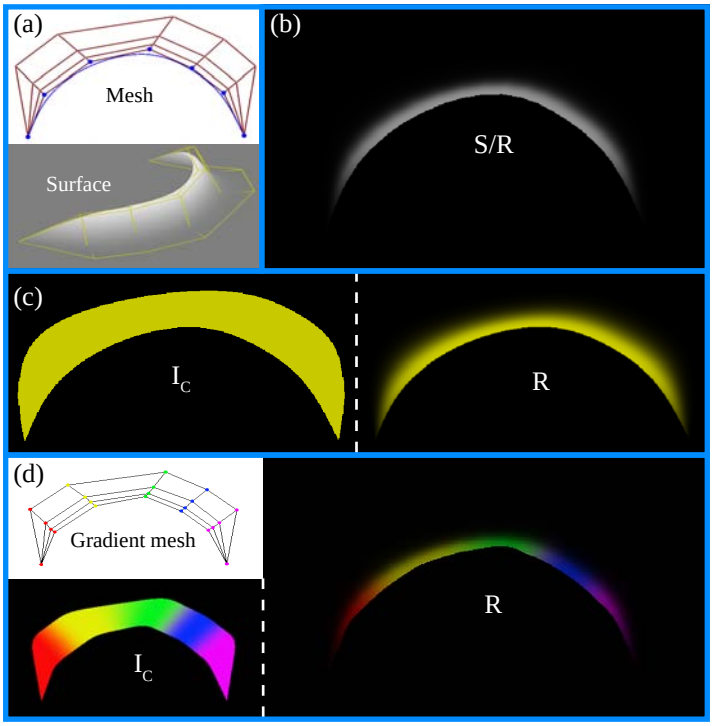


Figure 5.10: Options on colour space and method of rendering. Given an input surface (a), the depth buffer S is extracted in an off-screen rendering of that surface (b). As the colour of the underlying image is black (i.e. its luminance equals zero), S corresponds, in this example, to the final image R of luminance adjustment, according to Eq. 5.2. Two methods for rendering colour are available: (c) associate a single colour with a curve, defining the image I_C as a piecewise linear colour image; (d) or associate colours to curve control points Q_i and render a gradient mesh image I_C with these colours. For both methods, I_C is combined with S, according to Eq. 5.3, to produce R.

multiply (which darkens the image) and screen (which lightens the image) modes particularly useful in the setting of shading. Given two images a and b, multiply mode multiplies the two images ($f(a, b) = ab$). Screen mode is designed to provide the opposite effect of multiply mode and is defined as: $f(a, b) = 1 - (1 - a)(1 - b)$. If multiply mode is used, the underlying colour should be white (meaning no adjustment since the values are 1). Conversely, for screen mode the underlying colour should be black (this is demonstrated in Section 5.4, Figure 5.16).

5.4 Results

The shading curve is adaptable to a wide range of visualisations. In this section, I discuss various types of visualisations that can be drawn using shading curves and I explain how this primitive can be used. The prototype system was also informally tested with novice users (for example, Figure 5.11), which indicated that the primitive is easy to learn and use.

Recall from Section 5.1 the three-stage approach to drawing with the shading curve: (1) outline objects and main tonal areas, (2) fill in colours and define the influence of those colours, and

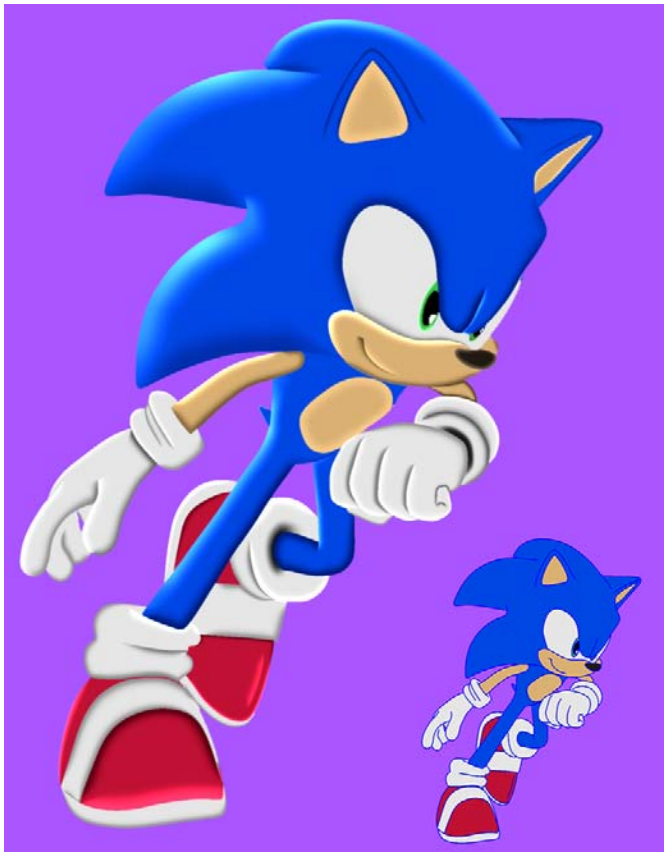


Figure 5.11: Image shaded with the system by a user with no artistic training in less than 10 minutes. The curves and colours (bottom right) were provided as input, which were drawn from an image of Sonic ©SEGA.

(3) smooth out colours with shading profiles. In the following discussion, I present various tips to how the final stage can be efficiently performed.

Figure 5.12(a-b) shows cartoon-like images created using the default settings ($(\alpha_{i,2}, \beta_{i,2}) = (0.15, 1.0)$; $(\alpha_{i,3}, \beta_{i,3}) = (0.4, 0.0)$; $h_i = 20$; $e_i = 3\%$ of the image diagonal); height is manually set negative for dark shadings. This is similar to Adobe Photoshop's bevel and emboss option for creating enhanced 3D cartoon looks. Adding slope curves to such images, as in Figure 5.12(d), can induce more glossy looks.

Simulated 3D effects can be achieved by varying the attributes along the curves (Figure 5.12(c-d)). From my experience, the best practice is to initially start with cartoon shading defined by the default settings, and then increase or decrease the height at selective locations along boundary curves. This will add a sense of location of the main light sources in the scene, as well as their strength (Figure 5.12(g)). The two other parameters, extent and shape, are then varied to give the shading form. Extent is typically set equal for all control points along a boundary at the preferred offset. This attribute does not notably influence the shading.

The shading profile is therefore the most important factor for creating various types of shadings (Figure 5.13). Varying the first shape point related to $P_{i,2}$ between $(0, 0)-(0, 1)$ $(\beta_{i,2} = 0)$ in the normalised coordinate system creates a steep shading fall off, and is suitable for more diffuse

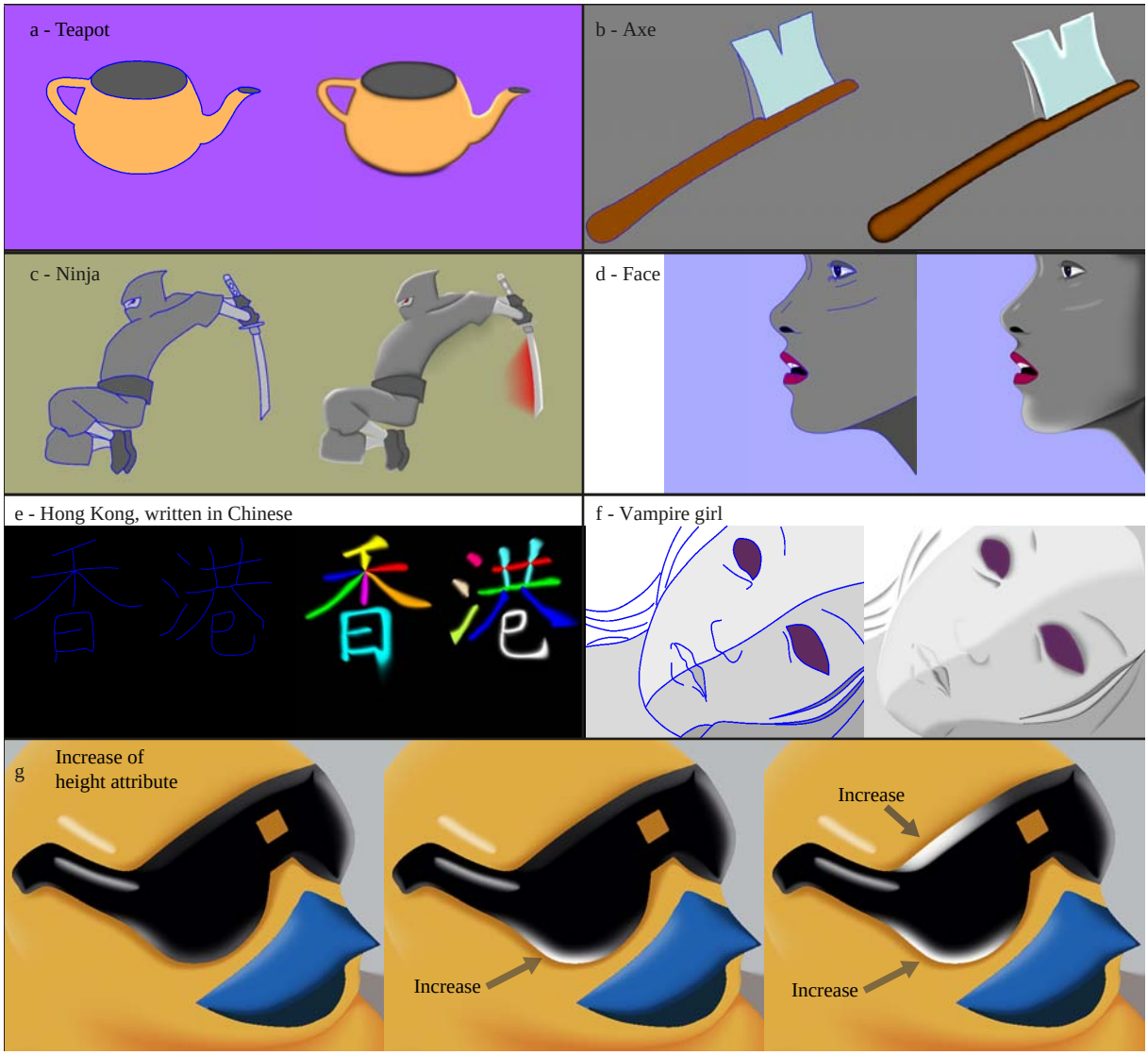


Figure 5.12: Images drawn with the system.

conditions. On the other hand, varying this shape point between $(0,1)-(1,1)$ ($\alpha_{i,2}=1$) creates a stronger profile out from the boundary and is suitable for depicting cast shadows and shading highlights. The second shape point, defining $P_{i,3}$, should be placed along $(0,0)-(1,0)$ ($\alpha_{i,3}=0$) for a smooth fall off to zero at $P_{i,4}$. The location of this fall off in the image is then controlled by this shape point. The offset curve defined by $P_{i,3}$ can also be shown in the main editing window so that the artist can directly visualise this fall off.

Figures 5.14, 5.16, and 5.18(left two images) show images shaded with the chiaroscuro-inspired drawing technique described in Section 5.1. Each area is defined as single layer and are all blended either with screen or multiply blending modes to produce the shaded result.

The choice of colour space can influence the visual aspects of the shaded image. The advantage of employing luminance adjustment, and not colour adjustment, lies in drawing efficiency. The artist is only concerned with the parameters forming shading profiles and is not concerned with aspects of colour; that is, the artist only defines shading profiles that either darken or lighten the

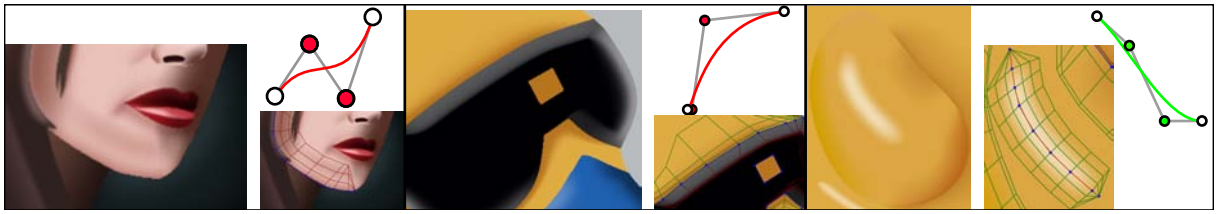


Figure 5.13: Shading profiles for various effects. Left: the chin is emphasised with a wavy profile. Middle: the 3D shape of the glasses is conveyed by the use of strong light and sharp fall off of the profile. Right: a specular highlight is drawn with a bell-shaped profile. The superimposed grids can be used to visualise the shading fall offs of the shadings.

Figure 5.14: Drawing with the three-step approach explained in Section 5.1. The smaller images, in the centre of the figure, comprise regions that are each shaded with a single colour. The larger images at left and right are these same images shaded off using the proposed shading curve framework.

underlying tone or colour and specifies their strengths. I have found such luminance adjustment suitable for shadings with boundary curves, as demonstrated in Figures 5.11 and 5.12(a;b;d). On the other hand, colour adjustments are obviously suitable when coloured profiles are defined. Additionally, luminance adjustment can produce unwanted colourings in certain cases, as demonstrated in Figure 5.15, where white highlights was preferred over the golden colourings created with luminance adjustment.

The shading curve is well suited as nodes for *vector shade trees*[92]. In contrast to Chiaroscuro-style drawing, drawing with vector shade trees is targeted towards depictions of specific materials. An image defined by vector shade trees is defined by multiple layers structured as binary trees. The nodes represent layering modes, such as screen and multiply, and the leaves represent image regions shaded with vector graphics. A given tree represents a target material and can be applied to new objects automatically. An image layer defined by shading curves is suited for this application since it is by construction targeted towards layering. Figure 5.16 shows two types of materials (candle wax and the material of the body of a car) defined as vector shade trees using shading curves as base nodes.

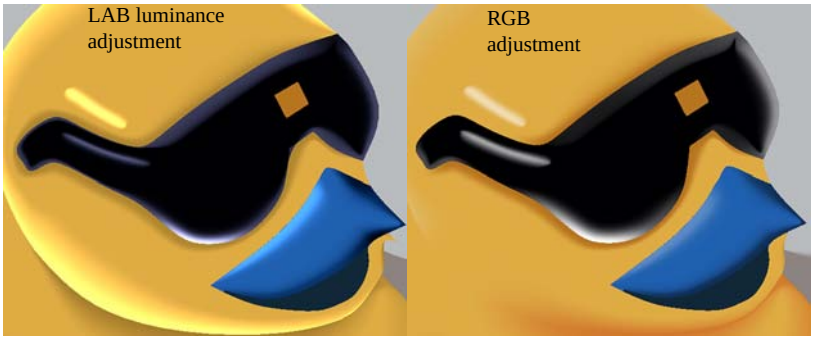


Figure 5.15: Adjustment in RGB was preferred in this scenario due to unwanted golden colourings associated with luminance adjustment of the underlying yellow colour.

	#CPs	#Ss	DDTT	SB	Resolution
Fig. 5.12 Teapot	140	25	6	17	800×600
Fig. 5.12 Axe	60	7	10	10	1024×1024
Fig. 5.12 Ninja	324	75	5	29	958×748
Fig. 5.12 Face	235	22	13	12	900×1302
Fig. 5.12 Vampire Girl	278	49	3	25	689×436
Fig. 5.11 Sonic	828	172	18	70	800 ×998
Fig. 5.12 Hong Kong	262	64	15	36	1000×720

Table 5.1: Timings for the single-layer results presented in this chapter. Columns, from left to right, show the figure's and the image's identifier, number of control points, number of surfaces, timing for the DDT tracing, timing for subdivision, and the image's resolution. The times are in milliseconds, and are averaged over 100 runs.

5.4.1 Performance

The performance of our naïve C++ single-core CPU implementation of the framework is reported in Table 5.1. These timings were recorded on a system running Ubuntu 12.04 on an Intel Core i7 CPU (2.93GHz x8). The system is interactive and responsive; all images have been produced under 200 ms, giving at least 5 updates per second, which is sufficient for an interactive drawing application. Its current bottleneck is transferring data to the GPU for OpenGL surface rendering, which takes about 50 ms.

There are various factors which influence the performance. For subdivision, the number of control points is the only performance factor. Increasing the subdivision level by one will make subdivision about four times slower. In the current implementation, based on images shown in this chapter, the program can perform three to four subdivision steps without sacrificing interactivity. This performance could be further improved with a GPU implementation[69]. From the reported timings on such implementations, one can expect a performance boost of at least one order of magnitude.

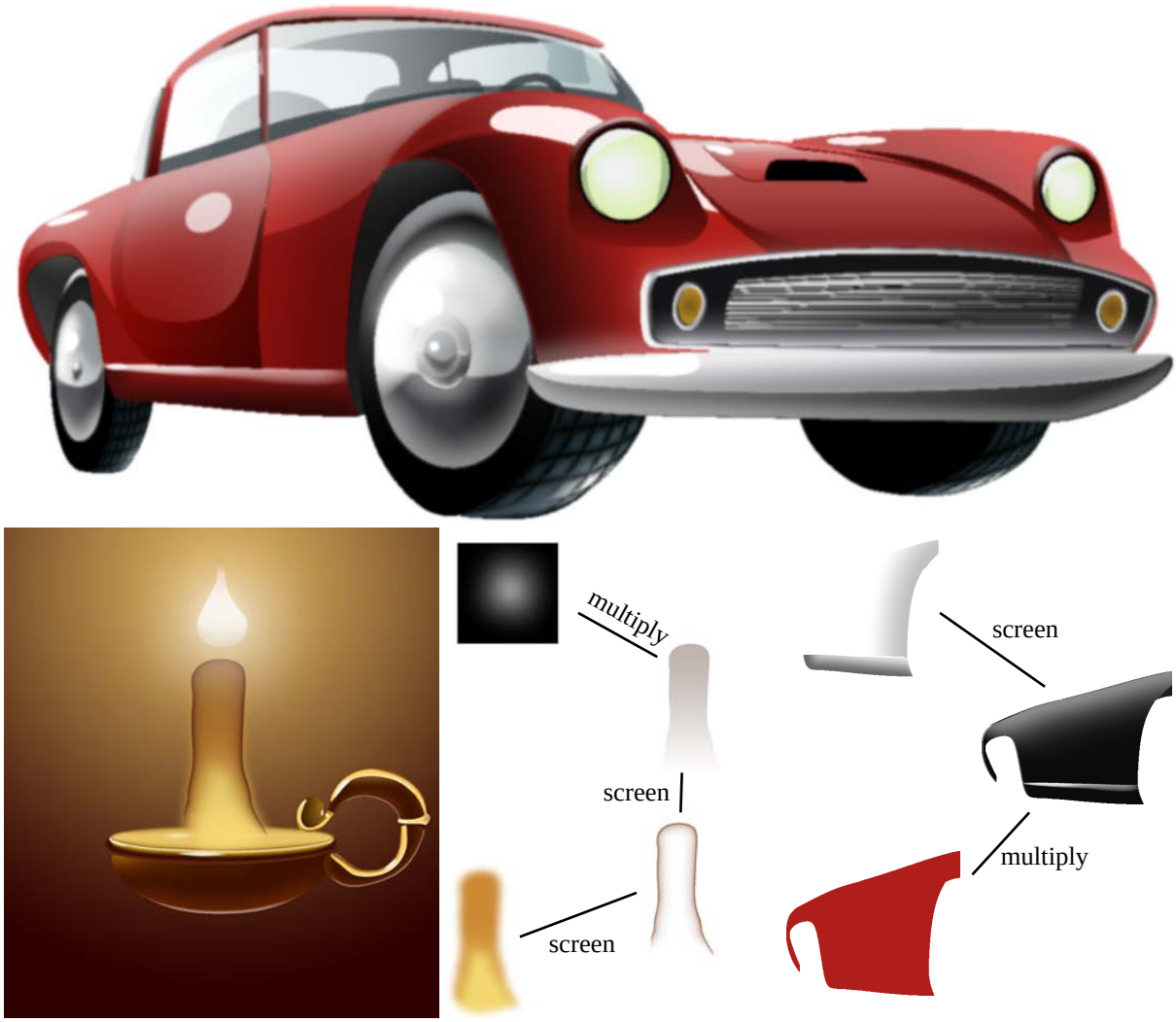


Figure 5.16: Depicting two types of materials with vector shade trees (bottom right) with the proposed primitive as base nodes. The images were manually drawn and shaded, with inspiration from ref. 61.

5.5 Comparisons with related work

The main difference to previous methods aiming to render similar primitives is that the shading curve employs subdivision surfaces. In contrast, DCs are evaluated via partial differential equations (PDEs). Additionally, similar primitives implemented in commercial software, such as Adobe Illustrator, are associated with linear gradients or gradient meshes. In this section, I compare and contrast all of these methods with our subdivision-based solution.

5.5.1 Comparison with first-order diffusion curves

The original DC primitive is defined via the Laplacian differential operator. The PDE is constrained by RGB colours the solution needs to take on the boundary. Thus, the approach pro-

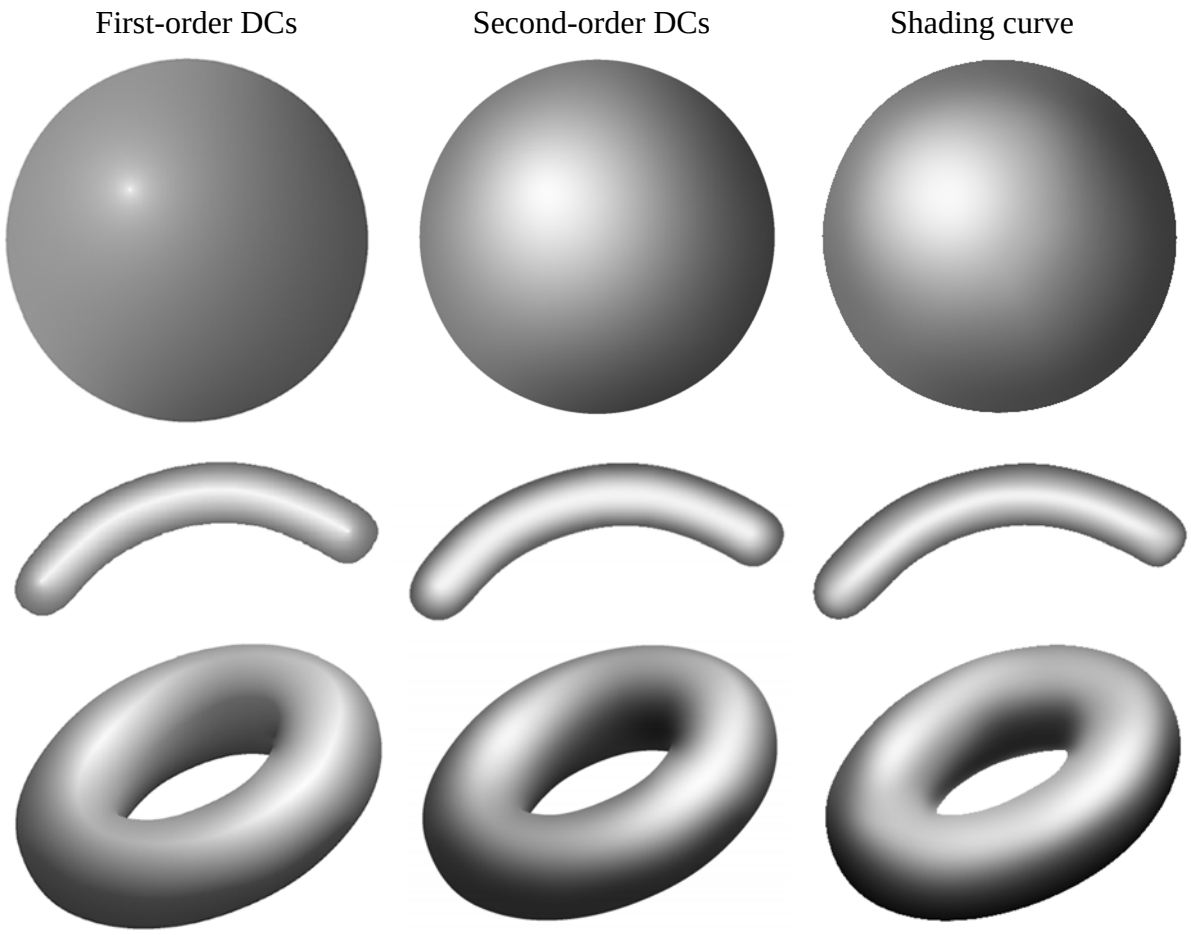


Figure 5.17: Shading a sphere, a tube, and a donut with three competing methods. The comparison between the two diffusion curve methods is fair and accurate, as the inputs are the same. However, the comparison to shading curves is inaccurate, as the inputs are fundamentally different from DCs and they are defined separately in different user interfaces. The results for the diffusion curve methods are from ref. 31.

vides easy control over colours at sharp boundaries.

The basic first-order DCs primitive (that is, not assuming additional attributes like blur or weights) is simpler than our primitive. The DC primitive can be associated with a single attribute on each side of the curve: colour. By contrast, our method requires the additional width attribute (assuming that height and shape are fixed). In certain types of scenarios, where colour images produced by Laplacian diffusion provide acceptable results, the DC primitive can be preferable as less input is required.

The disadvantage of first-order DCs, compared to the shading curve, is that they are limited to harmonic solutions. The artist is therefore constrained to first-order diffusion conditions. Figure 5.17 demonstrates this point: the solution of first-order DCs is only smooth away from constraints. Thus, this solution produces creases along curves, which can only be smoothed with by post-processing operations such as image blurring. By contrast, methods with control over derivatives *and* gradients, such as the shading curve and second-order DCs, support smoothness

across input curves.

5.5.2 Comparison with second-order diffusion curves

The limitations of first-order DCs motivated Finch and colleagues[31] to propose second-order DCs. The PDE is now constrained by RGB colours and derivative constraints. The derivative constraints enable modelling of virtually endless types of shapes of the colour function. In the framework proposed by Finch et al., a curve associated with derivative constraints will force the derivative of the colour function to zero either across the curve or along the curve. The authors did not explain why only zero derivatives were used or whether other settings were experimented with. However, experiments conducted for the problem presented in Chapter 6 indicate that non-zero derivative constraints easily over-saturate the colour function.

Comparison with this framework is challenging as the two competing methods both target similar goals, whilst being fundamentally different: our method produces spline functions defined with subdivision and the solution of Finch et al. produces thin-plate splines modelled with PDEs. In the following, I compare the mathematical differences between the two methods. I do not argue that our method is 'better' than second-order DCs, although both methods have their strengths and weaknesses. However, I do argue that the shading curve is a strong alternative to second-order DCs. Note that, in this section, 'DCs' refer to second-order diffusion curves.

Mathematical differences

Our method produces Catmull-Clark subdivision surfaces. In the regular setting, these are C^2 bi-cubic B-spline surfaces. Irregular elements in the control mesh, which in this setting are related to vertices of triangles, are preserved in each subdivision step. The continuity across such irregular elements is C^1.

On the other hand, the solution of Finch et al. produces thin-plate spline surfaces. A thin-plate spline is a polyharmonic spline and is a generalisation of the univariate cubic spline to higher dimensions. The formula for a thin-plate spline is the fundamental solution to the biharmonic PDE equation and is therefore suitable for vector editing with gradient control. Since thin-plate splines are solutions to the biharmonic equation, they can be viewed as 'least-squares harmonic' or 'as harmonic as possible'[31].

The two frameworks, Catmull-Clark subdivision and PDEs, are fundamentally different. In the regular setting of B-splines, the resulting spline surface is defined by a set of basis functions and the given control mesh. In subdivision, corresponding subdivision rules for vertices, edges, and faces can be defined to exactly produce B-splines of a given order. In Catmull-Clark subdivision, special rules are applied to irregular vertices and faces to produce smooth surfaces everywhere. By contrast, PDEs are defined by boundary conditions and are therefore not defined by control meshes. Such boundary conditions relate to values and derivatives of the underlying (unknown, multivariate) function. Moreover, many 'pre-defined' PDEs have been defined for various applications in scientific fields like engineering, physics, and computer science. DCs are concerned with Laplace's equation ($\nabla^2 f = 0$, or in biharmonic form: $\nabla^4 f = 0$). Boundary

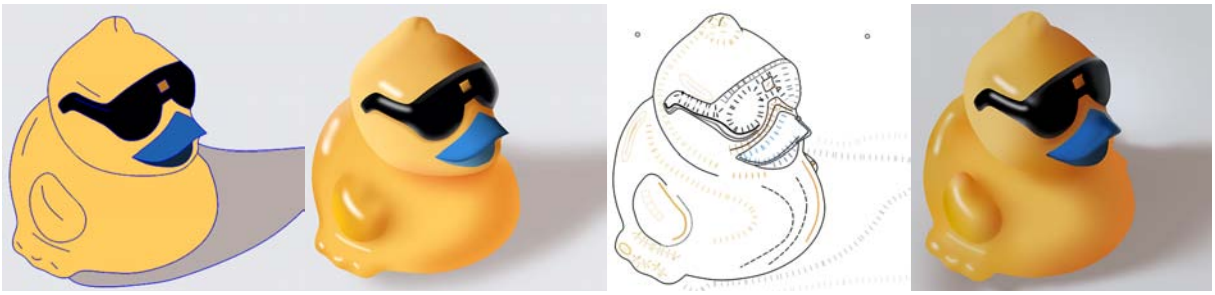


Figure 5.18: Informal comparison between the shading curve (left two images; drawn by myself) and second-order DCs (right two images from ref. 31). The artistic skill influences the quality of these images and the comparison is therefore inaccurate.

conditions for Laplace's equation include Dirichlet conditions, defining the value of the function at the boundary, and Neumann and Cauchy conditions, defining the normal derivative of the function at the boundary.

The influence of the two types of inputs, mesh control points for subdivision and boundary conditions for PDEs, on the colour function is different. In the regular setting for Catmull-Clark subdivision, the influence of a control point is defined by its related basis function. Recall from Section 3.1 that these basis functions are defined with minimal support, meaning that edits to control points give rise to local edits of the surface. Due to this minimal support property of B-splines, such edits can be viewed as 'as local as possible'. Consequently, the subdivision rules applied to mesh control points are only defined with non-zero weights in their local neighbourhoods. Rules for irregular elements are defined in a similar local manner, ensuring local control also at irregular elements. By contrast, PDE boundary conditions are not designed to provide such local control. For example, the heat equation, closely related to Laplace's equation, allows given heat values at the boundary of the domain to flow until a stationary state has been reached. The colour diffusion process employed by DCs behaves similarly to define a function that is as harmonic as possible.

From the discussion above, interesting analogies can be made with respect to the artistic perspective of the problem of drawing smooth colourings. Our subdivision-based solution is akin to surface modelling technology for CAD where the designer controls surfaces by approximating control meshes. Similarly, drawing with the shading curve involves implicit design of surfaces via control mechanisms familiar to a 3D designer, such as control polygons defining object boundaries and shading profiles, with additional attributes tuned for the 2D domain such as shading extent and strengths. On the other hand, drawing with DCs can be thought of as a process of building a physical simulation of colour propagation (as in colour diffusion or colour 'heating'). The simulation system is built with curves representing boundary conditions and the corresponding domain is coloured over time via a diffusion process until equilibrium is reached. Figure 5.18 informally compares the two methods with regard to the types of input curves associated with a typical drawing.

More concrete advantages and disadvantages between the shading curve and second-order DCs are now discussed. In short, the advantage of DCs is better control of vivid colourings, while advantages of the shading curve include more local control of the colour function, being layer-friendly, and being more computationally efficient.

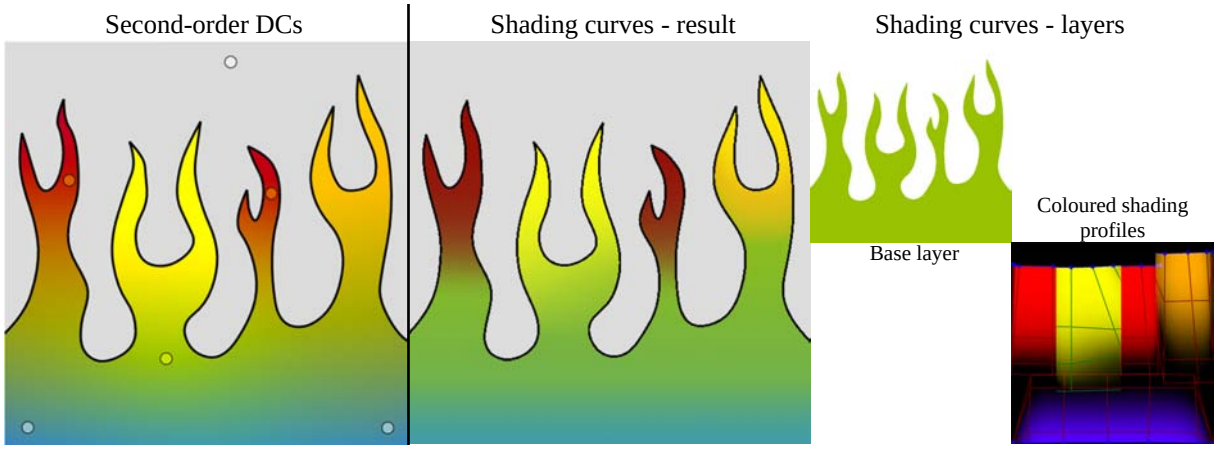


Figure 5.19: Limitation of the shading curve compared to second-order DCs: vivid colourings can be achieved by a sparse set of point constraints with second-order DCs. A similar colouring to this example can be achieved with shading curves by applying shading-curve colourings to a single layer in the image (the layer related to the 'fire'). Notice how the yellow and orange colours for the DC method have been produced by the diffusion process and were not specified by the user. Such behaviour can either be an advantage (less input) or disadvantage (unexpected behaviour) depending on the user's preference and the situation. Note that point constraints only defines interpolation with the selected colours; the colour gradient cannot be manipulated similar to the shading profile. The result for DCs is from ref. 31.

The advantage of DCs is related to vivid colourings. Our method uses surfaces as interpolants between the underlying image and the colour or luminance associated with shading curves. Thus, colour propagation is controlled along a shading curve and can only be manipulated in its perpendicular direction with shading profiles. While DCs do not support the flexibility of shading profiles they support points as constraints (Figure 5.19). A point constraint adds boundary constraints to its four nearest pixels (thus, it is more like a square constraint). In this way, the colouring of an object can be separated from the curves defining the object's boundary by placing point constraints inside the object. This approach can be preferred over drawing 'special' curves, as required by shading curves, to achieve certain colourings (e.g. Figure 5.19). This limitation of our method is further discussed in Section 5.6 and motivates the problem described in Chapter 6.

Aspects where the shading curve is preferable over DCs include local gradient control, layering, and computational efficiency:

– The shading profile, modelled with Catmull-Clark subdivision, provides a local way to model colour gradients. The extent of the edits are local because the control meshes are directly created from the extent attribute of the shading curve. The shading profile is ensured to only locally influence the colour/luminance function due to the minimal support property of B-splines. By contrast, DCs do not provide a way to locally control the colour gradient. Instead of curves, representing the gradient of the colour/luminance function in a given direction, DC's derivative constraints are restricted to forcing the derivative to zero in a given direction. Thus, DCs do not possess the same degree of freedom for manipulating the colour gradient. Additionally, the boundary conditions of

DCs are global constraints, influencing the entire colour function. Providing a similar mechanism to the shading profile is therefore challenging.

– DCs are not particularly suitable for layering. In image compositing, the spatial extent of a layer should be easily defined. This is ensured with the shading curve using the extent attribute. By contrast, the solution of DCs must be well defined inside the boundary specified by the DCs. This boundary must be explicitly defined by curves, meaning that influences of open curves and point constraints are not easily constrained. In practice, single objects, fully enclosed by curves or the image border, can be separated into layers. Local effects like specular highlights, however, would not typically be treated by layering and should be incorporated as a single layer to ensure that their effects are smoothly blended with each other.

– Interactive performance is easily achieved with our method. Even the naïve CPU implementation provides acceptable performance. By contrast, a diffusion process solves a large, sparse, linear system which is naïvely time consuming. Approximate solutions with low-resolution CPU finite element methods[10] or high-resolution (re-evaluation-only) boundary element methods[40] can be employed in order to achieve interactive speeds.

The challenges of constructing the two types of frameworks are now discussed. For DCs, solving the PDE in the discrete setting involves the setup of a linear system. To achieve this, the (bi-)Laplacian operator, defining Laplace's equation, must be defined in the discrete setting. Additionally, operations to discretise the boundary conditions must also be defined. To discretise such conditions from points or curves, the pixels associated with the condition values must form a water-tight boundary so that the boundary is well defined, thus avoiding leakage. Furthermore, discrete kernels for derivative constraints of any given direction (that is, anisotropic kernels) must be defined. To construct and render the shading curve, the meshing problem described in Chapter 4 is the principal challenge. The operations needed to form the complete framework, described in Section 5.3, are well understood and relatively straightforward to implement. While some of the challenges faced with the construction of these two frameworks are challenging, such as meshing and kernel discretisation, robust and computationally efficient solutions to all of these challenges have been proposed (see ref. 72 and ref. 31 on the fine details on how to construct DCs). Thus, I do not think there are any particular practical disadvantages of the defining aspects of the competing frameworks.

Finally, can these two competing frameworks approximate the behaviour of each other? That is:

– Can DCs be re-configured to support shading profiles?
The DC framework only supports derivative constraints that forces the colour function to zero in a given direction. To support colour propagation akin to a given shading profile, the derivative constraints must be re-configured to support non-zero derivatives. In Chapter 6, I demonstrate that non-zero derivative constraints are difficult to control, as such constraints give rise to over-saturation of the colour function. This is therefore a challenging problem.

– Can the proposed shading curve framework be re-configured to support DCs?
The proposed framework is not directly applicable to diffusion curves, since shading profiles are explicitly modelled out from curves and it does not natively support the diffusion

process employed by DCs. However, one could imagine a hybrid between our solution and DCs. That is, the values of the control points are not dictated by a shading curve, but are instead given colours via a diffusion process or a biharmonic interpolation procedure. This is a promising idea and is further discussed in Chapter 6.

5.5.3 Comparison with Adobe Illustrator

In commercial products, similar primitives targeted toward colouring aspects of vector creation have been implemented. I have found Adobe Illustrator to be the strongest alternative, as it provides a richer set of primitives with more degrees of freedom than related products, such as Corel CorelDRAW (this conclusion was drawn by comparing the list of features between related products as I do not have access to all of them). In the following, I compare with the two closest related primitives for colouring: the brush tool and the gradient mesh tool. Note that the technical details of these primitives have not been made publicly available and parts of this discussion are based on reasoned guesswork.

Comparison with the brush tool

Illustrator's brush tool supports linear and radial directions of colour gradients both along and across the brush stroke. The feature of applying the gradient across the stroke, named 'gradient on a stroke', was added in version CS6, 2012. Recall from Section 2.2.1 that a linear or radial gradient is defined as an editable univariate colour function applied in a given (linear or radial) direction. The gradient-on-a-stroke feature is similar to the shading curve since it can be combined with Illustrator's width tool (introduced in version CS5, 2010) to define editable colour gradients out from curves in a given extent. While the extent can be varied along the curve, using the width tool, the colour gradient must remain constant for the entire stroke.

The skeletal stroke framework[38] covers most aspects of the brush tool. Recall from Section 4.1 that this framework provides a way to transform a pre-defined image along a path. Given a gradient profile, this can then be rendered along the curve in its perpendicular direction, producing a 'gradient on a stroke'. Note that the skeletal stroke framework is only concerned with the deformation procedure. It is therefore not obvious how the gradient profile is actually rendered.

Mathematically speaking, the gradient-on-a-stroke tool and the shading curve are different. On one hand, they are defined similarly as they are both associated with a univariate function representing the curve's colour gradient. On the other hand, the renderings of the two primitives are fundamentally different. Illustrator's tool renders the brush primitive with reference to its associated colour profile. By contrast, our method creates a bivariate function (i.e. a surface), shaped according to the shading profile. Then, the rendering of the shading curve is related to the surface instead of being directly produced from the original shading profile.

Rendering the univariate colour function directly can give rise to several visual artefacts. In Figure 5.20, for example, artefacts are visible along the curves produced by Illustrator's tool. Note that as the rendering process of the primitive is undisclosed, it is challenging to pinpoint the exact source of these artefacts. Additionally, recall from Section 2.2.1 that the deformation procedure of skeletal strokes can produce folding artefacts. Such folds also give rise to artefacts,

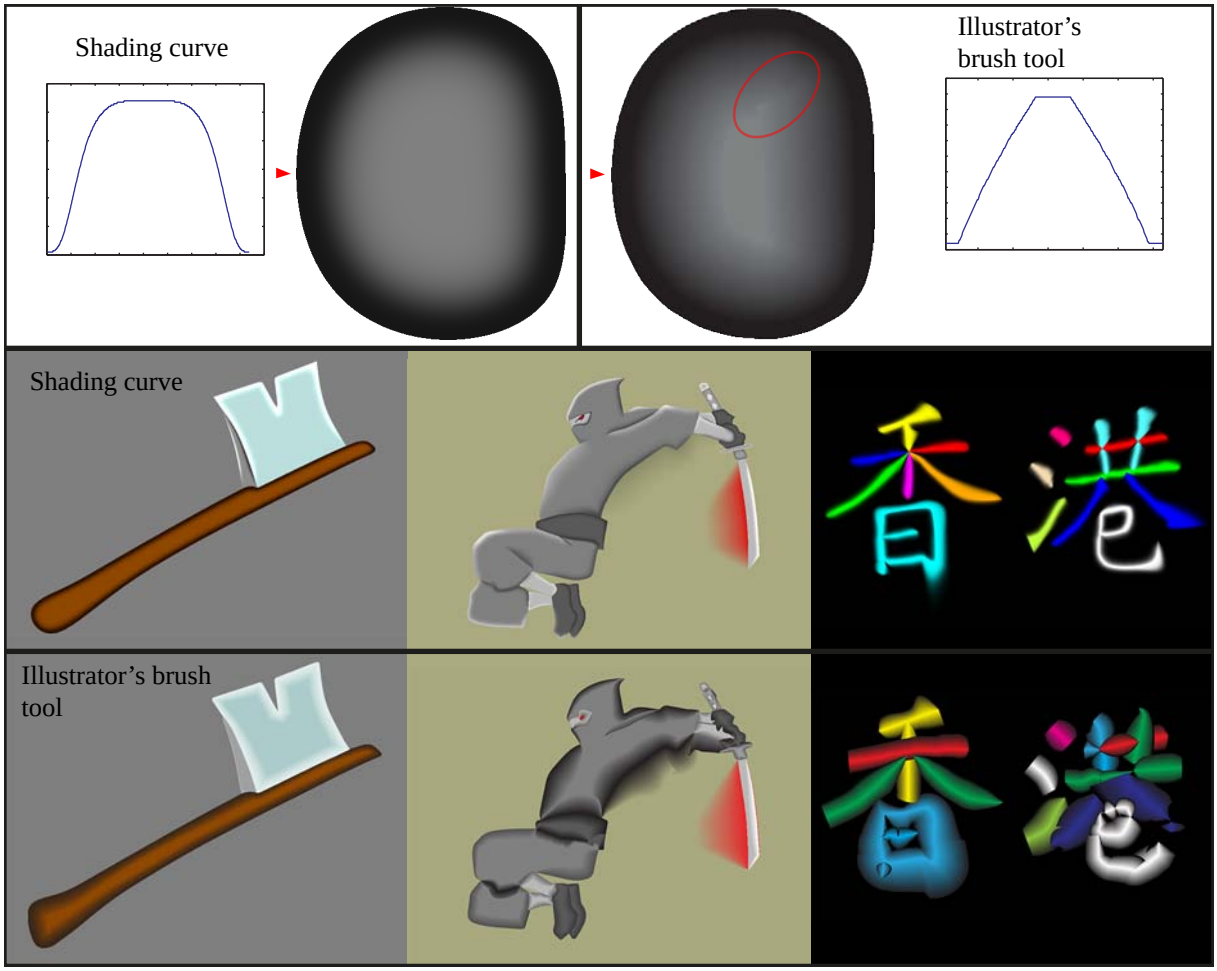


Figure 5.20: Visual comparisons between the shading curve and Illustrator's brush tool. Top: Two types of artefacts produced with Illustrator's tool are highlighted: the transition between the brush colouring and the underlying layer is not smooth, as shown in the cross section, and folds give rise to visual artefacts, as shown inside the red ellipse. The comparisons are relatively fair as a similar type of input is defined for both methods.

as indicated by the red ellipse in Figure 5.20. Finally, transitions between the colouring of the brush and the underlying layer are typically not smooth, as demonstrated by the cross section in the figure. Hence, there are three potential sources of crease artefacts in Illustrator's tool: the rendering procedure, the deformation procedure, and blending between layers.

In conclusion, Illustrator's brush tool is defined similarly to the shading curve by associating colour gradients across the brush's path in a given extent. However, by not creating smooth surfaces from this input makes it challenging to produce smooth colourings. Additionally, Illustrator's tool is restricted to a single colour profile along the brush's path. While rendering processes, using the univariate colour function directly, could be engineered, producing smooth gradients would be challenging. For these reasons, I do not see any practical advantages of rendering gradients along curves with direct reference to the univariate function. Creating a smooth bivariate colour function, as with our method and DCs, is more sensible because such functions are guaranteed to be smooth, are computationally efficient to render, and can be easily

defined and applied to the image.

Comparison with the gradient mesh tool

In contrast to the brush tool, the gradient mesh tool *does* support smooth colour gradients. Recall from Section 2.2.1 that the gradient mesh tool is used to manipulate colours and colour derivatives at control points of rectangular meshes. Mathematically, our method and the gradient mesh tool are similar since they both define bivariate colour functions defined by control meshes. Gradient meshes probably produces C^1 bicubic interpolatory spline surfaces from their rectangular meshes. Our method is topologically more flexible as it support triangular patches.

In practice, however, the two methods are fundamentally different: the shading curve is related to freeform curves and the gradient mesh is related to control meshes. While our method produces control meshes from the input curves, the user is not directly concerned with these meshes. By contrast, working with gradient meshes involves the creation and direct manipulation of control meshes. Such drawing is therefore similar to 3D design, where the designer directly manipulates vertices and edges of control meshes to define surfaces. Note that this difference is identical to the difference between gradient meshes and DCs, since DCs are also concerned with freeform curves. It is well known that, in the 2D setting, curves are easier to work with than control meshes[72].

While gradient meshes are more challenging to work with, they support more complex control over the colour function compared to the shading curve. Given a relatively dense mesh defined over a given domain, a shading curve will typically not provide as many colouring mechanisms over that domain. Thus, the effect of a shading curve is typically limited compared to the effect of a dense gradient mesh. Note that second-order DCs target this limitation of curve-based colour editing by the support for point constraints. However, such point constraints are limited compared to vertices of gradient meshes as they are restricted in terms of derivative control and local control. This is discussed further in Chapter 6.

These limitations echoes my ambition mentioned in Section 2.2.2: to create a superset framework that covers both DCs and gradient meshes. Thus, I do not argue that the shading curve or DCs should replace the gradient mesh tool. Instead, the two tools should be included in a suite of tools where the given drawing problem should dictate which tool to use. In the setting of shading object boundaries and defining main tonal areas, the shading curve would typically be preferable since it is easier to use. However, for vivid, intricate, colourings, the gradient mesh provides better control of the colour function. In a unified framework, I can imagine that the user is able swap between different modes (e.g. curve mode and mesh mode), while the underlying framework handles all modes simultaneously.

5.6 Discussion

The shading curve provides advantages that should be considered in the setting of rigorous colourings. In comparison with DCs, the shading profile provides advantages compared to DCs' boundary conditions by enabling local manipulation of the colour function. Incorporating

such local control in the setting of DC editing could strengthen this primitive. In comparison with the gradient mesh tool, the shading curve's underlying mesh structure, with support for triangular patches, is more flexible than the gradient mesh tool's strict restriction to rectangular grids. Relaxing this restriction can improve on many aspects of gradient mesh editing, including support for lower mesh densities and easier editing of complex objects.

On the other hand, the main limitation of the shading curve lies in its restricted capabilities in terms of colourings. Our approach is targeted towards layer-centric applications, where the colour associated with the curve is smoothly blended with an underlying image. By contrast, related primitives, in particular second-order DCs and gradient meshes, support more rigorous manipulations of the colour function.

In the next chapter, I present investigations to the above problems. Experiments indicate that defining the shading profile for DCs with its current definition is challenging. I therefore continue my study of applying subdivision, with its great advantage of freedom in mesh topology and local control, in the setting colour manipulation of vector graphics.

5.7 List of contributions

- A new vector primitive for defining smooth colour gradients out from curves.
- A new method to control lighting and shading effects in vector graphics using boundary and slope curves.
- Local control (via minimal support property) not demonstrated by previous DC methods.
- Interactive performance is more easily achieved with the proposed framework compared to similar approaches that are based on PDEs.

Chapter 6

Topologically unrestricted gradient meshes

This chapter presents research described in the following paper:

Topologically unrestricted gradient meshes

Recall from Section 5.6 that the shading curve is limited in terms of advanced colourings. In particular, DCs and gradient meshes both support certain aspects of colour manipulation that are not supported with the shading curve. On the other hand, they are not compatible with the shading curve: DCs do not support local control and gradient meshes do not support freeform curves as input.

Control meshes must be created to define gradient meshes from curves. This problem was presented in Chapter 4, where I discussed future work on creating quad meshes from curves. While such quad meshes could be created from curves, they would not be compatible with gradient meshes. This is because gradient meshes are restricted to rectangular mesh topology. This restriction also give rise to several other drawbacks such as high planning times and less freedom when manipulating objects with challenging shapes (Section 6.1).

To this end, I propose solutions that adapt the gradient mesh primitive to support control meshes of arbitrary *manifold* topology. By manifold mesh topology, I mean that any edge must be shared by either one or two incident faces and that the mesh is at least a 2-vertex-connected graph. The latter requirement means that if a vertex is removed, the mesh remains a connected graph (of course, it is assumed that the input mesh is a connected graph). The reason for requiring a manifold mesh topology is that the functions produced by the solutions presented in this chapter are 2D manifolds (that is, surfaces). Thus, there are no obvious reasons to why non-manifold meshes should be allowed as input. In the remainder of this chapter, I refer to 'arbitrary topology' as any mesh topology that is manifold.

In contrast to the shading curve framework, I do not consider the interaction between the solution and the user to a large extent. That is, I now assume that control meshes are already defined as input and I do not consider how they are created. I do, however, discuss ways for the user to interact with the solution to refine the coloured result. Then, the main challenge discussed in this chapter is to find a solution to the following interpolation problem: extend the behaviour of

the current gradient mesh tool in the regular setting to meshes of arbitrary topology.

To achieve a similar solution to previous solutions for defining colour gradients via control meshes, but with support for arbitrary topology, the following conditions must be achieved:

- **Condition 1**: the colour function interpolates the original control points in colour space.
- **Condition 2**: the colour function does not stray outside the given colour space.
- **Condition 3**: both geometry and colour function are smooth everywhere; that is, at least C^1 continuity of the surface, unless creases or discontinuities are specified.

I propose two solutions to this problem (Section 6.3). The conditions above are achieved by defining a set of special initial subdivision rules. I investigate two sets of such special rules (Sections 6.3.1 and 6.3.2). The refined mesh defined by these two solutions defines a Catmull-Clark surface that is guaranteed to satisfy the given conditions.

This study is, to my knowledge, the first analysis of the gradient mesh tool in the setting of arbitrary topology. In Section 6.1, I describe the background and the motivation for investigating this problem. In brief, the gradient mesh tool has been employed for both art creation and research. Its full potential, however, is hindered by the restriction to rectangular grids. I therefore argue that the relaxation of this restriction can improve on many aspects of gradient mesh editing and creation. As previously mentioned, gradient meshes that are unrestricted to topology can be used on (quad) meshes created from curves. In Section 6.5, I discuss future work that relates to this goal.

I present subdivision-based solutions to satisfy Conditions 1–3. However, other frameworks than subdivision are potential candidate solutions. Recent work on Hermite interpolation is especially intriguing (Section 6.2.2). Our approach has advantages over Hermite interpolation schemes, such as having more intuitive behaviour (Sections 6.2.2 and 6.4).

6.1 Background and motivation

Recall from Section 2.2.1 that the gradient mesh tool implemented in Adobe Illustrator and Corel CorelDRAW is a vector primitive defined by a rectangular grid. Colours and derivative constraints are defined on the control points in this grid. The pixels in the interior of this grid are interpolated smoothly between anchor points according to the derivative constraints. While the technical details of the gradient mesh tool have not been published, a reasonable assumption, based on the tool's behaviour, is that Ferguson patches are used for interpolating the interior pixels, as Ferguson's framework provides bi-cubic interpolatory C^1 patches where derivatives at each grid point can be edited. However, Illustrator performs various undisclosed simplification procedures for adding anchor points to the mesh, where grid rows and columns are hidden, making it challenging to infer the exact mesh construction. Additionally, first derivative constraints associated with anchor points do not seem to correspond directly to the related constraints in the Ferguson patches. Hence, it is difficult to accurately reproduce Illustrator's framework. In our experiments with Illustrator, we informally studied what Illustrator can produce and compared those results with what we can achieve.

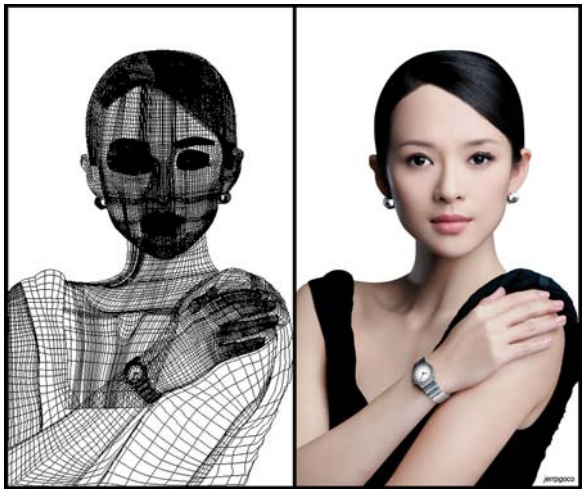


Figure 6.1: Illustrator gradient meshes created by professional designer © David Goco. Reported number of working hours: 120. Image from jerrp.deviantart.com.

The gradient mesh tool has been employed by both artists and researchers. In art, vector graphics creation with the gradient mesh tool is viewed as complicated and tedious[i]; mastering the tool is rewarding since complex imagery can be accomplished (for example, see Figure 6.1). Highly skilled and experienced artists have achieved photorealistic imagery[ii], which is otherwise unfeasible to achieve with the simpler linear gradient tool. For example, freelance artist Quarrie Franklin provides YouTube tutorials on most of his works[iii]. In research, the gradient mesh tool has inspired researchers to propose solutions to challenging colour-interpolation problems. In particular, multiple solutions to the problem of vectorising an input photograph to a vector-based representation have been proposed using gradient meshes (e.g. refs. 95;53).

Ferguson patches are parametric surfaces that are, in 2D, topologically restricted to rectangles and annuli (achieved by looping the rectangle). Figure 6.2(top) shows relatively simplistic topological layouts that cannot be achieved from a single rectangle. Additionally, Figure 6.2(bottom) shows that the rectangular definition of the control mesh can give rise to overly dense mesh regions. Such dense regions can be avoided with irregular mesh elements. These topological restrictions hinder the practical usability of the tool for multiple reasons. Below, I highlight local refinement, mesh density, and working with complex objects as aspects where the relaxation of this restriction can strengthen the usability of the gradient mesh tool.

– Rectangular grids are not locally refinable; that is, if a grid point is added, a whole row and column of new points must also be added. A control mesh of arbitrary topology, on the other hand, can be well defined with local densities of vertices and faces. A consequence of this restriction is that the artist needs to carefully predict the layout of the mesh before creating it.

– Rows and columns cannot merge, meaning that the density of the mesh is unnecessarily

[i] On the Internet, there are many blog posts and tutorials that describe the complications of the tool. For example, the following tutorial describes how to 'tame' the gradient mesh:
design.tutsplus.com/tutorials/quick-tip-how-to-tame-the-gradient-mesh–vector-3697

[ii] vectordiary.com/illustrator/top-10-masters-of-gradient-mesh

[iii] frankwyte81.deviantart.com

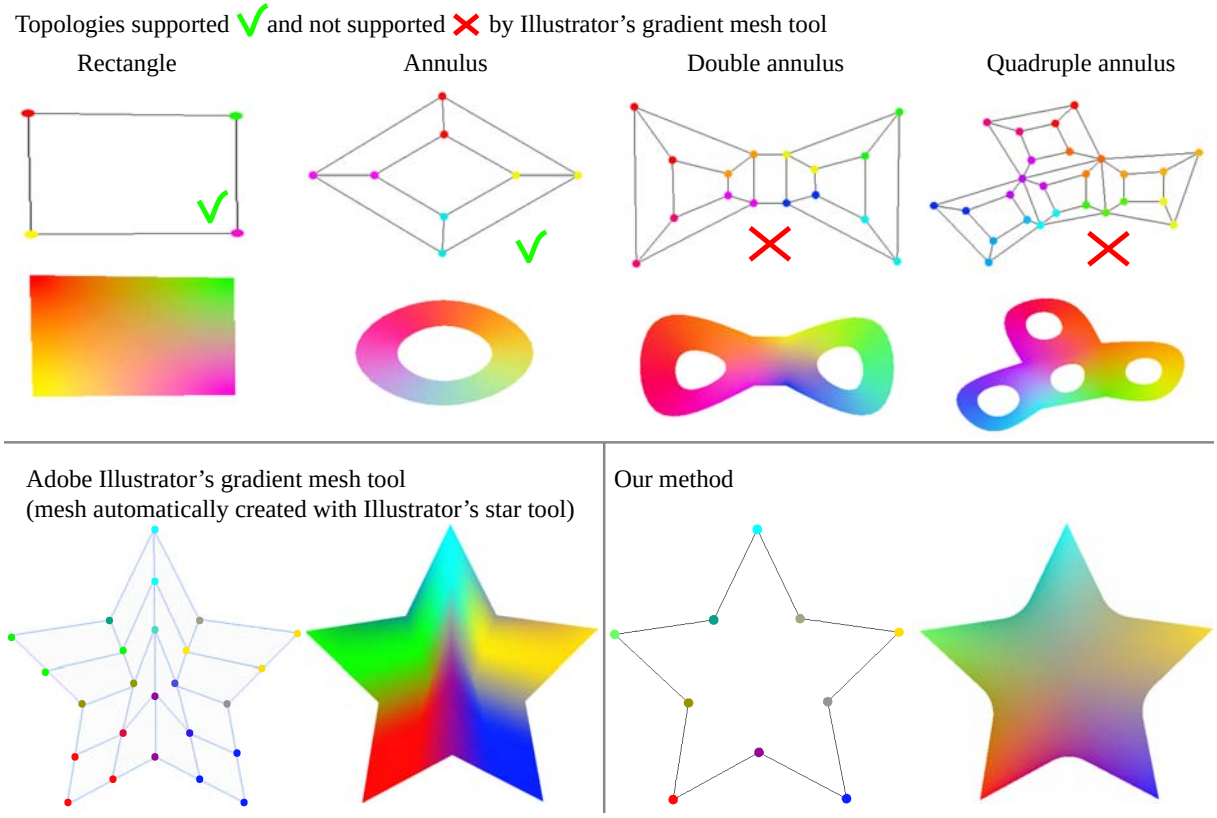


Figure 6.2: Illustrator's gradient mesh tool, used for vivid colourings in vector graphics, is constrained to rectangular and annular control meshes. Our method supports arbitrary mesh topologies. Top: Illustrator's tool supports rectangular and annular topologies only (annuli are achieved by looping rectangles). The colour images were rendered with our method, which *does* support any surface topology. Bottom: Illustrator's tool is restricted to rectangular meshes and therefore does not support irregular mesh elements, such as this decagon. By contrast, such irregular elements are supported with our method.

high around features not aligned with the grid. For example, shadows and specular highlights must typically be aligned with the grid in feasible gradient mesh editing. A consequence of this restriction is that editing of the mesh becomes tedious and time consuming, especially where rows or columns are clamped together. By contrast, meshing approaches not restricted to regular grids can support directional constraints so that meshes locally align to given features.

– Complex objects, such as human faces, require flexibility of the topology of the underlying mesh. Multiple patches are typically used to get around this limitation with Illustrator's gradient mesh tool (Figure 6.1). Features such as eyes and mouths are modelled as separate patches and then manually stitched with neighbour regions. Although layering is a helpful concept to separate parts of objects and sources of light and shade, the topological layout of the object boundary and its interior features should not be a neccessary source of layering.

An interesting analogy to our contribution is Pixar's move from NURBS to subdivision for

character animation in 1998 (ref. 23). As with Ferguson patches, NURBS are parametric surfaces with the rectangular grid constraint. Consequently, NURBS models typically consist of a patchwork of NURBS surfaces. The move to surfaces defined via arbitrary topology has shown to be extremely successful; Pixar's feature films from *Toy Story 2* (1999) have been created exclusively with subdivision surfaces. Local refinement, single-surface models, and planning times are highlighted as the major benefits[23]. Note that, even with these benefits, other industries, such as car manufacturing, still use NURBS. The reasons for this are that there is no exact backward compatibility to NURBS and that industrial models are typically more regular in their shape than natural objects usually depicted in animation. However, I believe for drawing and animation, topological freedom greatly bolsters the usability of such technology, as demonstrated by Pixar.

Diffusion curves

DCs were initially proposed by Orzan and colleagues as a more intuitive alternative to gradient meshes for drawing smooth colourings for vector graphics[73]. The motivational discussion of Orzan et al. therefore contains similar arguments as above. They argue (ref. 73, Section 1): 'While more powerful than simple gradients (or "ramps"), gradient meshes still suffer from some limitations. A significant one is that the topological constraints imposed by the mesh lattice give rise to an over complete representation that becomes increasingly inefficient and difficult to create and manipulate.'

While DCs provide a simplified type of input with freeform curves, as discussed in Chapter 5, DCs are limited compared to gradient meshes. Compared to gradient-mesh-based methods, like our method, DCs have the following disadvantages:

- The PDE conditions are global boundary conditions; it is very difficult or even impossible to achieve the same degree of local editing as achieved by Ferguson patches and our method. This also stems from the fact that DCs are evaluated via PDEs and not B-spline-based methods with the minimal support property.

- A large linear system needs to be solved for each frame which is computationally costly. Interactive performance has been demonstrated to be hard to achieve, as mentioned in Section 5.5. A particular disadvantage is the scale operation (that is, zooming and retargeting), where the system needs to be solved again, even for pixels outside the viewport. For mesh-based solutions, zooming only requires changes of the camera parameters; no recomputation is required.

- Diffusion curves only support freeform curves and points as input and the framework is not easily extendable to polygonal domains and control meshes. Thus, there is no backward compatibility to the gradient mesh tool.

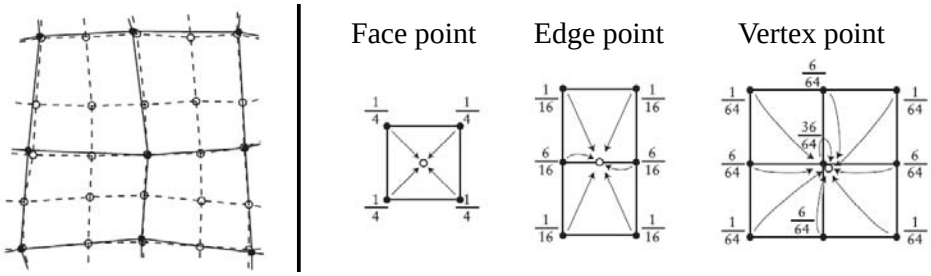


Figure 6.3: Catmull-Clark subdivision in the regular setting. Left: a mesh (solid dots and solid lines) that has been refined (open dots and dashed lines). Right: subdivision rules for face, edge, and vertex points. The fractions represent the weights of a related vertex used to compute the new vertex.

6.2 Related solutions

While the given problem has not been presented in the literature, the problem is closely related to interpolation over control meshes of arbitrary topology. In such settings, solutions concerning subdivision and Hermite interpolation are well-known to support smooth interpolation. In this section, I discuss potential solutions with current methods.

6.2.1 Subdivision

The main advantage of subdivision over related approaches, such as NURBS, is the support for arbitrary topology of the control mesh. Additionally, subdivision benefits from many properties of NURBS like local support. In Chapter 5, I discussed why such local support is important in the setting of colour manipulation. Since our proposed method extends the Catmull-Clark subdivision scheme, I briefly describe its subdivision rules.

Figure 6.3 illustrates Catmull-Clark subdivision on a regular mesh. There are three types of rules and, consequently, three types of new vertices. (1) A vertex is introduced close to a vertex in the original mesh. These new points are known as *vertex points*. (2) A vertex is introduced in the centroid of each face. These are known as *face points*. (3) A vertex is introduced close to the midpoint of each edge. These are known as *edge points*.

Figure 6.3 shows the subdivision rules for the three types of points in the regular setting. The valency of a vertex is the number of edges incident to it. The valency of a face is the number of vertices defining that face. Vertices and faces of valency four are regular and are irregular for all other valencies. For irregular vertices, special rules for vertex points are used and for irregular faces, the face point is, as with regular faces, placed on the centroid of the face. A vertex with valency n can be related with the following subdivision rule[12]:

$$v_s = \frac{n-2}{n} v_o + \frac{1}{n^2} \sum_j e_j + \frac{1}{n^2} \sum_j f_j; \tag{6.1}$$

where v_s is the subdivided vertex point and v_o is its related original vertex. Additionally, e_j are all vertices, in the original mesh, in the 1-ring-vertex neighbourhood of the original vertex (that

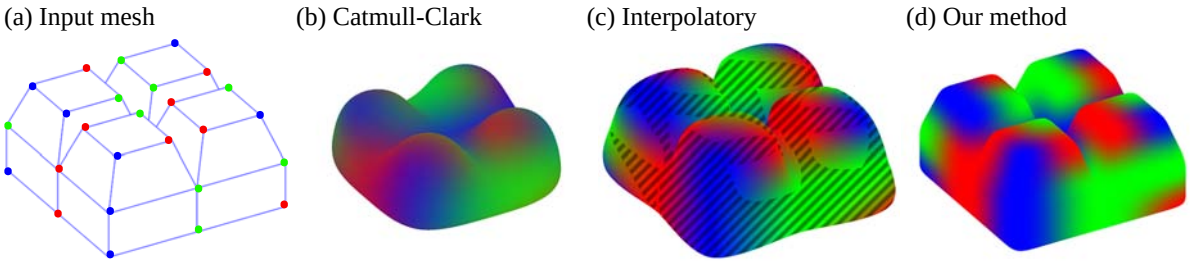


Figure 6.4: Demonstration of colour interpolation using three subdivision schemes. (a) A brick-shaped control mesh with colours at control points. (b) The approximating Catmull-Clark scheme is not suitable for this problem as input colours are washed out because they are not interpolated. (c) Interpolatory subdivision schemes do preserve colours, but the interpolating surface can stray outside the colour gamut (the surface in (c) was rendered with the method in ref. 56). Colours outside the gamut are shaded in (c) and have had to be clipped in colour space. (d) Our method, which treats the colour coordinates differently from the geometric coordinates, both interpolates the original colours and defines a surface that is inside the colour gamut.

is, all vertices one edge from the vertex), and f_j are all face points, in the new mesh, of the 1-ring-face neighbourhood of the vertex point v_s (that is, all faces adjacent to v_s).

The Catmull-Clark subdivision scheme is a *stationary* scheme, meaning that the same subdivision rules are applied in each subdivision step. An important aspect of subdivision is to understand the scheme's *limit surface*; that is, the surface defined by the refined mesh at the infinite subdivision level. In the regular setting (all mesh elements are regular), the limit surface of the Catmull-Clark scheme (simply known as a Catmull-Clark surface) has been proven[12] to be equivalent to a bi-cubic C^2 B-spline surface.

The Catmull-Clark surface has interesting properties at irregular mesh elements. Vertex points preserve their valency in each subdivision step. Similarly, face points will be defined with the same valency of their original face. Thus, an n-valency face relates to an n-valency vertex (its face point) in the new mesh. Irregular mesh elements are therefore preserved in each subdivision step as irregular vertices and the new mesh will be defined as a pure quad mesh, regardless of the topology of the original mesh. Furthermore, the *limit point*[iv] of an irregular vertex is a singularity on the limit surface, surrounded by regular spline surface. The continuity at such a singularity is C^1. The continuity of the limit surface everywhere else is C^2.

The Catmull-Clark scheme has been extended to be suitable for commercial settings, like character animation[23]. The most significant alteration is the sharpness feature, introduced by Pixar's research group[23], to model sharp C^0 creases and semi-sharp edges. A sharp-crease rule forces the surface to interpolate the edge, producing a C^0 crease in the surface points related to the edge. A semi-sharp crease is achieved by performing the sharp-crease rule of the edge a given number of subdivision steps and then switch to standard Catmull-Clark rules to produce a surface that is smooth, but closer to the edge than the standard Catmull-Clark surface.

While Catmull-Clark subdivision supports arbitrary topology, it is not suitable for gradient mesh editing. This is because the Catmull-Clark scheme is an approximating scheme, meaning that

[iv] Limit point of a vertex: the point on the limit surface related to the vertex.

its limit surface does not interpolate the original control points. This is undesirable in vector-based mesh editing where the artist typically expects selected colours to match their related rasterised pixels (Figure 6.4(b)). An alternative could be interpolatory subdivision schemes, which *do* interpolate original control points. However, they do not guarantee that the surface is defined inside the convex hull of the control points (Figure 6.4(c)). Thus, the surface might be defined outside the given colour space, resulting in clipping which produces undesirable sharp C^0 creases in colour space. To deal with these issues, our solutions treat the colour coordinates differently from the geometric coordinates (Figure 6.4(d)).

Additionally, Ferguson patches support manipulation of derivative constraints at control points. Such derivative constraints are not easily achieved in the setting of arbitrary meshes since it is challenging to define a single parameterisation of the underlying surface, especially at irregular mesh elements. Thus, standard subdivision schemes do not natively support such derivative constraints.

6.2.2 Hermite interpolation

Advances in interpolation theory make closed-form interpolation potentially suitable for colour editing. Interpolation schemes have evolved to support arbitrary simple polygons, achieved by generalising barycentric coordinates[104;32]. Such coordinates have been adopted to approximate popular polynomial functions, including harmonic functions, interpolating values only[44], and biharmonic functions, interpolating both values and derivative constraints[106]. To achieve smooth colour interpolation over arbitrary meshes, each face can be separately interpolated whilst ensuring that the gradient defined by the derivative constraints at each vertex is continuous. Polynomial interpolation with value and derivative constraints is known as *Hermite interpolation.*

Given arbitrary input constraints, Hermite interpolation does not achieve Conditions 1–3. While colours are interpolated, arbitrary derivative constraints can force the surface in colour space to stray outside the colour space, thus placing the surface at undefined coordinates. Such behaviour is demonstrated in Figure 6.5(middle), where the magnitude of the first derivative constraints are naïvely increased. This is obviously unwanted and justifies the derivative constraints of second-order DCs which are set to zero. Moreover, recall that Ferguson patches are also associated with first derivative constraints and that these are used to render Illustrator's gradient mesh primitive. Figure 6.5(bottom) illustrates that over-saturation does *not* occur when the derivatives are increased with Illustrator's tool. It is likely that these constraints only refer to the geometric dimension of the surface and that the derivatives in colour space are fixed to zero.

An additional problem with the specification of derivatives is how to define them at irregular vertices. In gradient meshes of arbitrary topology, derivatives are associated with edges so that they emulate the behaviour of gradient meshes at irregular elements. That is, we assume that users should be able to manipulate the gradient along all edges. Other types of derivative constraints could be investigated, for example, by requiring only two linearly independent vectors to define the tangent plane at the vertex. However, it is unclear how to orient the two vectors at n-valency vertices, whilst supporting 'meaningful' behaviour of the derivative constraints in relation to the surrounding mesh (and surface) topology.

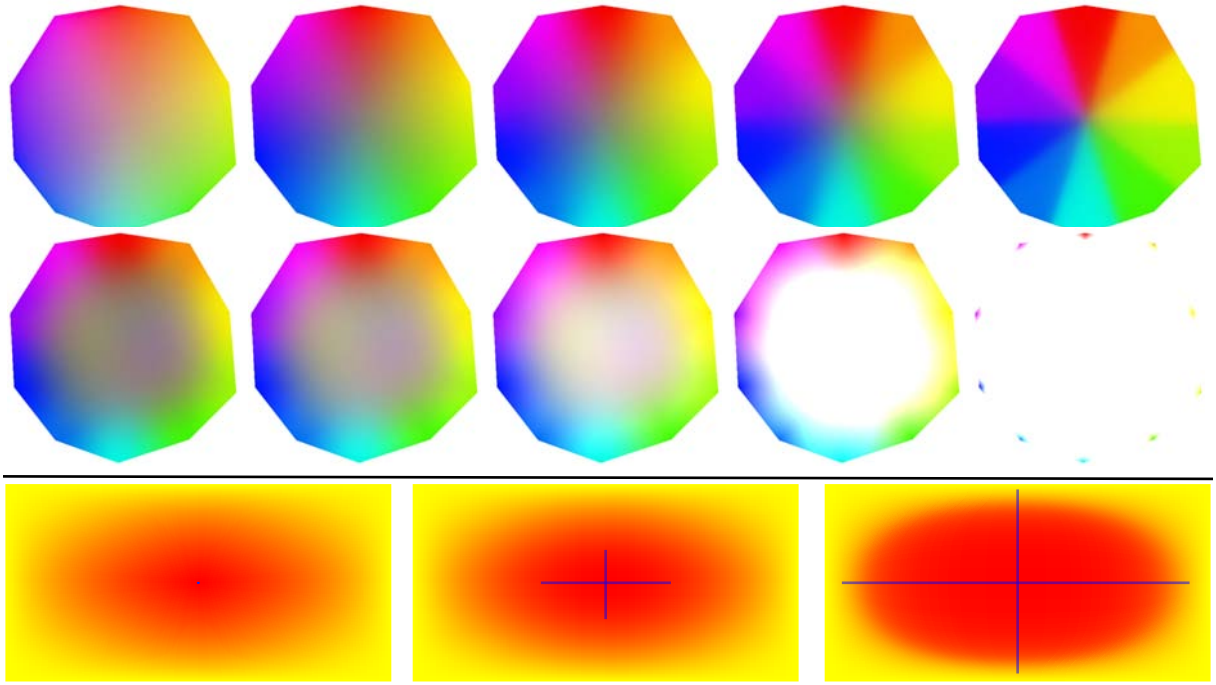


Figure 6.5: Comparing the mechanisms to manipulate first derivatives at the vertices with our method (top) and a Hermite interpolation scheme[57] (middle). Bottom: adjusting derivatives with Illustrator's gradient mesh tool (blue lines). Note that Illustrator's method does not produce over-saturation artefacts. Its behaviour indicates that derivatives in colour space are kept fixed and the derivative constraints only relate to the geometric coordinates.

On the other hand, derivative constraints associated with all edges can give rise to multiple gradients and tangent spaces that are not equivalent to a plane (for example, with more than two directionally independent vectors defined by the derivative constraints). These two aspects are challenging to satisfy when dealing with surfaces (which are 2D manifolds where each point on a surface is homeomorphic to the 2D Euclidean space; its tangent space must be a 2D Euclidean space).

Our method, described in the next section, provides an alternative view on gradient meshes' derivative constraints: instead of manipulating first derivatives, which could be a vague and non-intuitive concept to a designer, the user manipulates the propagation of a selected colour in its local neighbourhood. To this end, we chose to rename 'derivative constraints' to *directional colour weights*. In the regular setting, derivative constraints can be satisfied and directional colour weights can therefore be viewed as weights that can be used to manipulate derivatives. In the irregular setting, however, directional colour weights can support a type of behaviour that cannot be directly associated with 'regular' derivative constraints. This behaviour motivates the use of subdivision, which support local control, rather than Hermite interpolation, which is restricted to value and derivative constraints.

Recently, Li and colleagues[57] demonstrated a simplification procedure of gradient meshes with cubic mean value coordinates (CMVCs) that satisfy Hermite boundary constraints when the

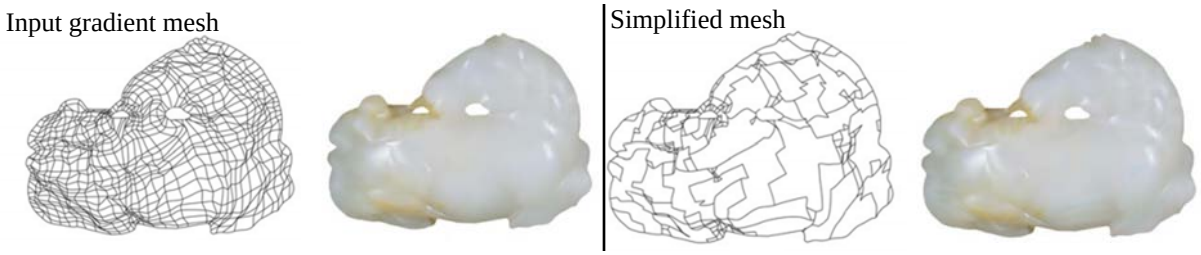


Figure 6.6: Simplification of gradient meshes with the method of Li et al. Images from ref. 57.

boundary is piece-wise linear. The coordinates are defined as:

$$\hat{f}[\mathbf{p}] = \sum_i a_i[\mathbf{p}] f_i + \sum_{i,s} b_i^s[\mathbf{p}] f_i^s + \sum_{i,s} c_i^s[\mathbf{p}] h_i^s, \tag{6.2}$$

where $s \in \{+, -\}$, $\hat{f}$ is the interpolated value at $\mathbf{p}$, f_i is the value of the i^{th} vertex of the input closed polygon, and f_i^s and h_i^s are the magnitudes of the first derivatives of f_i along the edges and along the normal components, respectively. The generalised barycentric coordinates, CMVCs, are given by a_i, b_i, and c_i.

Simplification of a given gradient mesh was demonstrated by Li et al. by iteratively removing vertices of the lowest (visual) cost (Figure 6.6). Smoothness across edges and vertices is achieved with their method by projecting the gradients from the original image to the remaining edges. Without an original image, however, it is not clear how to define the derivatives, f_i^s and h_i^s, as described above. That is, users are not able to adjust the gradients of the original image and the approach is therefore more suitable for simplification and compression purposes.

6.3 Topologically unrestricted gradient meshes

I now describe our solutions. The goal is to define a surface that emulates the behaviour of Illustrator's gradient mesh tool in the regular setting and to extend this behaviour for meshes of arbitrary topology. Recall from the introduction to this chapter that to achieve this goal, the surface should interpolate the original colours of the control points (Condition 1), it should not stray outside the given colour space (Condition 2), and it should be smooth everywhere (Condition 3). Additionally, the local propagation of a colour should be influenced by the directional colour weights.

To achieve our goal, we decided to split the treatment of geometric and colour coordinates. That is, separate subdivision schemes are employed for the colour and geometric dimensions, with the constraint that the two subdivision schemes generate the same topology. More specifically, the geometric coordinates can be produced in several ways, according to user input and algorithmic decisions (Section 6.3.3). Then, the colour coordinates are produced according to a set of fixed rules.

To explain the genesis of the method, I start with the following observation. When creating a cubic B-spline or Catmull-Clark surface, the surface is guaranteed to interpolate the colour of a vertex if the surrounding ring of vertices all have the same colour. This is most easily seen in

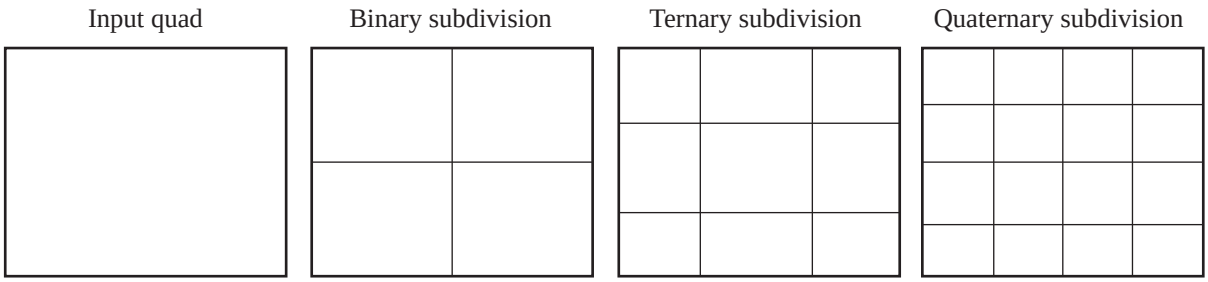


Figure 6.7: Different subdivision schemes can give rise to different topologies of the refined mesh.

the curve case (Figure 6.8), where placing vertices of equal colour either side of a vertex forces the resulting curve to interpolate that colour.

This observation led to the proposed method. That is: for every vertex in the original mesh, a ring of vertices around it that have the same colour is introduced (Figure 6.9(a)). This forces interpolation of the original vertex (Condition 1) and forces the final surface to lie within the valid colour space (Condition 2). Thus, initial special subdivision steps are performed to ensure that Conditions 1 and 2 are satisfied. The refined mesh defines a Catmull-Clark surface which guarantees at least C^1 continuity everywhere (Condition 3). Creases and semi-sharp features, which are supported using Pixar's sharpness features[23], can be specified by the user.

We are now left with two questions: at what geometric location should we place these new vertices?; and does this method produce visually-compelling output? The first of these questions, geometric location, is addressed in the following subsections, providing a control mechanism similar to that of Ferguson patches. The second question, visual quality, is addressed in Section 6.4.

As mentioned above, separate subdivision rules are applied to the geometric and colour coordinates, with the constraint that the two subdivisions are concerned with the same topology of the refined mesh. There are many options regarding the topology of this refined mesh. In subdivision, schemes can be classified according to the topological aspects of the scheme. For example, when subdividing a quad, a binary scheme produces a 2×2 grid, a ternary scheme produces a 3×3 grid, a quaternary scheme a 4×4 grid, and so on (Figure 6.7).

We investigated several subdivision schemes. In Section 6.3.1, I describe a solution which employs a single initial step of ternary subdivision. I demonstrate that such a ternary scheme is particularly well suited to this problem as it produces the minimal number of points required to satisfy Conditions 1–3. However, a challenge with such an approach is that it must preserve the valency of each face in the refined mesh. A non-standard approach to defining subdivision rules for new faces had to be invented. In contrast, binary subdivision produces a pure quad mesh. I demonstrate in Section 6.3.2 that a sensible refinement can be achieved in this setting by modifying the standard Catmull-Clark rules.

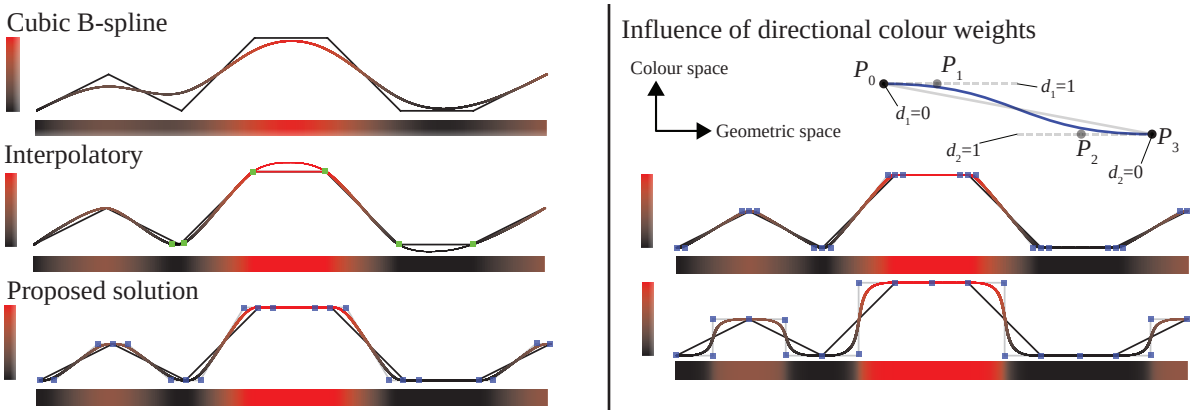


Figure 6.8: Colouring a horizontal bar. Left: comparisons between three subdivision approaches. The blue squares (bottom) show the control points defined in the special initial subdivision step of the proposed solution. The alternative subdivision schemes do not satisfy all conditions: The uniform B-spline curve (top) does not interpolate the original colours (breach of Condition 1). The interpolatory curve (middle) strays outside the colour space, as highlighted by the green squares (breach of Condition 2). Right: influence of the directional colour weights. Smaller weights (top colouring; weights set to 0.1) produce curves that approximate the control polygon more closely. Larger weights (bottom colouring; weights set to 1) result in larger influence of the given colour.

6.3.1 Initial refinement with ternary subdivision

The solutions for bivariate (2D manifolds – surfaces) colour interpolation are based on solutions defined for univariate (1D manifolds – curves) colour interpolation. For each solution presented in this subsection and in Section 6.3.2, I first present the solution in the univariate setting and then generalise that solution for bivariate colour interpolation.

A univariate solution

Given a control polygon, the solution defines a new set of control points that define a cubic B-spline curve satisfying the given conditions. The initial step of subdivision is special: it is a single ternary subdivision step of the input control polygon.

New vertex points are defined at the same coordinates, both in colour and geometry, as the original vertex. That is: the original vertices do not move and are included in the refined polygon.

New edge points are defined differently for colour and geometric coordinates. Along every edge, two new edge points are defined. Let P_0 and P_3 be the original vertices defining the edge. Each vertex has associated directional colour weights. P_0 is associated with the directional colour weight d_1; d_2 is associated with P_3. Directional colour weights are scalars in the range $[0, 2]$ (see Section 6.3.3 for discussion of weights above 1). In geometric space, the new edge

points, P_1 and P_2, are defined as:

$$P_1 = (1 - d_1)P_0 + d_1(P_0 + P_3)/2;$$
$$P_2 = (1 - d_2)P_3 + d_2(P_0 + P_3)/2.$$

The midpoint of the edge is used for the linear combination to make the directional weights compatible with the bivariate solution.

In colour space, P_1 is set equal to P_0 and P_2 is set equal to P_3. Thus, each original point is surrounded by points of the same colour.

The refined control polygon defines a cubic B-spline curve that satisfies the given conditions: the input colours are interpolated, whilst ensuring that the colour function is inside the colour space, by forcing the first derivative of the colour function at the original vertices to zero.

Figure 6.8 compares our solution with interpolatory and approximating subdivision and demonstrates the effect of the directional colour weights.

Finally, our solution produces results that look similar to Illustrator's curve editor, as derivatives at the original control points are manipulated with the new edge points P_1 and P_2. Via the multi-resolution feature supported by subdivision, where control points can be edited at any subdivision level, P_1 and P_2 can be manually edited in geometric or colour spaces by the artist. Editing these points is analogous to editing derivatives in Illustrator's curve editor, which most likely employs Ferguson curves.

A bivariate solution

The univariate solution above is generalised to the bivariate setting. The main challenge in this setting is to define subdivision rules that provide both natural-looking colourings and that satisfy the given conditions.

The solution first performs a single modified ternary subdivision of the input control mesh. This is followed by standard Catmull-Clark subdivision. The initial ternary subdivision step produces a new set of vertices and faces related with the original control mesh. Recall from Section 6.2.1 that new vertices are classified as vertex, edge, and face points.

The ternary subdivision step subdivides each face of n-valency into a smaller face of n-valency, surrounded by quadrilaterals. Each edge of the original face is associated with three new quadrilaterals on each of its two sides. This subdivision step is shown in Figure 6.9(d).

The ternary subdivision procedure is particularly well suited to this problem: All subdivided points are associated with a single original control point. Consequently, the colours of these points are all set to the same colour as their related original control point (Figure 6.9(a)). This step assigns colours to all vertices in the new mesh.

With the colour coordinates defined, the remaining, geometric, coordinates are described next. New edge and vertex points are defined as in the univariate solution.

New face points are defined as follows. Given a control point V and one of its incident faces, let the midpoints of the two associated original edges be E_1 and E_2. The related directional colour

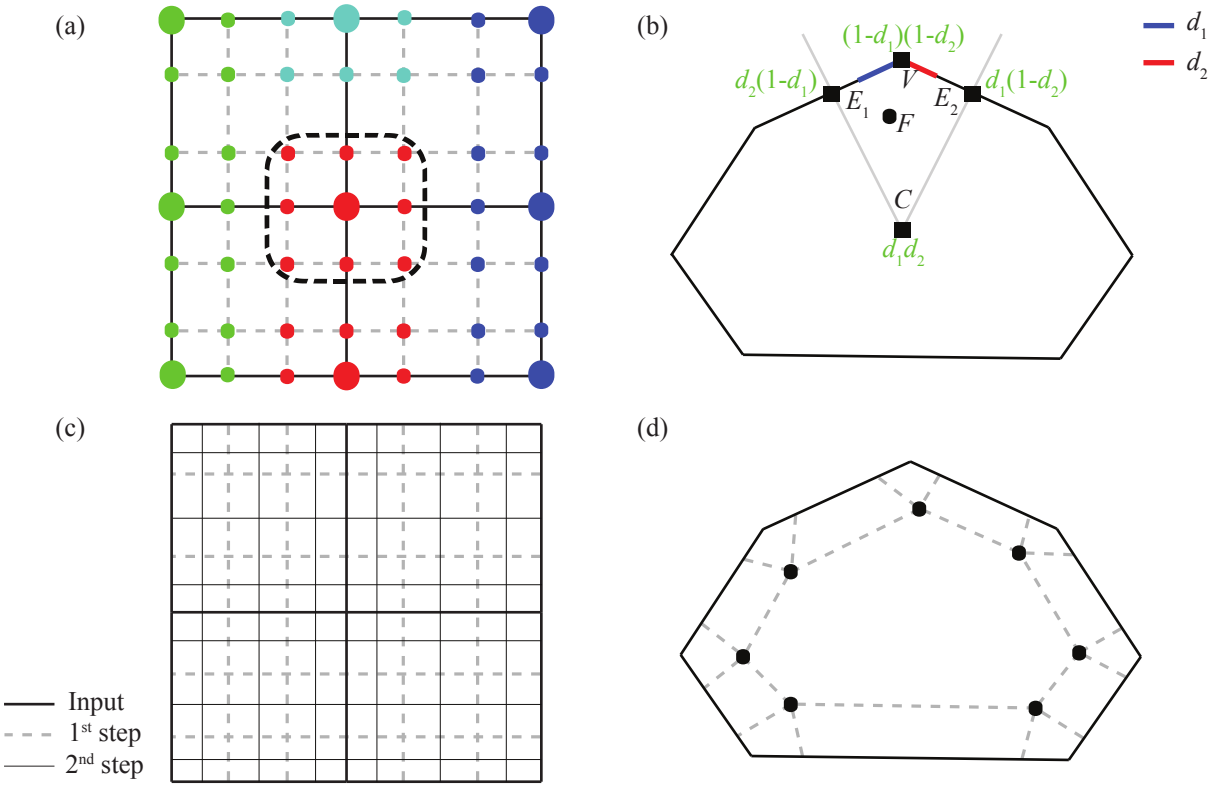


Figure 6.9: The proposed solution performs a single step of modified linear ternary subdivision followed by standard Catmull-Clark subdivision. (a) In colour space, all new edge, vertex and face points associated with an original point (large disks) are assigned the colour of that point. Thus, the one-ring of points around an original point are all defined with the same colour. (b) The geometric location of a new face point (black disk) is defined by bilinear interpolation over a quadrilateral defined by the original vertex V, the midpoint of the two related edges $E_{\{1,2\}}$, and the centroid of the face C. The weights of the bilinear interpolation (green) are defined by the related directional weights, d_1 and d_2. (c) The mesh configuration of a 2×2 grid after two steps of subdivision. (d) The first step produces a centre face of the same valency as the original face, surrounded by quadrilaterals.

weight from V to E_1 is d_1 and the directional colour weight from V to E_2 is d_2. Additionally, let C be the centroid of the face. Then, the geometric location of the new face point F is defined via bilinear interpolation (Figure 6.9(b)):

$$\begin{aligned} F =& (1-d_1)(1-d_2)V + d_1 d_2 C + \\ & d_2(1-d_1)E_1 + d_1(1-d_2)E_2. \end{aligned} \tag{6.3}$$

After this single special ternary step, standard Catmull-Clark rules are used for all subsequent subdivision steps to produce the final surface. Figure 6.9(c) shows the configuration of a 2×2 rectangular grid after one step of ternary subdivision and one additional step of Catmull-Clark subdivision.

Figure 6.10 shows results for two types of meshes with various settings of the directional colour

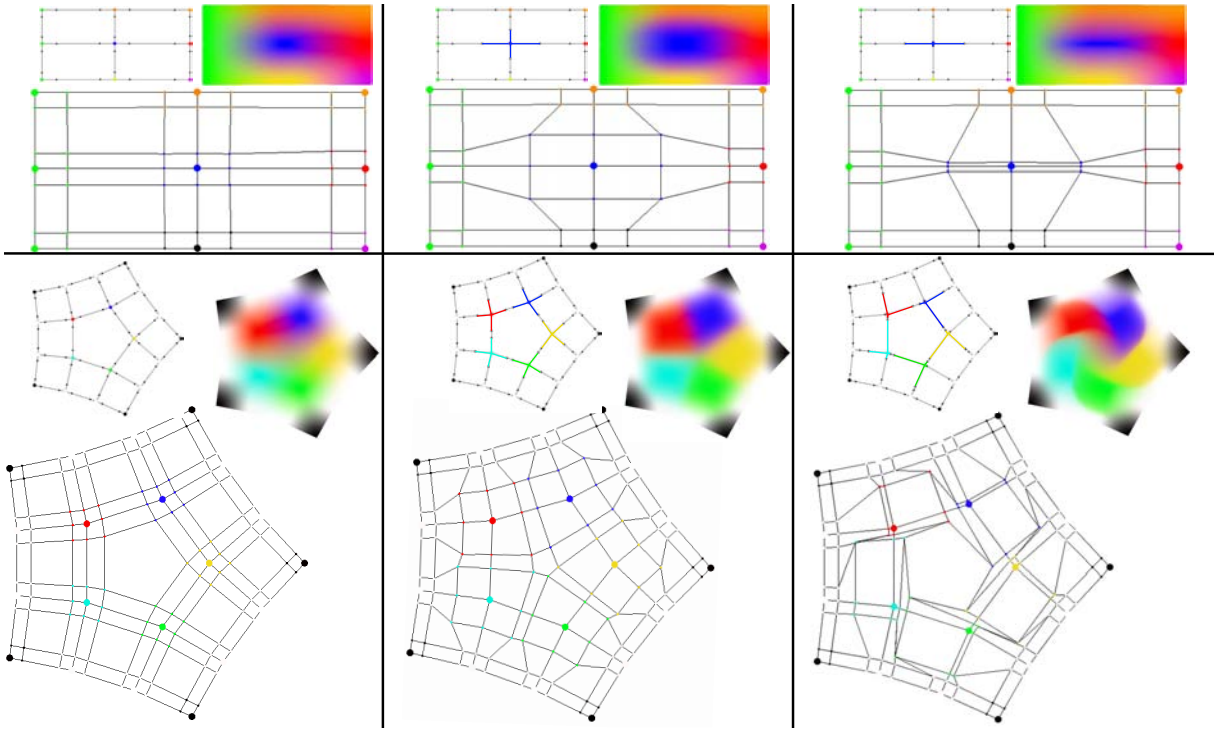


Figure 6.10: Illustrations of the control meshes computed after the first step of subdivision with the proposed ternary subdivision step. Original control points are marked by larger disks. In the input meshes (top left for each result), new edge points after the first step are marked by smaller black disks. Notice how the directional colour weights (adjustments have been highlighted with coloured lines) influence the local propagation of the colours. The coloured results have been further subdivided two times with Catmull-Clark rules and the surfaces were rendered with OpenGL.

weights. These results indicate that the method produces intuitive colourings according to the given input. The behaviour of the solution is discussed in more detail in Section 6.4.

6.3.2 Initial refinement with binary subdivision

An alternative scheme using binary subdivision is now presented. An advantage of binary subdivision over the ternary solution is that it produces a pure quad mesh, thus replacing relatively large central faces with smaller quads. This was, in fact, the first solution that was tried because it is the most obvious solution. Its drawbacks led to the development of the ternary solution.

In order to satisfy Conditions 1–3, two binary subdivision steps must be performed. A single step of binary subdivision will not produce a unique 1-ring neighbourhood for the original control points, which is required to interpolate original control points with Catmull-Clark subdivision. That is, original control points share points in their 1-ring neighbourhood after a single step of subdivision. Two steps of binary subdivision are required for the 1-ring neighbourhoods to be unique. Thus, modifications to incorporate the directional colour weights are performed after the second step of subdivision. Consequently, control points that are not defined in the 1-ring neighbourhoods of original control points must also be dealt with. Such additional points

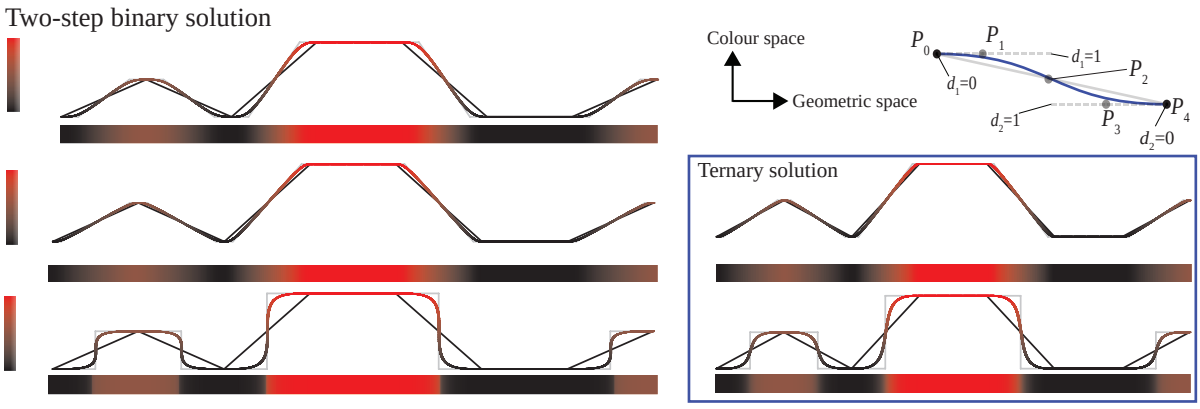


Figure 6.11: Colouring a horizontal bar with the two-step binary solution. The weights are, from top to bottom: 0.4, 0.1, and 1. Compared to the ternary solution, it creates an additional point, P_2, on the midpoint of the edges. The visual results are similar to the ternary solution. However, the ternary solution creates marginally smoother results for larger weights. This difference is seen more clearly in Figure 6.14.

are not defined in the ternary solution and represent the added complexity to this binary subdivision solution.

The treatment of these additional points complicates the presentation of the solution as more special rules must be defined. More specifically, edge, vertex, and face points are treated differently in the second step depending on their definition in the first step of subdivision. I therefore adjust the terminology of new vertex points in the second step: they are referred to with the notation <type>-vertex. For example, an edge-vertex point has been subdivided as an edge point in the first step, and then as a vertex point in the second step. Thus, a vertex-vertex point can be related to an original control point.

Note that this solution can alternatively be viewed as a single quaternary subdivision of the original control polygon. As I have found the two-step treatment more notationally manageable, I have chosen to describe the method as a two-step binary subdivision solution.

A univariate solution

The first step of subdivision is perfomed with linear binary subdivision of the input control polygon: new vertex points are defined at the same coordinates, both in colour and geometry, as the original vertex. New edge points are defined at the midpoint of the edge.

The special rules, incorporating the directional colour weights, are now defined in the second step of subdivision. New vertex-vertex points are defined at the same coordinates as the original vertex.

The coordinates of all new points created along a given original edge are now described. Let P_0 and P_4 be the vertex-vertex points defining the edge, P_2 be the edge-vertex point, and P_1 and P_3 be the new edge points. Each vertex-vertex point has associated directional colour weights. P_0 is associated with the directional colour weight d_1; d_2 is associated with P_4. Directional colour

weights are scalars in the range $[0, 1]$.

In geometric space, these points are defined as:

$$P_1 = (1 - d_1)P_0 + d_1(P_0 + P_4)/2;$$
$$P_2 = (P_0 + P_4)/2;$$
$$P_3 = (1 - d_2)P_4 + d_2(P_0 + P_4)/2.$$

Hence, the edge-vertex points are defined with the same coordinates, at the midpoint of the original edge, as its related point defined in the first step.

In colour space, P_1 is set equal to P_0, and P_3 is set equal to P_4. P_2 is the average of P_0 and P_4.

The refined control polygon defines a cubic B-spline curve that satisfies the given conditions. Figure 6.11 illustrates these subdivision rules and demonstrates the effect of the directional colour weights.

A bivariate solution

As with the univariate solution, the first subdivision step is defined as a linear binary subdivision of the input control mesh. That is: new vertex points are defined at the same position as its related vertex, new edge points are defined at the midpoint of the edge, and new face points are defined at the centroid of the face.

The colour coordinates of the subdivided points in the second step are now described. As previously mentioned, the 1-ring of new points surrounding a vertex-vertex point is unique. The colours of the points in this 1-ring neighbourhood are set to the colour of the original control point related to the vertex-vertex point.

The additional set of points not related to these 1-ring neighbourhoods must also be dealt with. Figure 6.12 illustrates these rules in colour space. Edge-vertex points and face-vertex points are dealt with separately. The colour of an edge-vertex point is defined as the same colour as its related vertex. The colours of new face-vertex points are defined with standard Catmull-Clark rules (Equation 6.1). Additionally, new edge points not related to the 1-ring neighbourhoods are defined between edge-vertex points and face-vertex points. Such a new edge point is given the same colour as its related edge-vertex point. Finally, all new face points are defined in the 1-ring neighbourhoods of vertex-vertex points and have therefore been already defined above.

A similar treatment is performed in the geometric space. Vertex-vertex points (P_1 and P_4 in the univariate case) and edge-vertex points (P_2 in the univariate case) are defined as in the univariate solution. There are two types of new edge points: those related to a vertex-vertex point and an edge-vertex point, and those related to a face-vertex point and an edge-vertex point. The former type of edge point corresponds to P_1 and P_3 defined in the univariate solution and are therefore defined in the same way. The latter type is illustrated in Figure 6.12(d). Such points are related to two vertex-vertex points and two directional colour weights. The new edge point is then defined as a linear combination of the edge, using the average of the two directional colour weights as the weight for the linear combination.

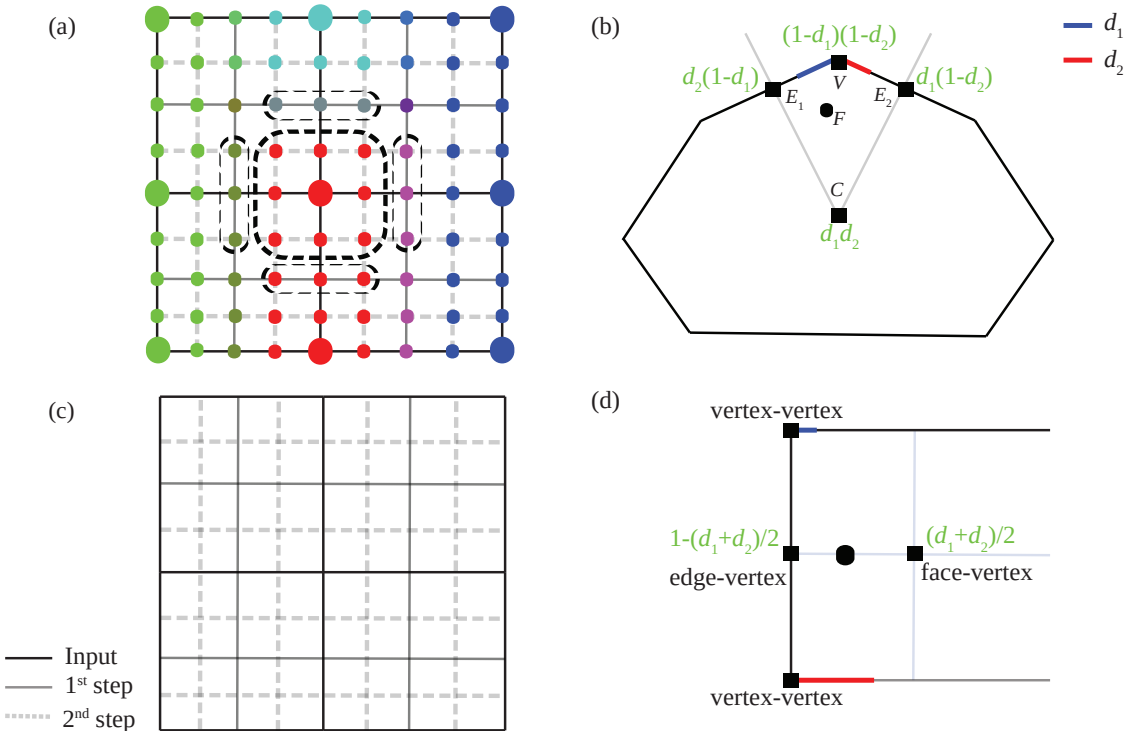


Figure 6.12: Solution using two steps of initial binary subdivision followed by standard Catmull-Clark subdivision. (a) In colour space, the colours of points in the 1-ring neighbourhood of original vertices (large disks) are set to the colour of the original vertex. Additionally, the colours of edge and edge-vertex points in the second step not in 1-ring neighbourhoods of original vertices are set to the average colour of the two related original vertices. (b) A new face point (black disk) is defined as the ternary solution. (c) Mesh configuration of (a) after two steps of subdivision. (d) Subdivision weights (green) of new edge points related to an edge-vertex point and a face-vertex point.

Finally, new face points are defined. The first step defines a pure quad mesh. Each quad in this subdivided mesh exactly corresponds to a quad used to define new face points in the ternary solution. Thus, given a quad defined with the following points: V (original vertex), E_1 and E_2 (midpoints of the original edges and new edge points defined in the first step), and C (the centroid of the face and a face point defined in the first step); then, the new face point F is defined as:

$$\begin{aligned} F =&(1-d_1)(1-d_2)V + d_1 d_2 C + \\ &d_2(1-d_1)E_1 + d_1(1-d_2)E_2. \end{aligned} \tag{6.4}$$

Standard Catmull-Clark rules are used for the subsequent iterations to produce the final surface. Figure 6.12(c) shows the configuration of a 2×2 rectangular grid after two steps of subdivision with this solution. Figure 6.13 shows results produced with this solution for some configurations of the directional colours weights.

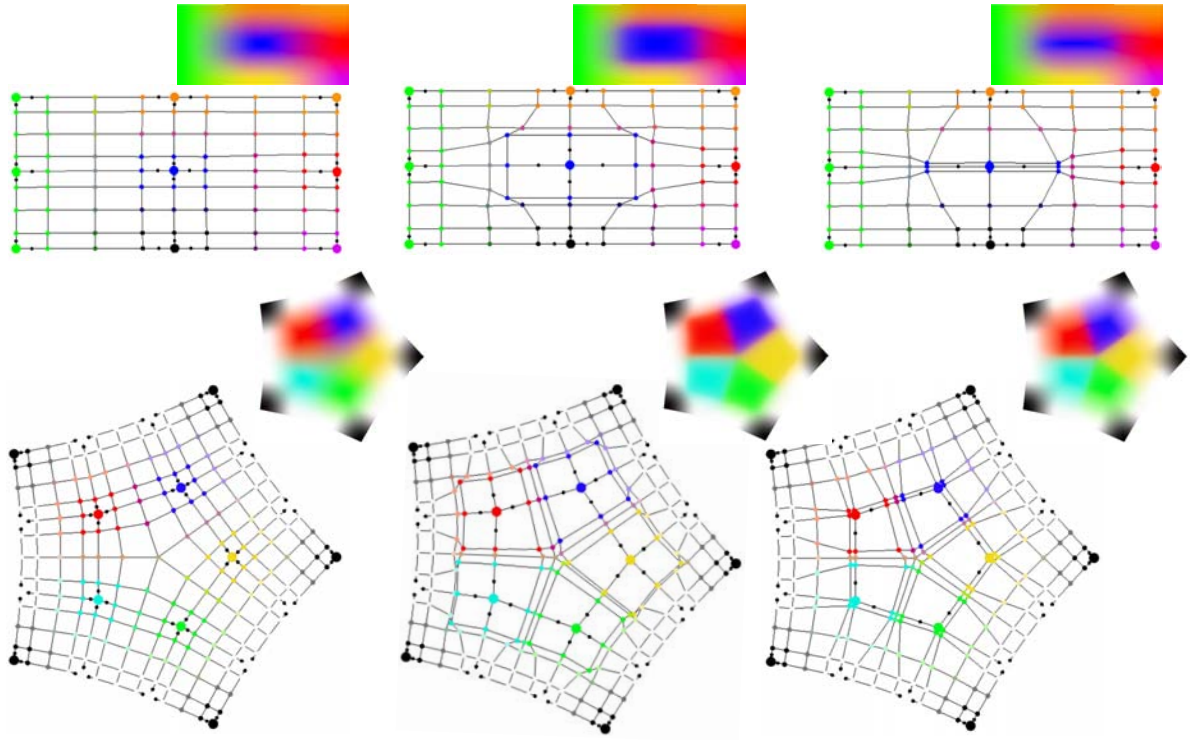


Figure 6.13: Illustrations of the control meshes computed after two steps of subdivision with the alternative binary solution. Original control points are illustrated with larger disks. Smaller black disks relate to directional colour weights.

Comparisons with the ternary solution

Figure 6.14 shows comparisons between the binary and the ternary solutions. As the ternary solution produces marginally smoother results for larger weights, this solution is deemed better. Consequently, we provided no special modifications to the new points created in the first step and folding can therefore occur for directional colour weights above 1. Such large weights are directly supported with the ternary solution, as shown in the figure. The binary solution was first developed because it was not clear how to define new face points with a ternary approach. By contrast, a two-step binary approach simply means that the Catmull-Clark rules would need modification after the second step to satisfy Conditions 1 and 2. The binary approach therefore seemed more feasible in the initial stages of the project.

Using the experience from the binary solution, various rules for new face points for a single-step of ternary subdivision were experimented with. The binary solution inspired us to opt for the more 'local' approach of bilinear interpolation of the directional colour weights in a local quadrilateral. Thus, traces of the binary solution can be seen in the ternary approach: the set of quadrilaterals used to define new face points in the ternary solution corresponds to the new faces created in the first step of binary subdivision. The advantage of the ternary approach is that these quads are only used intermediately to define new face points. By contrast, the binary approach includes these quads in the final mesh simply because they are created in the first step and, as a consequence, are used to define new faces in the second step. The binary solution therefore introduces an unnecessary set of faces and vertices to achieve Conditions 1 and 2.

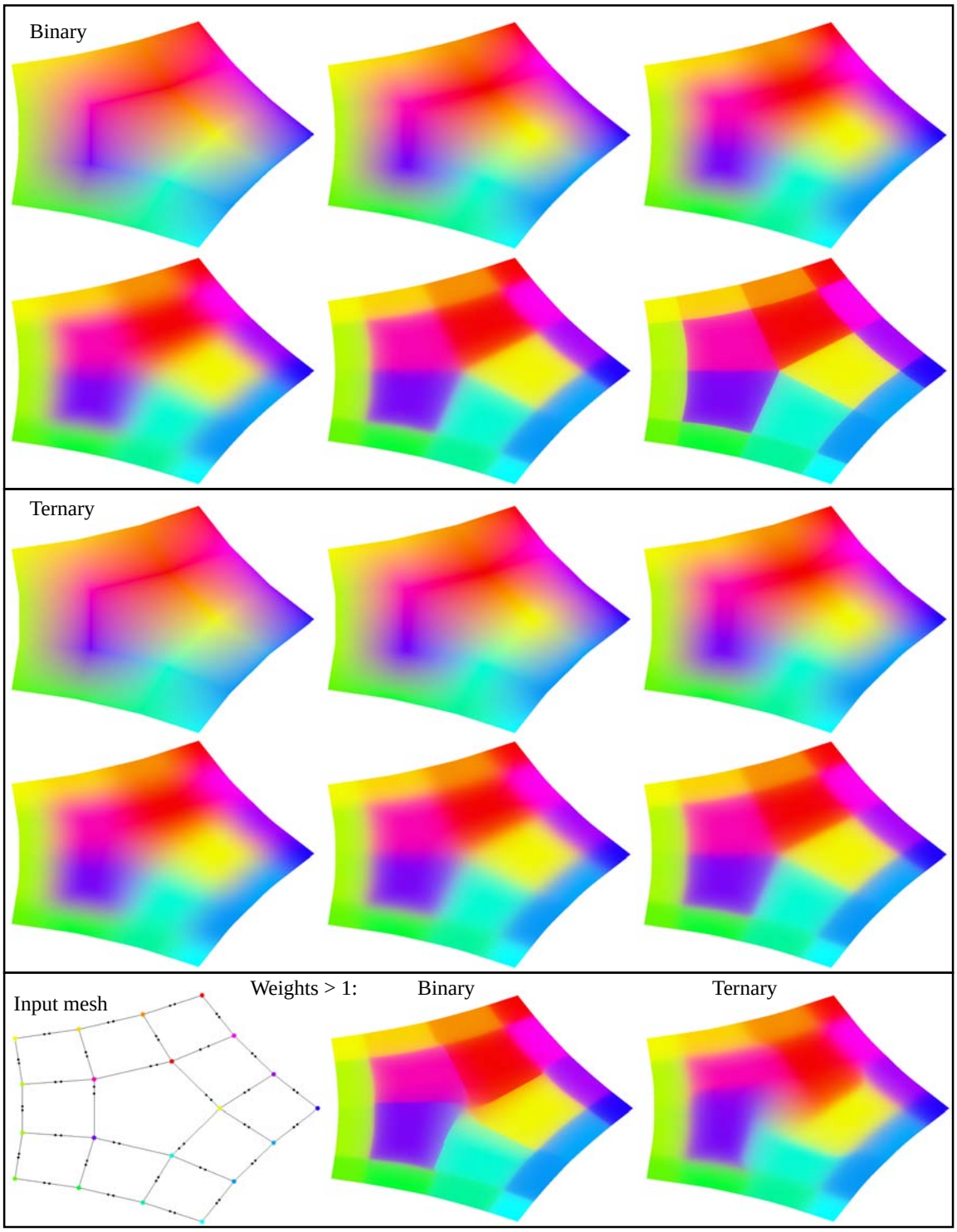


Figure 6.14: Comparisons between the binary and ternary solutions. In the top two rectangles, the directional colour weights are globally increased from 0 to 1 from top left to bottom right. Notice how the results for the ternary solution are visually smoother compared to the binary solution, especially for larger weights. In the bottom rectangle, several directional colour weights are above 1, producing folding artefacts with the binary solution. By contrast, the ternary solution produces smooth results.

As the ternary approach is deemed better than the two-step binary solution, I will, in the remainder of this chapter, only refer to the ternary solution (i.e. 'our solution' refers to the ternary solution).

6.3.3 Discussion

There are several aspects of our solution that are open for discussion. I address alternative definitions of face points, the various options available in the geometric space, and folding issues.

On definition of face points

There are many viable options to defining weights of the vertices of original faces for new face points. The proposed solution, bilinear interpolation on a smaller quad, is relatively local. Alternative approaches using pre-defined weights for all of the vertices of the face, similar to Catmull-Clark rules, can be considered. A challenge, however, is to incorporate the directional colour weights. Moreover, the solution defines relatively large central polygons. While such a weighting scheme would be suboptimal for most applications, larger central polygons are sensible for this particular colouring problem. The reason for this is that the geometric and colour coordinates are separated, meaning that using a more global weighting scheme can force a given colour to have influence outside the local neighbourhood of the given control point. Thus, the colour function can behave unnaturally in relation to the directional colour weights with such global schemes.

There are multiple possibilities to defining the quadrilaterals used for the bilinear interpolation of directional colour weights for new face points. For example, the scheme can be reconfigured to use the entire edge, and not midpoints, to produce the same results. Midpoints are used to define sensible local influence of the directional colour weights. More specifically, if all directional colour weights are set in the range of $[0, 1]$, colourings in the local neighbourhood of the control points will not conflict with their adjacent control points. While a directional colour weight d can be set in the range $[1, 2]$ to extend the influence further than the midpoint, the corresponding directional colour weight along the given edge must be set to maximum $2 - d$ in order to avoid folding (Figure 6.15). This restriction can be enforced in the user interface.

On geometric coordinates

There are several options available for the subdivision rules in the geometric space. Specifically, two questions should be considered: should the geometric coordinates of the original control points be interpolated or approximated?; and how should boundary vertices be treated?

Our solution does not interpolate the *geometric* coordinates of the input control points. In contrast, Ferguson patches *do* interpolate both colour and geometric coordinates. We did not notice visual differences between the two approaches (Section 6.4), probably owing to the fact that the limit point L corresponding to a vertex V in the input mesh lies close to V due to the

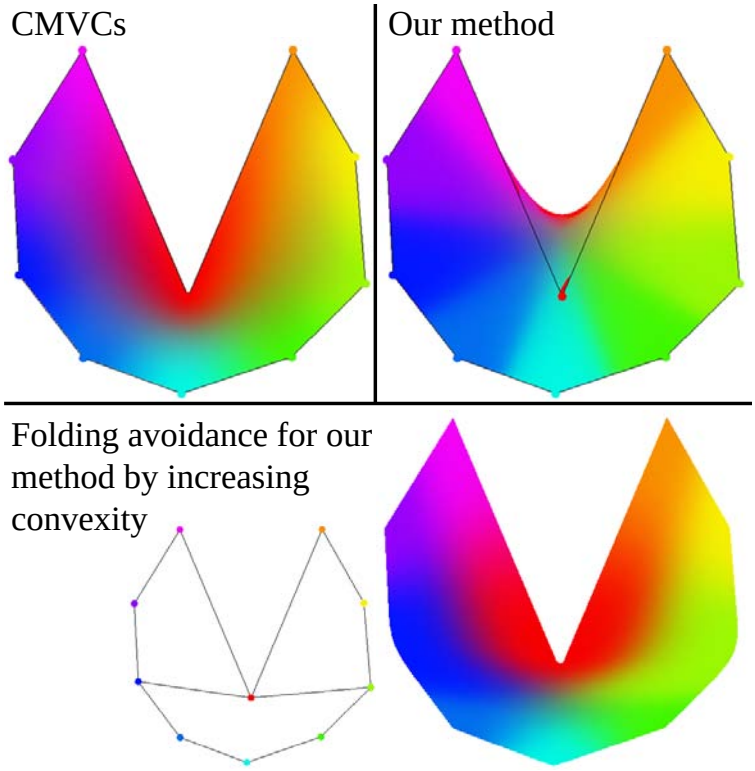


Figure 6.15: Concave polygons can produce folds with spline-based interpolants, such as our method; directional colour weights have been increased to better highlight this issue. Interpolation schemes with support for concave polygons, such as CMVCs, do not create such folds.

initial ternary step and the local behaviour of Catmull-Clark subdivision. However, the initial ternary step could be tuned to ensure that geometric coordinates are interpolated; that is: $L = V$ for all vertices. This could be achieved by using the limit projection given in Section 3.2 in ref. 60, which imposes a single linear condition on the 1-ring of V after the initial ternary step. Moreover, since these 1-rings do not overlap, the linear conditions for all V do not form a system; they are independent of each other and can thus be easily incorporated.

The boundary subdivision rules in the geometric space should be carefully considered depending on the given application (the boundary rules give rise to a 1D manifold at the boundary of the (open) surface). The reason for this is that boundary rules influence not only the propagation of colours but also the shape of the boundary. For example, artists familiar with Ferguson curves and patches, where one adjusts the shape of the boundary with derivative constraints, would prefer the boundary vertices to be interpolated. On the other hand, artists more familiar with B-spline curves and surfaces are used to adjusting the shape with control points of the control polygon or mesh. As the rules of the boundary vertices can be trivially adjusted to support both options, we chose to subdivide the boundary with similar rules as those used for the interior vertices; that is, original colours are propagated to the new vertices (see the boundary vertices of the mesh in Figure 6.12(a)).

Folding problems

Folds of the spline surfaces can create C^0 artefacts (Figure 6.15). Such folding can only occur for concave polygons and faces. We note that this is a general problem when dealing with spline surfaces, such as Ferguson patches, in the 2D image domain and is not an artefact produced by our method specifically. There are several ways of dealing with this issue. Illustrator's gradient mesh tool seems to apply a trimming procedure that clips the folds. To avoid folding, input control points can be adjusted so that the centroid of the polygon is defined inside that polygon. A third approach is to refine the polygon to a mesh of convex faces. Finally, we note that such folding problems are not encountered with interpolation schemes that support concave polygons, such as CMVCs (Figure 6.15(left)).

6.4 Results and comparisons

In this section, I discuss results, performance, and multi-resolution editing. Furthermore, I compare our results against Illustrator's gradient mesh tool and CMVCs. I stress that the goal of our method is to provide a similar behaviour to Illustrator's gradient mesh tool in the regular setting and to extend this behaviour for meshes of arbitrary topology. In my comparisons, I show relatively simple meshes, because these allow me to demonstrate larger spatial influence of the colour of each control point, to best analyse the behaviour our method. I have found that visual differences are best distinguished with primary colours. To demonstrate the artistic applicability of the method, I also provide some complex examples. The control meshes in these examples were created, manually, with SketchUp. Finally, the colour function has been defined in the RGB colour space when comparing with other methods. Otherwise, I have used the LAB colour space for interpolating colours. I prefer the LAB space because it provides uniform transitions between colours without introducing additional hues, which, in my opinion, provides visually more pleasing colour transitions[v].

Results

Results are shown in the figures in this chapter and in Appendix B. These visualisations indicate that our solution produces intuitive results according to the directional colour weights. By intuitive, I mean results that one could expect from Illustrator's tool (which I compare againts later in this section). Additionally, Figure 6.4 demonstrates that our solution is also well defined for control meshes in 3D.

Performance

We achieve interactive performance with our naïve C++ OpenGL single-core CPU implementation. Tested with an Intel SU4100 1.3 GHz CPU, the software produced three levels of sub-

[v] See the following blog for visual comparisons between multiple colour spaces:
howaboutanorange.com/blog/2011/08/10/color_interpolation

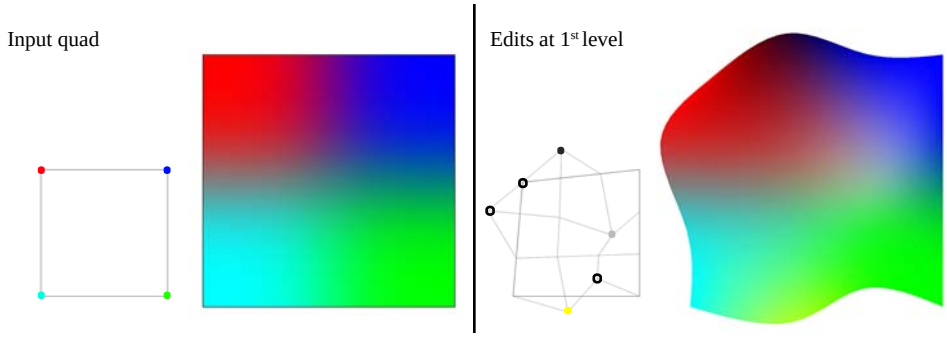


Figure 6.16: Multi-resolution editing after the first subdivision step. Right mesh: coloured disks represent vertices that have been moved and recoloured; black rings represent vertices that have been moved but not recoloured.

division in the range of 10–390 milliseconds (ms) for all results shown in this chapter and in Appendix B (for example, separate times for the 1st;2nd;3rd steps for: Figure 6.10(bottom): 1;5;20 ms, Figure 6.18(our method; bottom): 20;70;300 ms, Figure 6.19: 8;33;120 ms). The number of subdivision levels required for smooth-looking results depends on the spatial layout of the input mesh, where sparser meshes might require additional levels. In practice, we have found that three levels are typically sufficient. We note that the performance can be further improved, at least for subdivision levels using Catmull-Clark rules, with a GPU implementation. From the reported timings on such implementations[69], we should expect a performance boost of at least one order of magnitude.

Multi-resolution editing

A particular advantage of subdivision, as demonstrated in surface modelling for animation[23], is the native support for multi-resolution editing[113]. That is, meshes can be edited after each step of subdivision as the mesh is progressively refined. Thus, meshes at the coarsest level can be relatively sparse, whilst still supporting complex colourings in the interior of the object. Figures 6.16 and 6.19 demonstrate manipulation of the shape of the boundary and colour refinements with multi-resolution editing. Note that we only investigated multi-resolution editing for refinement purposes. As the subsequent subdivision steps with Catmull-Clark rules are approximating, I do not demonstrate interpolation of selected colours and applying local weights of such refinement, although such a feature could be implemented relatively easily (by colouring the 1-ring of the refined control point after two subdivision steps with its colour).

Comparisons with Illustrator's gradient mesh tool

Informal comparisons with the gradient mesh tool are provided in Figures 6.18 and 6.17 and in Appendix B. These comparisons indicate that our solution behaves similarly to Illustrator's tool in the regular setting. While the two methods produce visually similar results, they are not directly compatible. In the regular setting, our method produces C^2 bi-cubic B-spline surfaces, whereas Ferguson patches are defined as C^1 bi-cubic interpolatory spline surfaces. In practice,

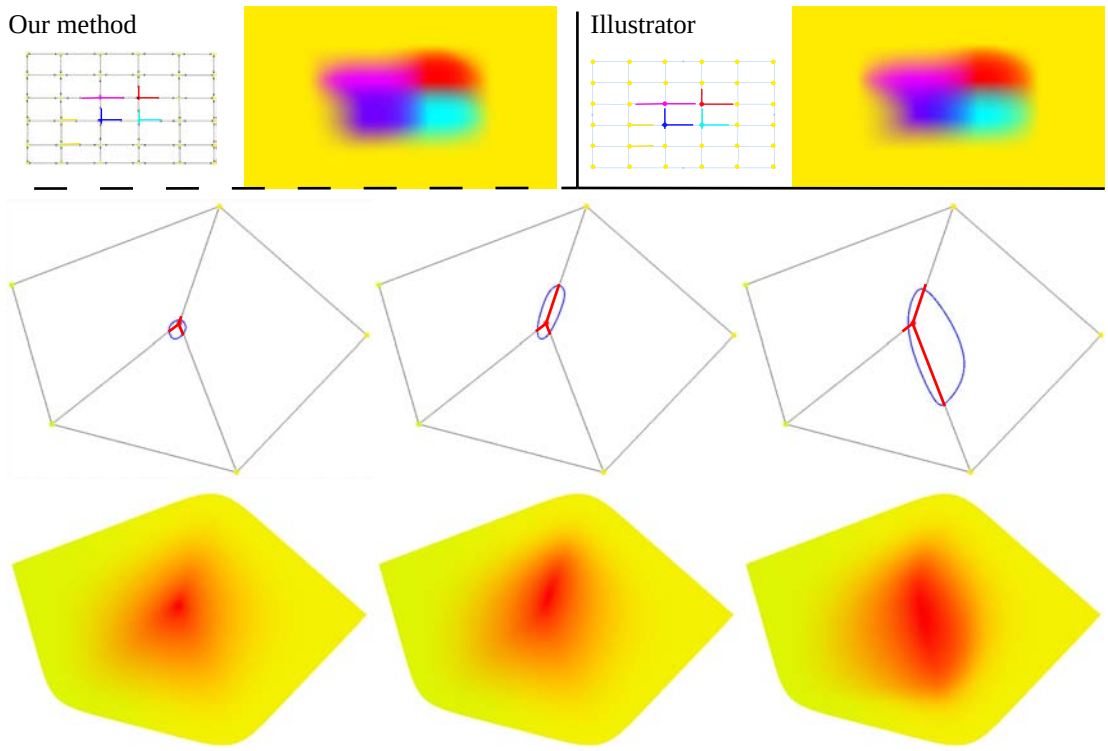


Figure 6.17: Behaviour in relation to derivative constraints with our method and with Illustrator's gradient mesh tool. Manipulated derivative constraints have been highlighted with coloured lines. The bottom three examples illustrate the behaviour of the directional colour weights at an irregular vertex. The propagation of the red colour relates to its 1-ring neighbourhood, which is influenced by the directional colour weights. The blue curves are the cubic B-spline curves defined by the vertices of the 1-ring neighbourhoods and they therefore relate to the shape of the red colour in that local neighbourhood.

this difference in continuity for regular regions does not provide any particular advantage to the artist. The advantages of our method lie instead in its ability to handle arbitrary mesh topology.

This advantage is demonstrated in Figures 6.18(bottom) and 6.19: meshes of arbitrary topology can be defined with varying density of the mesh according to the underlying features of the object. Thus, this contribution improves on the restrictions mentioned in Section 6.1 by supporting more flexible topologies of the mesh. I note that the improvement of our approach lies in the fact that meshes of arbitrary topology are not *unnecessarily* dense. Dense meshes *are* required in order to capture finer image details (Figure 6.19). I believe our subdivision-based approach can be advantageous over alternative approaches in such settings, since the meshes can be initially sparsely defined. Then, the finer details can be manipulated by multi-resolution colour adjustments. In contrast to the current approach with Illustrator (that is, moving points on a fixed dense mesh), the spatial layout of the mesh control points can be manipulated at the first two levels; thus, colour is the only parameter that should be manipulated at the finer levels. Such an approach can improve on creation and planning times, as demonstrated by Pixar[23].

Figure 6.17 and Appendix B illustrate the influence of the directional colour weights and compares it with Illustrator's gradient mesh tool. Note that Illustrator's tool does not give rise to the

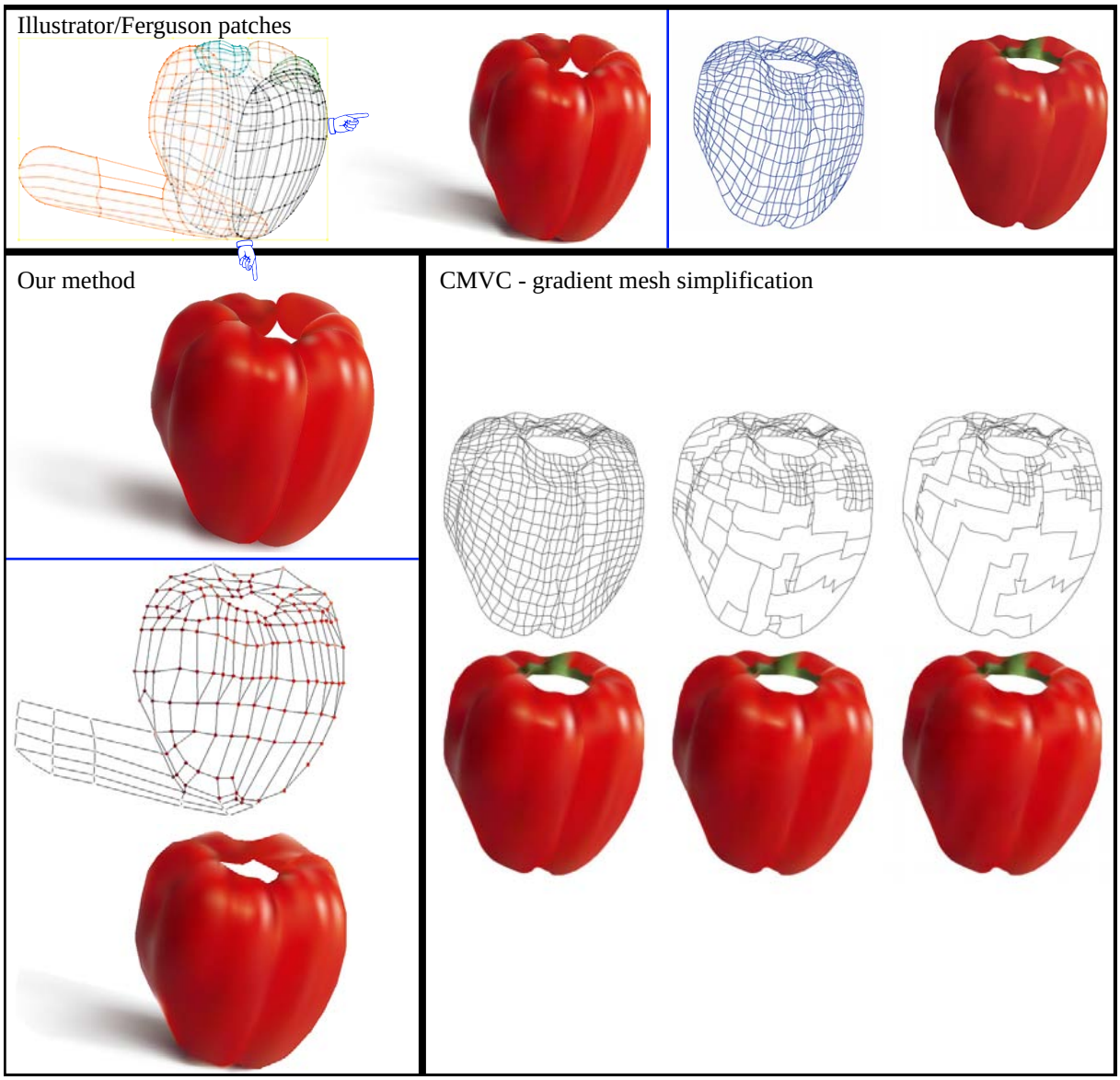


Figure 6.18: This pepper example is commonly used to visually compare methods in vector graphics. Illustrator/Ferguson patches: the left example was rendered with Illustrator; the corresponding seven-layer mesh was provided by lifeinvector.com©. The right example was rendered[53] with Ferguson patches. The single-layer mesh was extracted from the image produced by Illustrator (left), using the method of Lai et al.[53]. Gradient-mesh simplification with CMVC[57]: Given the mesh from Lai et al., two irregular meshes are produced according to their method (Section 6.2.2). Colour and gradients are projected to the remaining vertices from the original image. Our method, top: rendering using the seven-layer mesh (meshes were manually re-created). The directional colour weights were adjusted to best match Illustrator's result. Our method, bottom: single mesh with irregular elements.

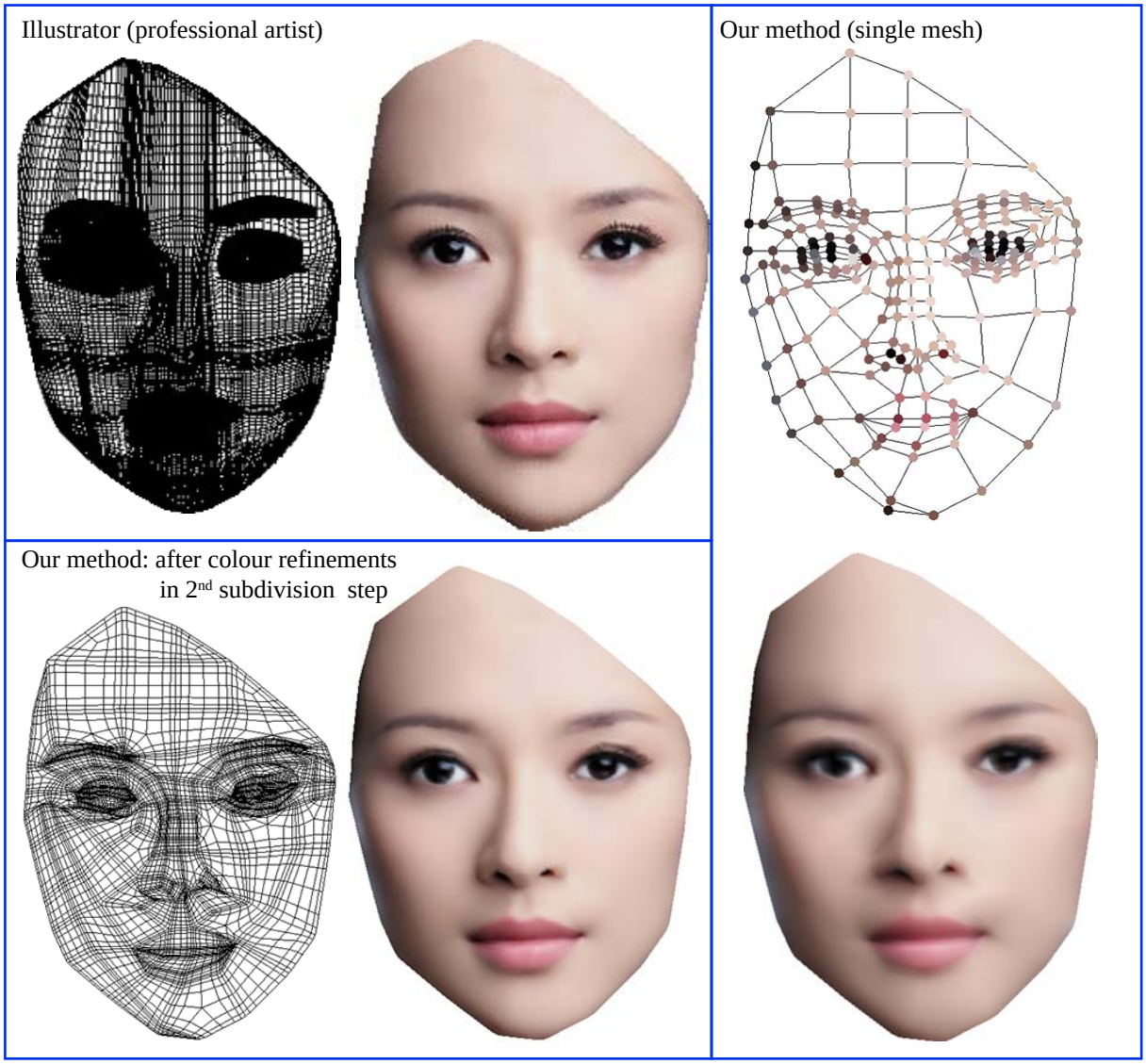


Figure 6.19: Reproduction of the face of the girl (Chinese actress and model Zhang Ziyi) in Figure 6.1. Due to the support for arbitrary mesh topology, our method supports a single, relatively sparse, mesh to represent the entire face. With Illustrator's gradient mesh tool, restricted to rectangular mesh topology, the artist is typically required to define multiple meshes, which are manually stitched, to produce such complex drawings.

saturation artefacts I demonstrated with Hermite interpolation in Section 6.2.2. This means that Illustrator's tool does not feed the derivatives directly to the Ferguson patches. Instead, Illustrator's behaviour is visually similar to our method. These comparisons indicate that Illustrator's tool uses fixed derivatives in colour space, like our approach, and therefore adjusts the colour gradient exclusively in the geometric space. Thus, Illustrator's and our method's 'derivative constraints' do not relate directly to the first derivative of the colour function. To this end, I chose to refer to our constraints are 'directional colour weights': constraints that manipulate the colour gradient locally along given directions (the edges). Figure 6.17 illustrates how the colour profile in the local neighbourhood of an irregular vertex, where derivative constraints

along edges do not make much sense, is influenced by directional colour weights.

Comparisons with Hermite interpolation

Figure 6.5 and Appendix B show informal comparisons with CMVCs. Comparing with CMVCs is a sensible because they exactly satisfy value and first derivative constraints if the boundary is piece-wise linear[57]. However, it is unclear how to adapt CMVCs to render gradient meshes of arbitrary topology with arbitrary derivative constraints, as discussed in Section 6.5. Thus, future work is necessary to adapt such Hermite interpolation methods to our problem.

In terms of computational complexity, our method uses only local linear combinations of control points to define the surface and is therefore efficient. In comparison, CMVCs require evaluation of distances (via square roots) and trigonometric functions. Our naïve CPU implementation of CMVCs is therefore slower than the similar implementation of our method. However, on modern hardware with support for parallel computations on the GPU, this difference in computational complexity is not significant.

6.5 Future work

The topological restrictions of gradient meshes have, as many researchers have previously pointed out (e.g. refs. 57;73;53;110;58), hindered its potential in many applications within vector graphics. I envision that our approach could stimulate future work in vector graphics and related fields. The two most directly related applications are free-form vector-based design and vectorisation of photographs.

I believe our contribution is a step closer to unifying diffusion curves and gradient meshes. In the future, I envision an approach that automatically creates high-quality meshes from free-form curves. Such an approach will improve on diffusion curves in two ways: local control over colour gradients and better computational performance. Figure 6.21 illustrates how local control can be advantageous in the setting of colouring images. As described in Section 4.3.2, a significant challenge is to create control meshes from boundary curves. If point constraints are also used as input, an additional challenge is to propagate these colours to mesh control points. I imagine both global and local behaviour should be achieved, as illustrated in Figure 6.21.

State-of-the-art vectorisation algorithms employ similar frameworks to ours. Early methods[95;53] use optimisation procedures to align and sample gradient meshes (that is, Ferguson patches) with an image or an image segment. Such vectorisation greatly improves the creation process of gradient meshes, since gradient meshes are typically manually traced from input images[95]. As pointed out by Liao and colleagues (ref. 58, Section 2), Ferguson patches give rise to limitations of these vectorisation algorithms, since colour discontinuities can only be defined by degenerate quads or folds. Moreover, the rectangular restriction hinders adaptive layouts, making it challenging to align colour discontinuities with image features. Liao et al. improved on this restriction by using triangular meshes, which are more adaptive. The interpolation problem was solved with the Loop subdivision scheme, which was extended to support sharp features

and colour discontinuities (tear curves). The use of subdivision surfaces enabled Liao et al. to demonstrate novel vector-based image edits.

I believe more general control meshes, not constrained to grids or triangles, could improve various aspects of vectorisation. I envision an approach where quad meshes or similar sparse constructions are used to represent multiple levels of abstractions of the image. Such sparse meshes would potentially be preferable by artists over the relative dense triangular meshes proposed by Liao et al., as illustrated in Figure 6.20. Simplification procedures, akin to Li et al.[57], could be employed to provide such sparse meshes. Finally, control meshes should be of high quality, where faces are close to equiangular and equilateral; the simplified meshes proposed by Li et al. (Figures 6.6 and 6.18) do not hold such properties and might therefore be more difficult for an artist to work with.

There are several additional challenges that arise when dealing with vector-based representations of photographs:

- It is challenging to represent a high-frequency image signal with sparse meshes. Dense meshes are therefore required to represent certain image details. On the other hand, dense meshes are difficult to manipulate directly and a major advantage of vector graphics would therefore be lost. That is, if meshes are pixel-level dense, manipulating the image via the vector representation turns similar to directly painting pixels. A related problem is to compress photographs to make the storage size of the image as low as possible, whilst maintaining the visual quality of the original photograph.
- If meshes *are* dense, users should not be forced to directly manipulate them, but instead be abstracted from the underlying vector representation. Users should still be able to manipulate the image in a vector-centric way.
- A vector representation has obvious advantages such as scalability and support for geometric operations. However, standard filtering operations, such as contrast and detail enhancement, are not natively supported in vector graphics. While Liao et al.[58] have proposed various approaches to vector-centric filtering on their triangular meshes, they rely on dense meshes to approximate the behaviour of pixel-based filters.

In the next two chapters, I provide an alternative view to vector-centric image processing. Instead of converting the image signal into a complete vector representation, I present methods that *supplement* a photograph with a vector image. This approach avoids the issue of being able to faithfully represent the original photograph, whilst still enjoying the benefits of vector graphics, such as local, explicit, control.

6.6 List of contributions

- I present a new vector primitive for defining smooth colour gradients with control meshes of arbitrary topology.
- New interpolation schemes, using subdivision, that guarantee interpolation of original colours, do not stray outside the colour space, are smooth, and adapt colour gradients according to directional colour weights. I argue our approach emulates the behaviour of

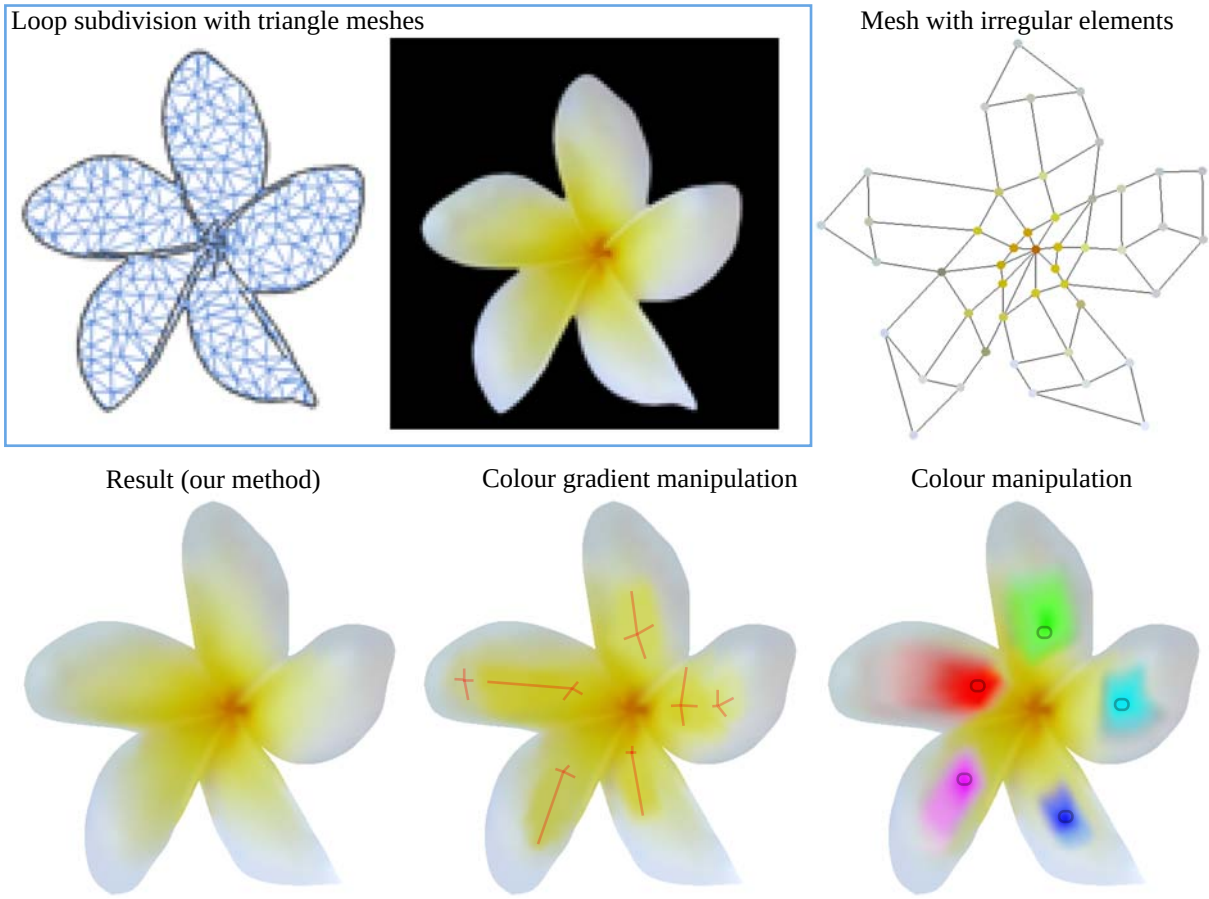


Figure 6.20: Comparison between Loop subdivision with triangle meshes for image vectorisation[58] and our method. While colours can be manipulated at vertices with Loop subdivision, the scheme does not support derivative or gradient control. Consequently, a denser mesh, compared to our approach, is in this example required in order to colour the flower. By contrast, our method both supports manipulation of the colour gradient and meshes of arbitrary topology (and not only triangles). The bottom images illustrates gradient and colour manipulation, associated with a small number of control points, which would be challenging to demonstrate with a pure triangle-based representation.

Ferguson patches in the regular setting and that it extends this behaviour to the irregular setting.

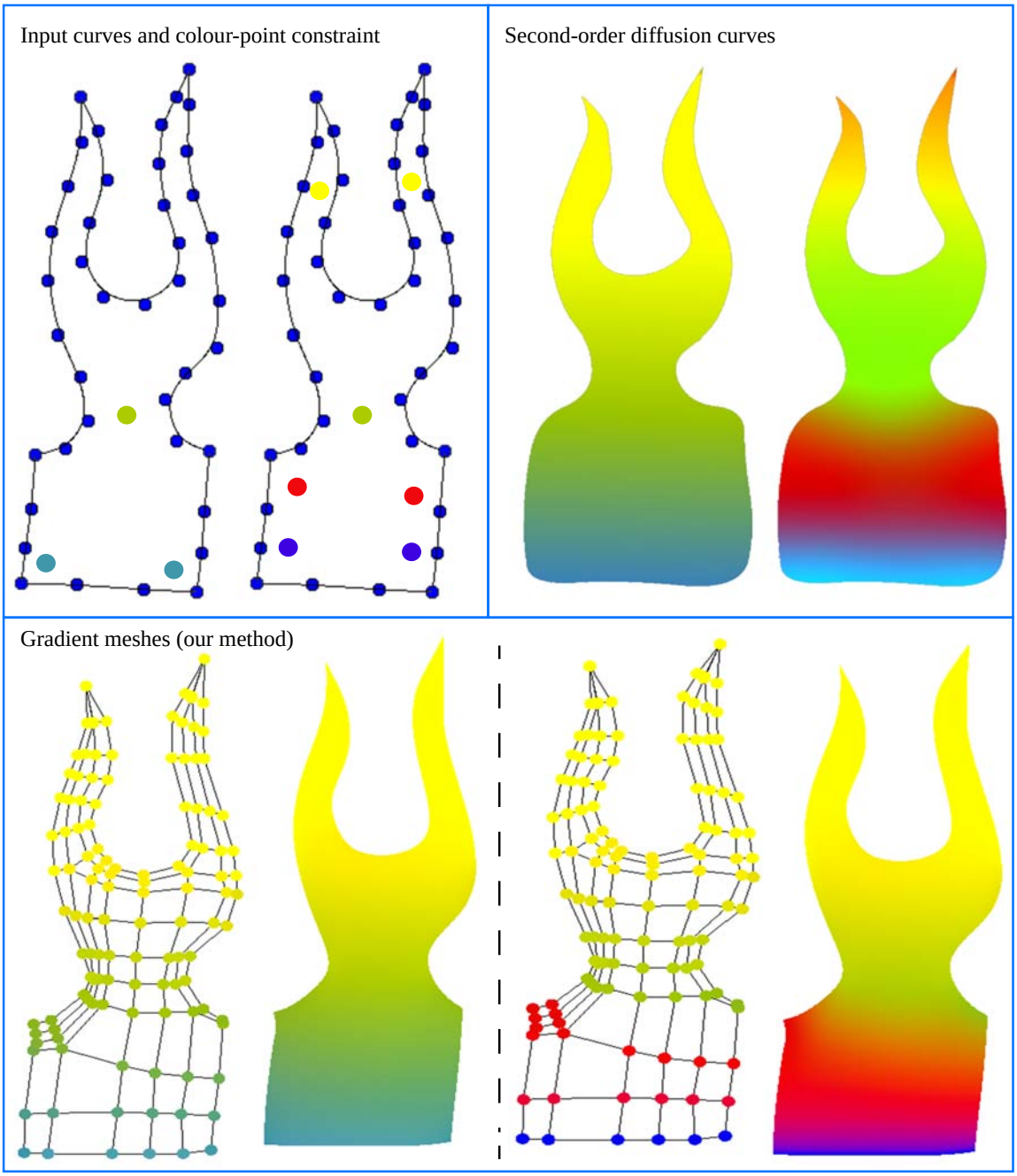


Figure 6.21: Colour editing with point constraints using second-order DCs and proposed behaviours with gradient meshes. The two results for DCs were created with three and seven points. Notice the global behaviour of diffusion: in the first example (three points), the topmost yellow colour is not explicitly specified and is produced from the three (green and blue) points. When yellow colours *are* specified, however, the topmost colour turns orange and the green and blue colours are predominantly brighter than the input colours. By contrast to DCs, our method supports local control. One the other hand, gradient meshes require control meshes, and not points, as input. In Section 4.3.2, I discussed the potential of using quad meshes to represent the colour surfaces. An additional challenge is how to propagate colours from points to mesh control points. In this example, I manually demonstrate a global diffusion-like behaviour (left) and a more local behaviour (right).

Chapter 7

Surface modelling for automatic contrast enhancement in photographs

This chapter presents research described in the following paper:

Cornsweet surfaces for selective contrast enhancement

In the previous two chapters, I presented two vector primitives for creating images from scratch. In the next two chapters, I focus on processing photographic images. In this chapter, I present a technique to automatically enhance contrast of photographs.

Contrast enhancement is a well-studied problem in image processing. Contrast is the difference in luminance or colour that makes objects distinguishable. There are many approaches to contrast enhancement in image processing: some increase the overall dynamic range of the image while others only adjust luminance or colour locally, aiming to increase the *perceived* contrast between objects. This chapter presents a method that aims to alter the perceived contrast between image regions.

I do *not* suggest a method to generally enhance the appearance of photographs, as with many previous approaches in image processing. Instead, the usage of the our framework is targeted towards applications where objects, or regions across specific edges, are emphasised with contrast-enhancement profiles. The image is enhanced automatically, where input edges are found using a high-level edge-detection algorithm that aims to identify meaningful edges in the image. I envision applications such as smart object enhancement procedures that can be automatically applied as a quick post-processing step on the digital camera device. Additionally, the framework can also be used in situations where a user manually specifies which edge to enhance and to what extent. Such user interaction is described in Chapter 8.

The type of enhancement that is created with the proposed framework has been well studied in psychology. We drew inspiration from the Cornsweet illusion where the perceived contrast of whole regions can change by modelling a specific luminance profile onto the image (Section 7.1). From this inspiration, 3D *Cornsweet surfaces* are modelled (2D in image space, 1D

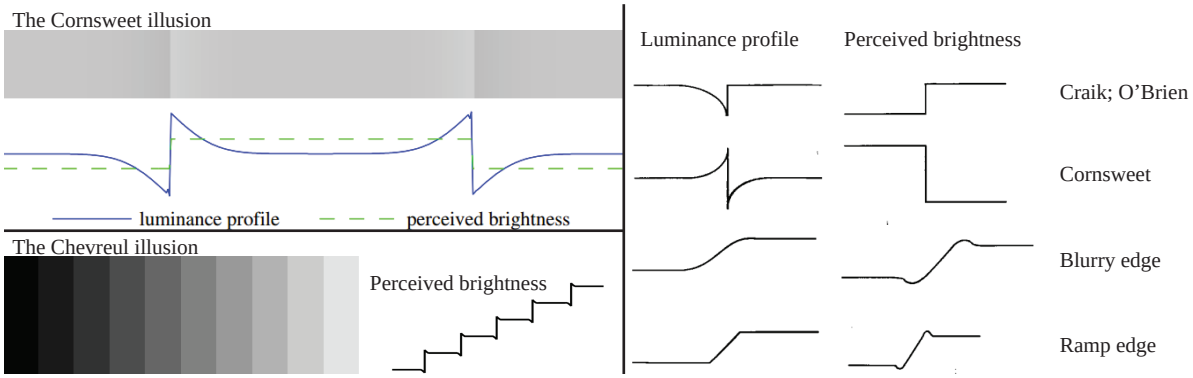


Figure 7.1: Left: The difference between the Cornsweet illusion (top) and Chevreul's illusion (bottom). The Cornsweet illusion affects large regions: the middle region is perceived as brighter even though all three regions have the same brightness except close to the edges. Chevreul's illusion induces colour gradients, even though they do not exist. In reality, each band has constant brightness. Right: Various profiles and their perceived brightness. Illustrations from Kingdom and Moulden[50].

in luminance enhancement). These surfaces are extruded or depressed in the luminance dimension of the image to alter the perceived contrast of the surround of the related image edge.

In addition to defining control meshes for these Cornsweet surfaces, pre- and post-processing steps are required (Section 7.3). The pre-processing step requires input curves to be extracted from the image. This is achieved by fitting B-spline curves to image edges that have been extracted from the image with a high-level edge detector. Using these fitted curves, control meshes are created with Solution 1 from Chapter 4. The post-processing step requires surfaces to be rendered and aligned with the original edges. Rendering the surfaces directly into the image, like the approach in Chapter 5, is insufficient because pixels (on the wrong side of the original edges) might be incorrectly altered. Such incorrect alterations are therefore explicitly dealt with.

Before outlining the solutions to these problems, I briefly present research in perceptual psychology this idea is based upon (Section 7.1). Additionally, related work in image processing is discussed (Section 7.2).

7.1 The perception of contrast

Contrast has been well studied over the centuries. It is the difference in colour that makes an object distinguishable. In art, the use of complementary colours is fundamental to creating strong contrasts. I, however, present a contrast enhancement algorithm that introduces subtle changes in luminance and not drastic changes in colour contrast. In this section, I describe the theory behind the phenomenon that this algorithm is based upon: the Cornsweet illusion. For a boader view of this research, see the review by Kingdom and Moulden[50].

There has been significant work on how colours of two different objects influence each other[50]. This type of phenomenon, known as simultaneous contrast, was first identified by Chevreul[15]

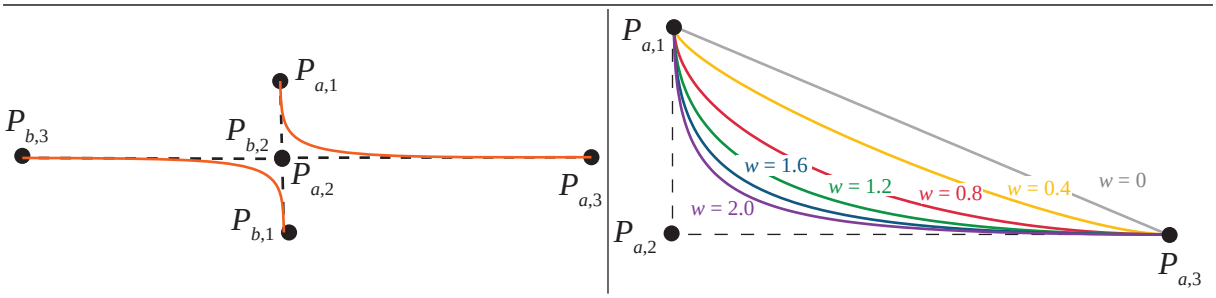


Figure 7.2: Left: The control points defined in Chapter 4 define Cornsweet profiles for each edge control point. Right: These profiles are modelled as quadratic NURBS. The weight, w, is associated with $P_{a,2}$. The other two points have a weight of 1.0.

(1839); a related illusion is called the Chevreul illusion. The theory behind this illusion describes how two entire adjacent areas influence each other. On the other hand, the theory behind the Cornsweet illusion describes how smaller surrounds of an edge influence the perception of the entire surround. The differences between these two illusions are shown in Figure 7.1(left).

To identify intrinsic properties of an object, the visual system must recover information about the reflectance of the surface of the object. The visual system solves this problem by separating the surface colour and illumination colour. The extent to which the visual system is able to solve this problem is called *lightness constancy*. Lightness constancy is broken, as demonstrated by the Cornsweet illusion, by altering the surround of the surface in question (Figure 7.1(top-left)).

A large number of alterations of this surround have been studied; in particular the luminance profile of the border. Figure 7.1(right) shows some of these, including their effect on the perceived brightness. The Cornsweet illusion[20] (1970) gives the most dramatic effect which combines both sharp and gradual changes in reflectance on both sides of the edge. Such a break in lightness constancy was first identified by Craik[21] (1940) and O'Brien[70] (1958) who only considered one side of the edge. In nature, effects such as blurry edges on shadow boundaries have been shown to have similar, but weaker, profiles[93].

Many variations of the Cornsweet profile have been proposed, including linear[105], parabolic[25], and sinusoidal and exponential ramps[50]. We decided to model a profile similar to the original profile Cornsweet proposed[20] since Cornsweet edges are known to provide the strongest effect[25]. Such a profile can be modelled with a quadratic NURBS curve with three control points. Figure 7.2 shows various profiles with different NURBS weights for the middle control point. The images presented in this chapter are rendered with the weight set to 0.8, since the profile defined by this weight has a Cornsweet-like shape. Note that the coefficients of the Cornsweet (polynomial) profile have not, to my knowledge, been derived; researchers seem to be more interested in its general shape rather than its exact definition. Thus, we had only the visual shape of the profile as a reference when we made our definition of it.

Figure 7.3 shows the perception of such profiles for various edge widths in visual degrees from a study[25] on the effect of the Cornsweet illusion. This study demonstrates that wider profiles create higher contrasts. A question that arises is whether this generalises from simple 2D stimuli to depictions of 3D environments, like those we find in photographs.

This particular problem is not fully understood in psychology yet[50]. However, Purves and col-

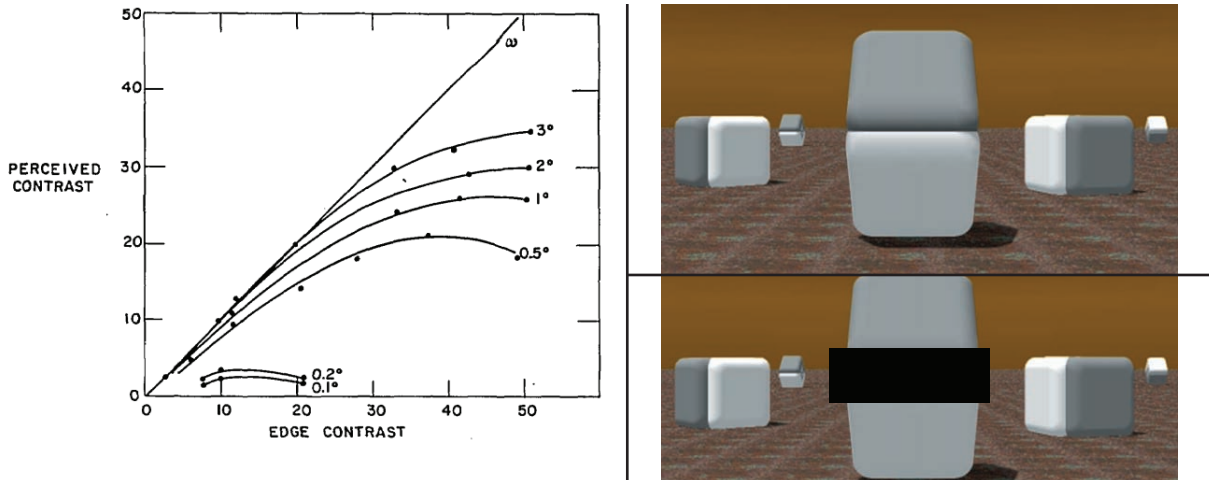


Figure 7.3: Left: Effect of the Cornsweet profile of the perceived contrast[25]. Wider profiles give rise to higher perceived contrast. Right: The Cornsweet effect can be enhanced by careful placement of surfaces and choice of colours[81]. Notice how the centre stacked objects have the same colour except at the edge.

leagues provide an interesting insight[81]: 'the perception of equiluminant territories flanking the Cornsweet edges varies according to whether these regions are more likely to be similarly illuminated surfaces having the same material properties or unequally illuminated surfaces with different properties.' This means that the Cornsweet effect decreases for surfaces with similar perceptive reflectance under the same illuminant; conversely, the effect is increased for adjacent surfaces with different perceptive reflectance under different amounts of illumination. Figure 7.3 shows a scene in which Purves et al. maximise the Cornsweet effect from their findings.

A recent study[101] (2012) has identified the artefact threshold for Cornsweet-like halos produced by the unsharp mask in photographs of real scenes. A halo artefact is defined as a bright halo surrounding an object that is clearly noticeable. The conclusion was that there is a significant difference of the artefact threshold between simple stimuli and real photographs. This can be explained with the theory of *visual masking*[30]: texture (that is, high frequencies) hide the effects of faceting, banding, aliasing, noise and other visual artefacts produced by sources of error in graphics algorithms. This is relative intuitive: if the image signal is of high frequency, larger magnitude of low frequency changes can be added to the image, without being objectionable, than in a signal with no high frequencies.

Contrast in art

There is an interesting link between these illusions and artistic techniques. Ratliff[83] has provided several examples related to paintings and pottery engravings in Asian art, some of which can be traced back to AD 1000. The Cornsweet illusion is, according to Ratliff, induced in several cases. This is remarkable, considering the fact that such strong effects are not seen in nature and the artists must have been inspired in some other way or have stumbled across the effect by chance.

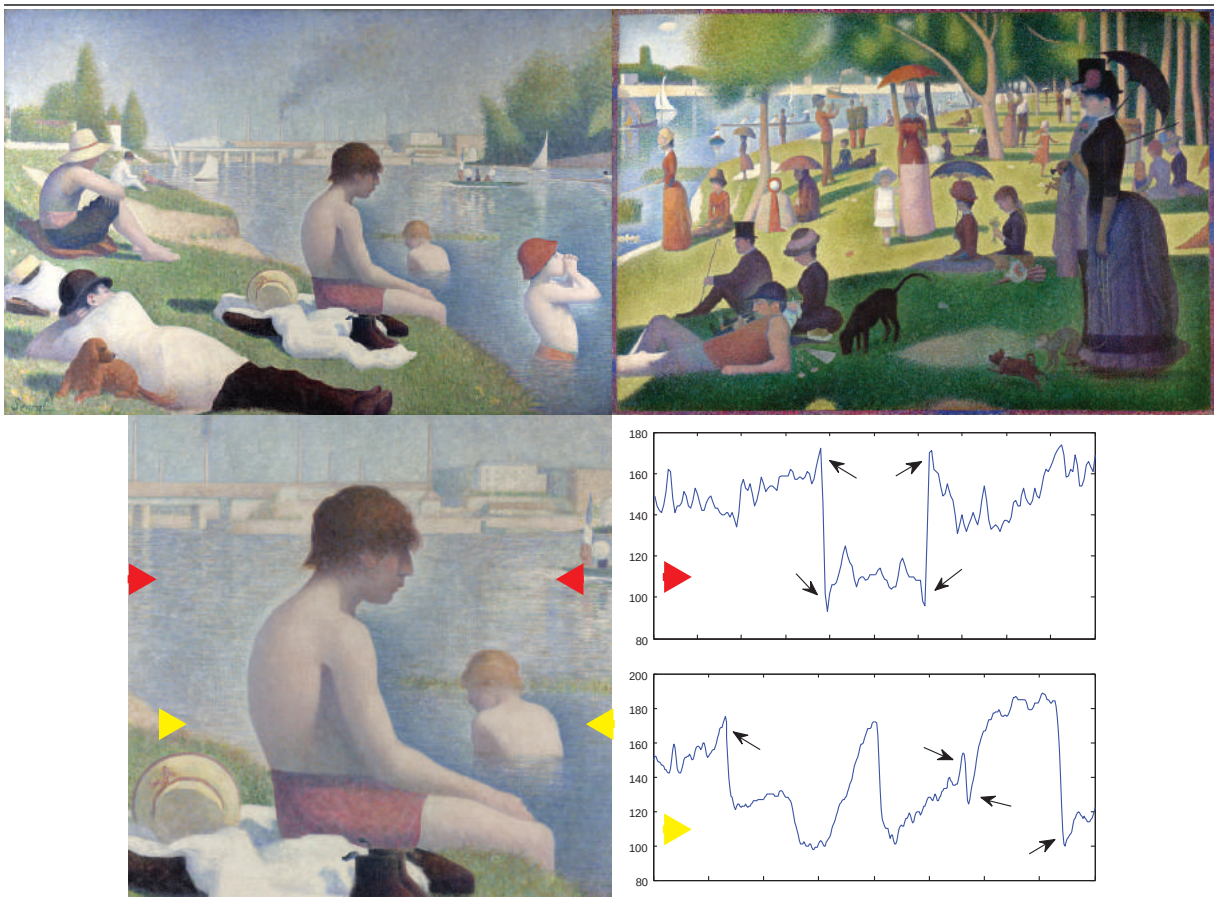


Figure 7.4: Georges-Pierre Seurat's 'Bathers at Asnières' (1883, left) and 'A Sunday afternoon on the island of La Grande Jatte' (1886, right). Notice the charismatic colourings typical for neo-impressionism, especially around objects. Bottom: Two horizontal profiles are shown. The Cornsweet profiles are indicated by black arrows. The image was blurred by a 5-pixel radius Gaussian blur to smooth out the brushstrokes.

Chevreul's Mach band illusion was picked up by artists during the impressionist era. Georges-Pierre Seurat is most notable (Figure 7.4) as he initiated the neo-impressionist movement. The techniques which arose from this period were directly mimicking the Mach band effect, with coloured halos decaying according to the artist's interpretation of Mach bands. When seen from a sufficient distance, colours charismatically blend (visually) to create the distinctive look of neo-impressionist paintings.

7.2 Background

Various methods in image processing and computer graphics have been proposed to emulate the Cornsweet illusion. All of these methods are pixel-based, building on a specific convolution operator known as the unsharp mask. In this section, I first present a brief introduction to the convolution operator and the unsharp mask (Section 7.2.1). I then present related work related to the Cornsweet illusion for computer graphics and imaging (Section 7.2.2).

7.2.1 Convolution and applications to image processing

Definition

Convolution is the standard operator for filtering images. It is an operation on two functions f and g. Convolution is typically viewed as an operator that produces a modified version of f. This modification (for example, sharpening of an image) is then characterised by g. Convolution is defined in the continuous and discrete domains as:

$$f * g(t) = \int_{-\infty}^{\infty} f(t-\rho)g(\rho)d\rho;$$
$$(f * g)(x,y) = \sum_{i=1}^{m}\sum_{j=1}^{n} f(x-i, y-j)g(i,j);$$

where f is the input image, of size $m \times n$ in the discrete setting, and g is the convolution filter.

Properties

The convolution operator has various advantageous properties. In terms of algebraic properties, it is commutative, associative, distributive, and associative with scalar multiplication. It is invariant to translation, so that any translation invariant operation can be represented as a convolution. Additionally, the Fourier transform, $\mathbb{F}$, of a convolution is the point wise product of Fourier transforms:

$$\mathbb{F}(f * g) = \mathbb{F}(f) \cdot \mathbb{F}(g).$$

By utilising the fast Fourier transform, the complexity of the operation is $O(n \log n)$, which is typically acceptable for interactive applications.

In image processing literature, *filtering* images with specific functions (for example, Gaussian filtering), typically refers to convolutional filtering.

Applications

Differential operations. The convolution operator approximates various important mathematical operations in the discrete setting. Prewitt[79] has provided differential filters for $\partial f/\partial x$ and $\partial f/\partial y$; the gradient is defined as $\nabla f = [\partial f/\partial x, \partial f/\partial y]$. The Laplacian operator can be defined via the second derivative: $\nabla^2 = \partial^2 f/\partial x^2 + \partial^2 f/\partial y^2$.

The Gaussian function G_σ (Equation 7.1) encompasses various properties which makes it suitable for image processing. It is rotationally symmetric, has a single lobe, both in spatial domain and in frequency domain, and is easy to implement. It is suitable for operations like smoothing and noise reduction.

$$G_\sigma = e^{-\left(\frac{x^2+y^2}{2\sigma^2}\right)} \tag{7.1}$$

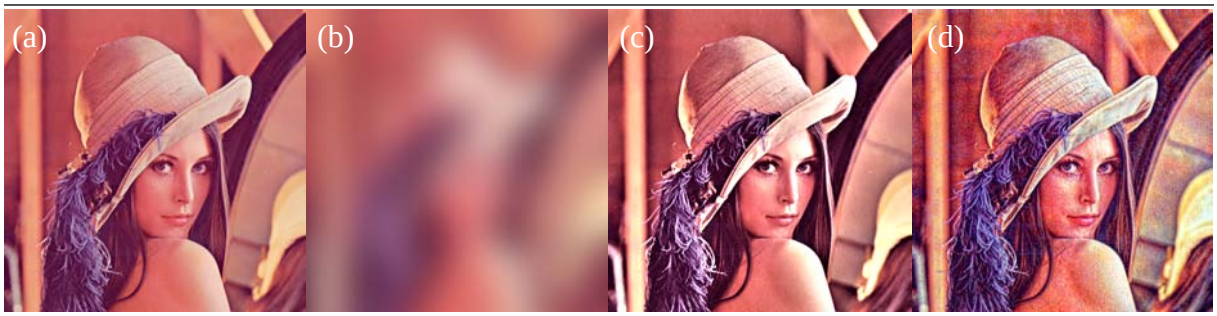


Figure 7.5: Smoothing (b) and sharpening (c,d) of an image (a). Smoothing is performed with the Gaussian filter. This unsharp image (b) is subtracted from the original image, producing a sharpening effect (c). The rightmost sharpened image (d) is produced via scale space analysis[75].

Smoothing and sharpening. The Gaussian filter smooths the image and the Laplacian filter sharpens the image. A single pass with the Gaussian filter is analogous to a weighted average of the pixel neighbourhood. In frequency domain, this means that only low frequencies are kept. Conversely, the Laplacian filter accentuates differences with local average and high frequencies are kept.

Scale-space analysis. An important aspect of image processing and computer vision is the analysis of the scale space of an image. The scale space can be produced by iterative convolution; that is, a single iteration convolves a given convolution filter with the previous convolved image. Two representations are common for scale-space analysis. The first option is to perform iterative convolution whilst keeping the original resolution of the image. The scale space is therefore represented as a cube, or image stack, where images of higher levels (original image is at level 0) are more blurred. The second option is to construct an image pyramid, where both scale and space are sampled. A typical approach is to quarter the number of pixels in each iteration, producing a regular pyramid.

There are various types of scale spaces depending on the convolution filter. The Gaussian and Laplacian[11] scale spaces are particularly useful, defined by the Gaussian and Laplacian filters respectively. For example, extreme enhancements have been demonstrated by building the Laplacian pyramid of the output image, via the Gaussian pyramid of the input image[75].

Sharpening via smoothing. Subtracting the Gaussian filter from the original image I (Equation 7.2; $C = I$, $\lambda = 1$ (default value)) creates a similar filter to the Laplacian operator, passing high frequency bands. This is known as the *unsharp mask* operator. Small standard deviations σ increase sharpness and large standard deviations increase contrast[68].

$$I' = (1 + \lambda)I - \lambda G_\sigma * C \tag{7.2}$$

Figure 7.5 demonstrates some examples to the applications given above.

Figure 7.6: Unsharp masking, increasing the Gaussian kernel (from left to right). Smaller kernels only sharpen the image, while larger kernels both sharpen and increase contrast.

7.2.2 Cornsweet-style contrast enhancement with the unsharp mask

The unsharp mask produces the Cornsweet profile in some circumstances. Consequently, its relation to the Cornsweet illusion has been well studied in perceptual psychology[50]. The Cornsweet profile is best produced for image features whose scale is similar to or larger than the size of the Gaussian kernel[52]. Moreover, smaller Gaussian kernels are not suited for such contrast enhancement as high frequencies in the image signal are amplified[68;52] (Figure 7.6). Finally, the unsharp mask does not increase contrast, in a Cornsweet-like manner, whilst avoiding sharpening the image; that is, as contrast is enhanced, the overall look of the image is typically altered. Consequently, previous methods propose alternative methods to unsharp masking so that the contrast enhancement does not alter the original look of the image.

Previous work

The method proposed in this chapter is not the first to apply the Cornsweet profile to enhance the perceived contrast in images. The idea was first implicitly introduced by Luft and colleagues[62] (2006). Inspired by neo-impressionist art, Luft et al. noticed that the unsharp mask can produce similar selective enhancements for visually important edges using depth maps. More specifically, the depth map is blurred with the Gaussian filter, which is then subtracted from the original image (in Equation 7.2, C equals the depth map). Edges in depth maps typically refer to object boundary edges and can therefore be deemed more 'interesting' than edges in the original image. Moreover, as their method is dependent on pixel-accurate depth maps, it is best suited for synthetic rendering. In such applications, unsharp masking the depth map enhances the cognition of spatial distribution of objects (Figure 7.7).

Cornsweet-style enhancement for synthetic applications has been further improved for temporal coherency by adapting the unsharp mask to geometric contexts[87] (Figure 7.7). Instead of processing intensities of pixels, a 3D adaption processes the outgoing radiance defined on the mesh to a viewpoint. Thus, temporal coherency is automatically achieved as the enhancement operation is associated with the geometry. Such Cornsweet-style enhancement was later shown to be significantly preferred over rendering with no enhancement[39]. However, some aspects of this contribution, relating to larger widths of the profiles, have been disputed[101].

The first method, to my knowledge, aiming to explicitly produce Cornsweet profiles in images was devised by Krawczyk and colleagues[52] (2007). Their method performs unsharp masking

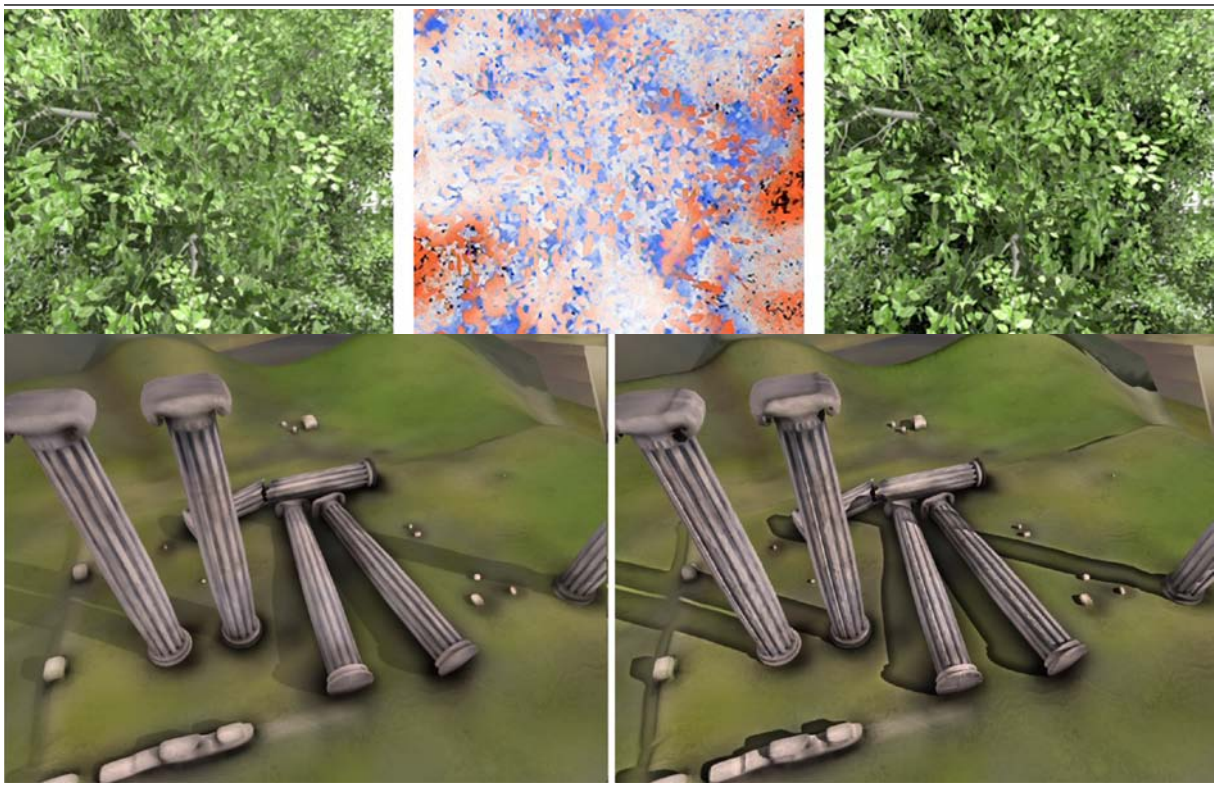


Figure 7.7: Unsharp masking to enhance the cognition of scene coherence for synthetic rendering. Top: unsharp masking the depth map (middle) provides a selective enhancement of important edges[62]. Bottom: unsharp masking the scene geometry, according to outgoing radiance towards the camera, enhances the perception of the spatial layout of scene objects[87].

in the input image's (pyramid) scale space, where the Gaussian kernels were adjusted to match the contrast of a reference image at each level of the pyramid. Their main application was to bring back lost (perceived) contrast from high-dynamic-range (HDR) tone mapping. That is, a tone mapped image is processed, reintroducing lost contrast via the Cornsweet effect, using the original HDR image as reference (Figure 7.8).

The main advantage of applying the unsharp mask in the scale space is that sharpening can be avoided. At low levels of the pyramid, where the image is less blurred, the unsharp mask tends to sharpen the image, even with large kernels. However, at higher levels, where the image is more blurred and high frequencies have been smoothed, unsharp masking with large kernels can be employed *without* sharpening the image. Thus, to apply the Cornsweet effect without introducing sharpening or other major alterations to the image, the Gaussian kernel can be increased as the images in the pyramid get smaller and more blurred.

While the method of Krawczyk et al. works well for applications like post-processing of tone mapping, where a reference HDR image is available, it is not suitable for images without a reference. The problem for single images is to define consistent kernel sizes so that users do not have to fine-tune this parameter for each image. From my experiments, this problem is non-trivial. I will later in this section demonstrate intrinsic limitations of the unsharp mask, which indicate the challenges of this problem.

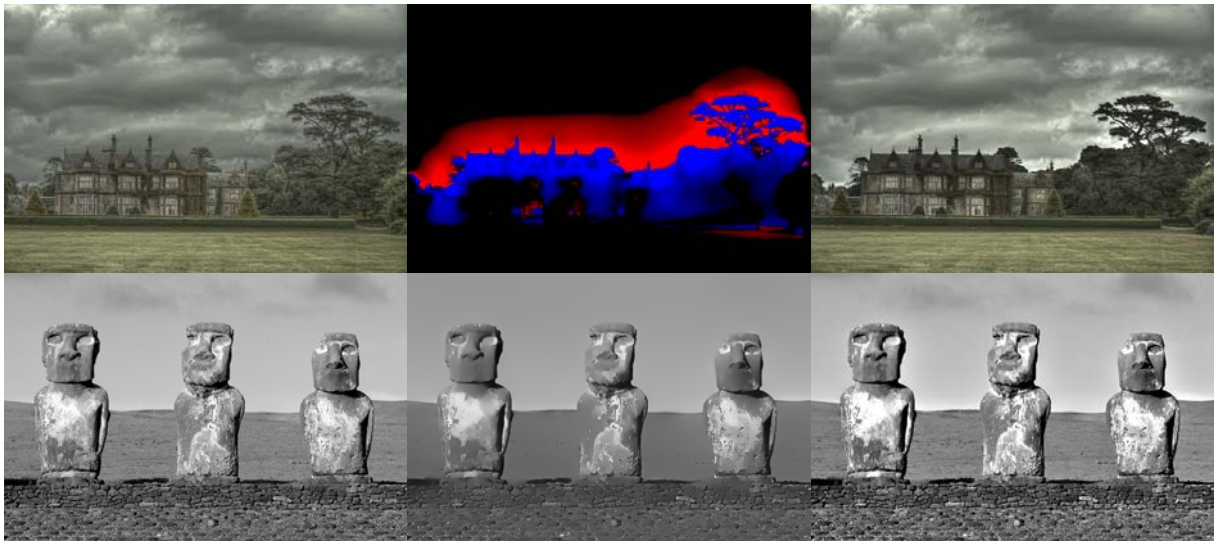


Figure 7.8: Selective Cornsweet-style contrast enhancement (left: input; right: result). Top: reintroducing lost contrast from HDR tone mapping[52]. The middle image shows the added enhancement (red: positive, blue negative). Bottom: Unsharp masking using a segmented version (middle) of the input image[101].

The first (and only) method to produce Cornsweet-style contrast enhancement on a selective set of edges from a single image was proposed by Trentacoste and colleagues[101] (2012). Their 'additional' image (C in Equation 7.2) for the unsharp mask is defined as a colour segmentation of the original image (that is, a piece-wise linear colour image), produced automatically from the input image with a state-of-the-art edge-aware smoothing method[28]. Since the segmented colour image is piece-wise linear, with a constant signal except at hard edges, the unsharp mask will typically not produce additional sharpening (Figure 7.8). As with the method by Krawczyk et al., it is challenging to define consistent kernel sizes for general images without introducing saturation artefacts, as described next.

Limitations of the unsharp mask

From my experiments on modelling the Cornsweet profile with the unsharp mask, two limitations have been identified. First, to correctly model the Cornsweet profile from a given edge, the following topological constraints must be satisfied: the edge must be sufficiently straight and must not be too close to other edges. This limitation relates to the size of the kernel. Second, edges of different luminance give rise to different magnitudes of the profile. Thus, achieving a constant effect across images is challenging. This limitation relates to the magnitude of the enhancement.

Figure 7.9 demonstrates the topological restrictions of the unsharp mask. For straight edges, where the convolution kernel is only covering two image regions, the Cornsweet profile is modelled accurately by the unsharp mask. However, if the kernel covers additional regions not associated with the enhancement edge, the unsharp mask operation will not typically produce the Cornsweet profile. This can happen where nearby edges are too close to each other or at

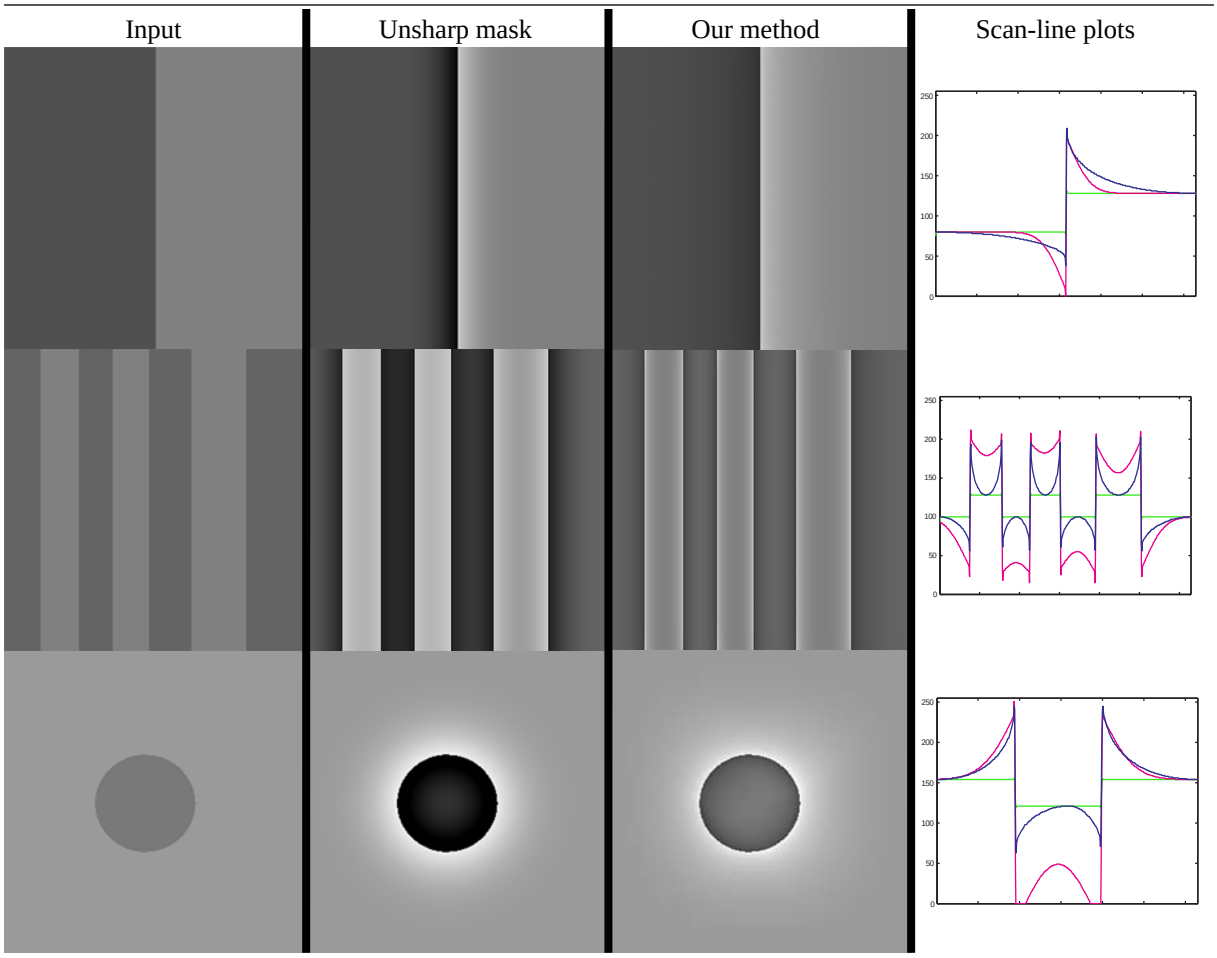


Figure 7.9: The unsharp mask models the Cornsweet profile well for straight edges adjacent to sufficiently large areas (top). However, saturation can occur for narrow regions (middle row) and curved edges (bottom). Our method handles all of these cases. Additionally, our method guarantees to level out the Cornsweet effect to the original colour, whereas this is not guaranteed by the unsharp mask, as shown in the scan-line plots (blue curves: our method, magenta: unsharp mask, green: original image).

curved edges.

To reduce these saturation artefacts, the magnitude of the blurred image, and therefore the Cornsweet profiles, can be adjusted to fit the given image. While such editing can be sufficient for single images, it is not suitable for automatic applications and mass editing of image libraries. The unsharp mask is not suitable for such automatic applications due to its image-dependent behaviour. Figure 7.10 demonstrates this point: the magnitude of the Cornsweet effect is dependent on the difference in luminance of the related image regions. Such behaviour is expected from any convolution-based image operator. By contrast, our method is robust to such image content.

The discussion above identifies inherent limitations of filter-based methods for the application of Cornsweet-style contrast enhancement. To counter these issues, we investigated a fundamentally different approach with NURBS surfaces. Surface modelling can achieve sufficient control

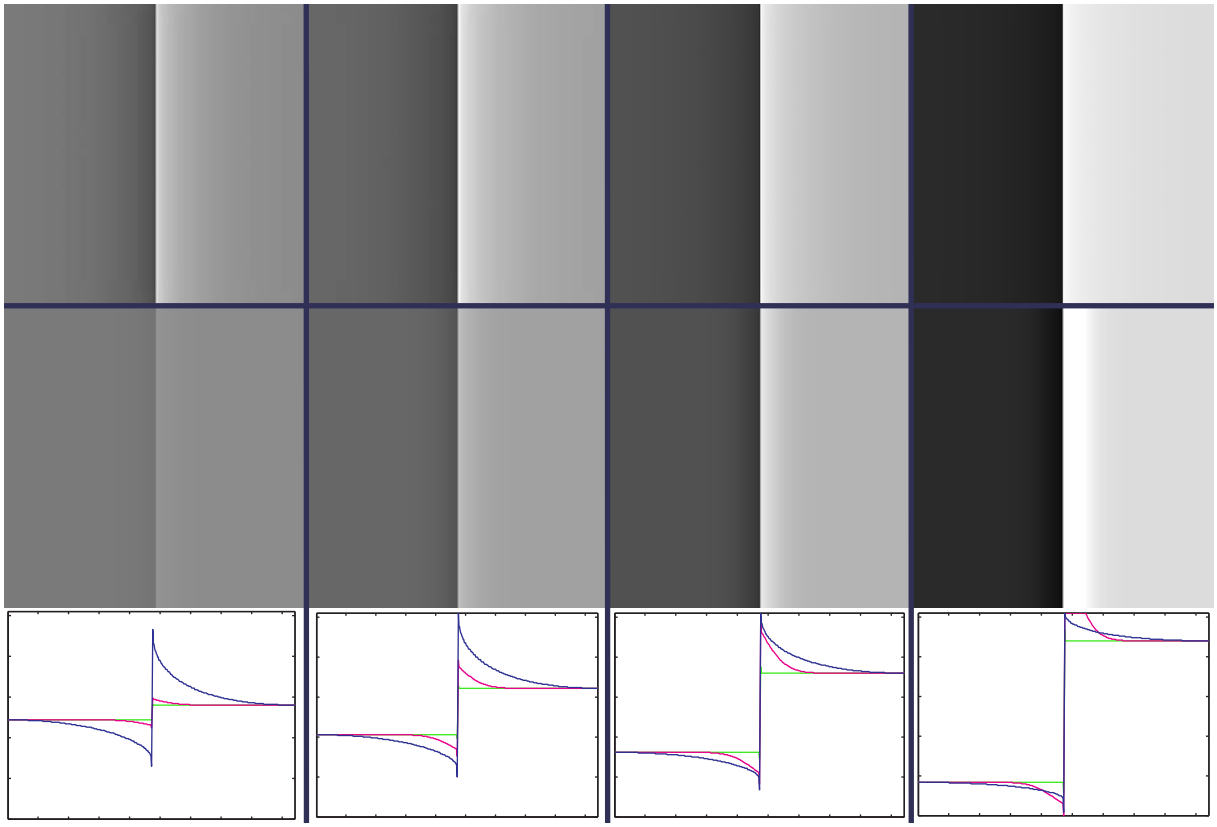


Figure 7.10: Images produced with our method (top) and the unsharp mask (bottom) with fixed parameters for all images. The scan-line plots show that our method (blue) has a constant behaviour and avoids clipping. The unsharp mask (magenta), on the other hand, increases enhancement magnitudes as the difference between the regions rises. The green plots show the original luminance.

over the behaviour of the given effect, as demonstrated by the figures above. The research question addressed in the remainder of this chapter is whether these advantages can be generalised to general photographs.

7.3 Selective contrast enhancement with Cornsweet surfaces

In this section, I present Cornsweet surfaces for increasing the perceived contrast in images. The main technical challenge is to create suitable control meshes related to a set of input image edges. This problem was discussed in Chapter 4. This section presents the framework for automatically producing a selective enhancement of an input photograph. The additional problems that are dealt with have been well studied in previous literature. Thus, I do not present techniques aiming to improve such problems. Instead, I discuss the reasoning behind the various design decisions we made when forming this framework.

The input to the framework is an RGB image and two (global) parameters: the maximum spatial extent $\sigma \in \mathbb{R}^+$ of the Cornsweet profiles and the magnitude $\lambda \in \mathbb{R}^+$ of the enhancement.

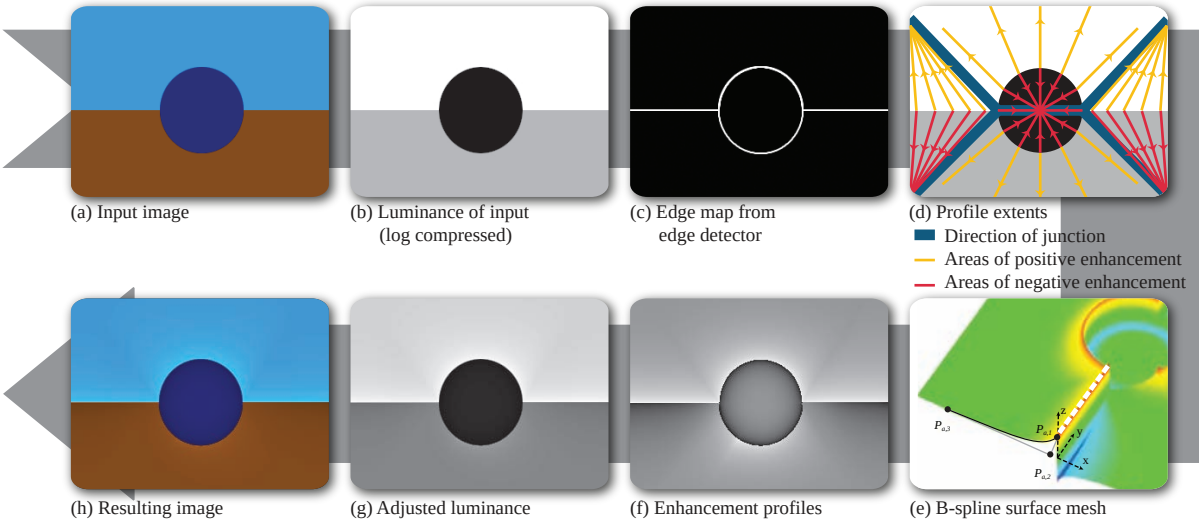


Figure 7.11: An overview of the steps of the enhancement algorithm.

Figure 7.11 shows the pipeline of the contrast enhancement framework. The following steps are performed:

1. Input edges extracted from the input image (Section 7.3.1).
2. Edges fitted to produce B-spline curves (Section 7.3.2).
3. The B-spline curves are converted into suitable control meshes (Section 7.3.3).
4. Cornsweet surfaces, defined by the control meshes, are applied to the original image (Section 7.3.4).

It is important that the luminance changes introduced are tuned for human visual perception. We decided to use the chrominance-luminance colour space xy$Y$ (ref. 5), where Y represents luminance and xy represent chrominance. The Cornsweet surfaces are applied only to the luminance channel and the chrominance channels are therefore not altered. Furthermore, the visual perceptual system is sensitive to multiplicative contrast; for example, a luminance ratio of 1:2 is perceived as the same contrast as a ratio of 100:200. Luminance should therefore be added in the log domain of Y.

7.3.1 Boundary edge detection

The Cornsweet contrast enhancement should only be applied to 'important' edges in the image. As with previous methods[62;101], such important edges refer to object boundary edges. The reasoning behind this choice is that Cornsweet-style contrast enhancement between scene objects enhance the cognition of scene coherence. Thus, a computer vision boundary detection algorithm is used to define the edges to be enhanced.

Boundary detection algorithms (e.g. refs. 37;54;86) define edges according to a wide range of high- and low-level cues such as texture, colour, junction configuration, area, length, and 3D

cues. Note that edge detection and segmentation problems are difficult to define (and solve) as object segmentation in images is subjective[64].

We found the boundary detection algorithm by Hoiem and colleagues[37] sufficiently robust for us to demonstrate a wide range of automatic results. They employ a machine learning approach based on boundary, region, surface layout, and depth-based cues, as well as a training data set from human labelling. Due to the complexity of the algorithm, the computation time for an image is typically around 5 minutes. The advantage of using their (MATLAB) system is that a set of edges connected by junction points is returned. No post processing of the data is required and spline curves can be directly extracted.

7.3.2 B-spline fitting

The discrete edges must be converted to B-spline curves for the creation of control meshes. These curves will form the boundary of the Cornsweet surfaces. Curve fitting is a well-studied problem with many different types of solutions. We chose to define curve control points by pixels of high curvature of the discrete edges. This approach enabled us to define relatively uniform spacing of the control points. We opted for uniform spacing becuase the curve control points are directly used to sample Solution 1. Additionally, recall from Section 4.2.3 that Solution 1 requires 'corners' to be defined as input. These corners are identified in this subsection.

We chose to use the method of Medioni and Yasumoto[66] to define the fitted B-spline curves. Their method identifies pixels of high curvature by fitting local cubic B-spline curves. Using these splines, Medioni and Yasumoto proposed two types of thresholds, a curvature threshold and a displacement threshold, to identify pixels that can be used as curve control points of B-spline curves. Below, I provide the equations, given by Medioni and Yasumoto, to compute the two measures used to perform this thresholding. For the reasoning behind these equations, I refer to ref. 66.

A discrete edge is defined by pixels (defined by (x, y) coordinates) arranged in a list (conveniently, this structure is also provided by the edge detector of Hoiem et al.). The discrete curvature measure of Medioni and Yasumoto is defined as:

$$C_v = 4\frac{(x_{i+1} - x_{i-1})(y_{i-1} - 2y_i + y_{i+1}) - (y_{i+1} - y_{i-1})(x_{i-1} - 2x_i + x_{i+1})}{\left((x_{i+1} - x_{i-1})^2 + (y_{i+1} - y_{i-1})^2\right)^{3/2}}.$$

The displacement measure represents the displacement between the fitted (local) cubic B-spline curve and the discrete edge pixel. It is defined as:

$$\delta_x = x_{i-1}/6 - x_i/3 + x_{i+1}/6;$$
$$\delta_y = y_{i-1}/6 - y_i/3 + y_{i+1}/6.$$

An edge pixel is considered as a potential curve control point if the curvature measure is above 0.4 and the displacement measure is above 0.2. These thresholds were suggested by Medioni and Yasumoto. To ensure relatively uniform spacing, the spacing between control points are ensured to lie between 10 and 20 edge pixels. The list of curve control points Q_i is constructed

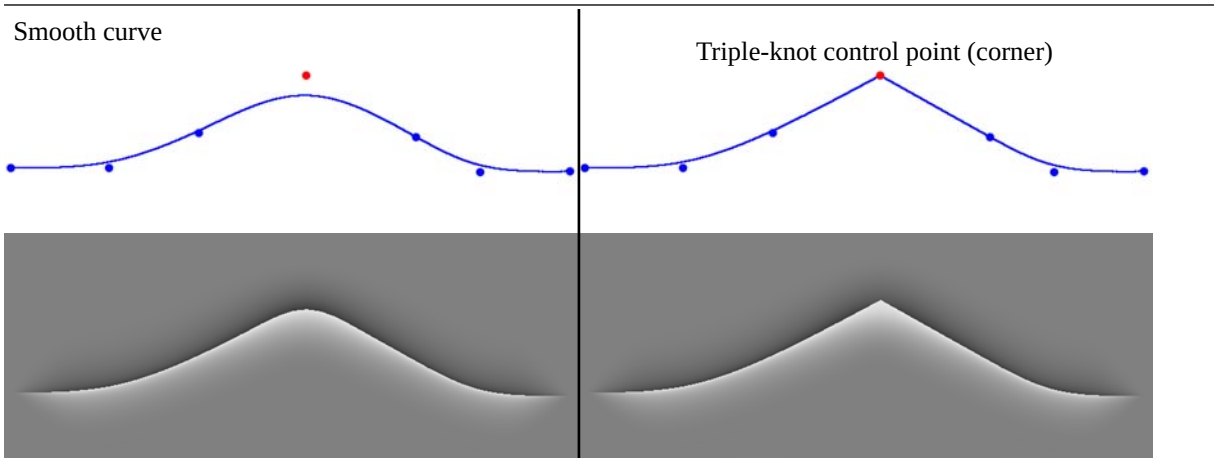


Figure 7.12: Illustration of triple-knot control points. Triple knots force the curve to interpolate the control point. The bottom images show the resulting Cornsweet enhancements for these two curves.

by iteratively adding the point with the highest curvature and displacement and with a spacing, compared to current points in Q_i, above 10 edge pixels. Note that there is not much reasoning behind the range of the pixel spacing (10 to 20 pixels). The given spacing provided sensible results. Informally speaking, I do not see why the spacing should be below 3 pixels (too dense sampling) and above 70 pixels (too sparse sampling). A more strict sampling (e.g. between 10 and 20 pixels) means that the sampling artefacts described in Section 4.2 are less likely to occur.

After the high-curvature and large-displacement edge pixels have been added to the list of Q_i, additional edge pixels are included to ensure the spacing to be below 20 edge pixels. That is, if the pixel spacing (along the edge) between two curve control points Q_i is above 20 pixels, the edge pixel on the midpoint (along the edge) between the two related pixels is added to Q_i. For example, if the edge index of Q_1 is 1 and the edge index of Q_2 is 40, then the edge pixel with index 21 (rounded from 20.5) will be added to the list of Q_i (becoming Q_2).

Corners are now tagged. Such corners are identified by considering the displacement between the discretised cubic B-spline curve defined by Q_i and the discrete edge. For all Q_i, the distance from its related curve point to the edge is found. In my implementation, the distance from a point to a discrete edge is found brute-force by calculating all distances from that point to all edge pixels and then picking the shortest distance. A corner is tagged $\hat{Q}_i$ if the largest displacement is above 1 pixel. This corner is made, technically, a corner by forcing the B-spline curve to interpolate $\hat{Q}_i$; thus, the continuity is C^0 at $\hat{Q}_i$ (as one would expect at corners). This can be achieved by zeroing the three B-spline knot intervals associated with $\hat{Q}_i$. As a consequence, $\hat{Q}_i$ is defined with a *triple knot*. Figure 7.12 illustrates such a triple-knot control point. This corner-identification process is performed iteratively until the largest displacement is below 1 pixel. Note that as more triple knots are included, the B-spline curve will come closer to the edge, making this iterative labelling procedure terminate quickly.

Alternatively, one could use the curvature measures from Medioni and Yasumoto to label corners. I opted for the approach above because it provides a sensible way to incorporate (sharp) corners. However, if performance issues arise with this approach, thresholding curvature measures is a sensible option. Other models based on, for example, active contours can be used. I refer to book by Prince (ref. 80, Section IV) for a broader view of this topic.

7.3.3 Defining Cornsweet surfaces

The problem of defining control meshes for the Cornsweet surfaces is twofold. First, the control meshes used to render Cornsweet surfaces must be defined. This is the problem described in Chapter 4. Second, the coordinates in the luminance domain need to be defined. These coordinates should not only rely on the user-defined magnitude of the Cornsweet enhancement, but also rely on the image content.

The Cornsweet surfaces are modelled as NURBS surfaces. The reason for using NURBS was that non-uniform knot intervals were experimented with. Additionally, the degree along the direction of the B-spline curves; that is, along the first dimension of the parameter space u, is 3 (cubic), whereas the degree along the Cornsweet profile; that is, along the second dimension of the parameter space v, is 2 (quadratic). We chose to model the Cornsweet profile as a rational quadratic profile because it can naturally provide a single parameter (the weight w). By contrast, two control points would have been required ($P_{i,2}$ and $P_{i,3}$) to model a cubic profile. This is less convenient since more parameters would have to be tuned to model the Cornsweet profile (with no obvious advantages).

A sensible option, as demonstrated in Chapter 5, is to use subdivision. However, a problem with subdivision is that it not easily defined to the setting above (with cubic profiles along u and quadratic profiles along v). To employ subdivision, bi-quadratic surfaces, as achieved by the Doo-Sabin scheme, could be an alternative if one accepts quadratic profiles along the surface boundary. While the Doo-Sabin scheme does not natively support NURBS weights, rational subdivision surfaces could be used to mimic such weights[90].

Definition of control meshes

Recall the following definitions of the input curves and attributes from Section 4.1.2: An input B-spline curve is defined by a sequence of n control points $Q_i = (x_i, y_i)$, $i = 1, \ldots, n$. A set of attributes, height and extent, are attached to each control point:

- extent, $e_i \in \mathbb{R}_0^+$: defines the extent of the mesh in the perpendicular direction in the image plane from the curve at Q_i. The extent attribute is defined globally in this chapter (becuase the method is automatic). It is set equal to σ, the input parameter for extent. In Chapter 8, I discuss ways for the user to manipulate the extents locally (using strokes).

- height, $h_i \in \mathbb{R}$: defines the height of the mesh in the perpendicular direction to the image plane at Q_i. As with extent, this attribute is set globally, in this case, by the λ parameter. The height of the mesh also takes local considerations into account. This is discussed later in this section, when I am defining the luminance coordinates, $z_{i,1}$.

The shape of the surface profiles is dictated by the weight w. In this chapter, w is fixed to 0.8 to model the Cornsweet profile. In Chapter 8, the user can alter this parameter to control the shape of the enhancements, similar to the shading profile in Chapter 5. The advantage over the shading profile is that we are only concerned with a single scalar rather than a curve, which can be sufficient for some applications (Chapter 8).

Recall the following definitions of the output control meshes from Section 4.1.2:
$P_{i,j} = (x_{i,j}, y_{i,j}, z_{i,j})$, $i = 1, \ldots, n; j = 1, \ldots, m$, where n is the number of control points associated with Q_i. The control mesh has three rows; thus, $m = 3$. The following coordinates can then be defined, given $Q_i = (x_i, y_i)$:

$$(x_{i,1}, y_{i,1}) = (x_i, y_i);$$
$$P_{i,2} = (x_i, y_i, 0);$$
$$z_{i,3} = 0.$$

The mesh control points $P_{i,j}$ related to a curve control point Q_i therefore forms a right-angled triangle, with the shape of the profile dictated by the weight w associated with $P_{i,2}$. This configuration, including the influence of w, is shown in Figure 7.2 in Section 7.1. The weights of the other two control points $P_{i,\{1,3\}}$ are 1.0. A resulting NURBS surface is shown in Figure 7.11(e).

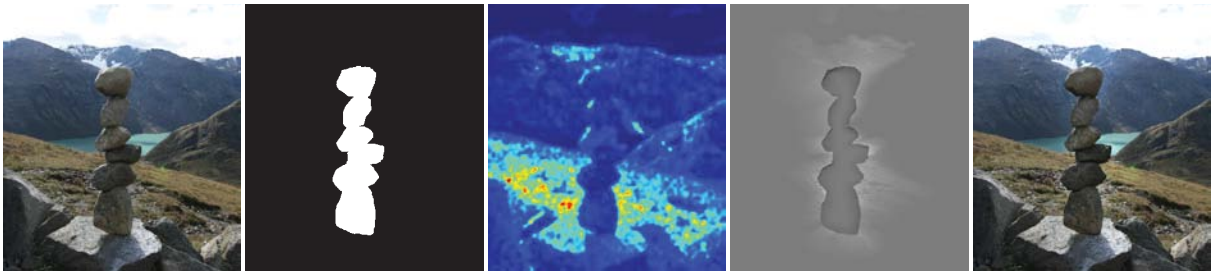
The coordinates $(x_{i,3}, y_{i,3})$ are defined by Solution 1 described in Section 4.2.3. The two solutions presented in Chapter 4 are compared in Section 4.2.6, where I provided reasons to why Solution 1 should be employed, rather than Solution 2, for the application presented in this chapter. In short, Solution 1 is more useful for automatic applications where corners can be easily identified. Specifically, corners are more easily identified from the image edges compared to identifying 'correct' extents along the edges. Using the three-step solution (that is, including the optional thresholding step), meshes can be created without any restrictions on the extent attribute. To model Cornsweet profiles 'as far out as possible', σ can be set to infinity (in practice, equal to the image diagonal).

The motivation for creating meshes as far out from edges as possible is that the unsharp mask often break (causing saturation artefacts) for larger filters (Section 7.1). Large extents of the profiles can give rise to stronger enhancements (Figure 7.3 in Section 7.1). To reflect this, I recommend, in the automatic setting, the σ to be set to infinity. Thus, the shapes of the edges, represented by B-spline curves with corners, dictate the extent of the profiles. According to Solution 1, the profiles will be defined as far out as possible, but still follow the general outline of the related edge.

The threshold used in the third step of Solution 1 is set to 15 pixels. This thresholding step is used because neighbouring extents (length of L_i lines) may be very different after the two first steps which can lead to extents with inappropriate profiles (see Section 4.2.4 for more discussion). Varying this threshold has limited impact on the final result, within reason. Its value should depend on the density of the sampling. If a dense sampling is used, a strict threshold should be used (e.g. 5 to 25 pixels for spacing between ca. 3 and 20 edge pixels), and vice versa (e.g. between 25 to 75 pixels for spacing between ca. 20 and 70 edge pixels).

Defining the luminance coordinates

The coordinates of the luminance dimension are primarily controlled by the user-specified magnitude of the Cornsweet enhancement λ. Additionally, the enhancement magnitude should also vary along the image according to image content. Recall from Section 7.1 that more contrast can be introduced where visual masking occurs. Thus, higher enhancement magnitudes can be modelled in image regions with higher frequencies.



(a) Input (b) Edge map (c) Textureness map (d) Resulting adjustment (e) Result ($\lambda = 1.00$)

Figure 7.13: Owing to visual masking, higher contrast can be added in areas of high frequencies ('textureness').

To account for visual masking, a measure of *textureness* T, proposed by Bae and colleagues[2] (Figure 7.13(c)), is used. The textureness measure is computed by the cross-bilateral filter[26;77] between the original image I and a high-pass filtered version H of I. The high-pass version is computed by filtering I with the Laplacian operator. The cross-bilateral filter is defined as:

$$T(I)_p = \frac{1}{k} \sum_{q \in |H|} g_{\sigma_s}(||p-q||) g_{\sigma_r}(|I_p - I_q|)|H|_q;$$

$$\text{with: } k = \sum_{q \in I} g_{\sigma_s}(||p-q||) g_{\sigma_r}(|I_p - I_q|);$$

where g_σ denotes the Gaussian operator with a standard deviation of σ ($\sigma_r = 0.5; \sigma_s = 4$). The reason for using this filtered high-pass version, and not H directly, is that the Laplacian operator does not take local edge information into account. Frequencies will then bleed across edges. This does not happen with an edge preserving approach, such as the cross-bilateral filter, and therefore provides a more suitable solution (in this application) close to edges. Finally, this textureness image T is scaled so that it is defined between 1 and 1.2.

To account for textureness, an intermediate $z'_{i,1}$ is computed:

$$z'_{i,1} = s \log_{10}(1+\lambda) \overline{T}_{M_i}; \tag{7.3}$$

where $s = \pm 1$ depending on the orientation of the countershading profile for that region. If the average luminance of a region is higher than the region M_i across the curve, $s = 1$ and vice versa. M_i is the pixel neighbourhood, extending to 40% of the extent of the current profile. Hence, $\overline{T}_{M_i}$ is the average textureness in the neighbourhood of $(x_{i,1}, y_{i,1})$.

The local neighbourhood M_i represents the pixels that are primarily affected by the profile. It is defined as the pixels from the edge to 40% of the profile because the Cornsweet profile, defined by the rational quadratic surface profile, starts to level out to zero at roughly 40% from $P_{i,1}$ to $P_{i,3}$.

The luminance profiles are added in the log domain of Y because the visual perceptual system is sensitive to multiplicative contrast, as mentioned in the introduction to this section. To guarantee a well-defined value of $z'_{i,1}$, λ is added by 1 when its logarithm is computed.

To avoid clipping where the enhancements would exceed the maximum or minimum allowed luminance of the output format, $z'_{i,1}$ is reduced, if necessary, by considering the luminance

values in M_i. The following test is performed to keep pixel values within the dynamic range $[\mathcal{Y}_{min}, \mathcal{Y}_{max}]$ of the input image:

$$Z_i = \max_{(x,y)\in M_i} \log(1 + Y(x,y));$$
$$z_{i,1} = \begin{cases} \log(1+\mathcal{Y}_{max}) - Z_i & \text{if } Z_i + z'_{i,1} > \log(1+\mathcal{Y}_{max}); \\ z'_{i,1} & \text{otherwise.} \end{cases}$$

For negative values of $z'_{i,1}$, a similar calculation is performed to avoid results below $\log \mathcal{Y}_{min}$.

Finally, a similar thresholding step to the third step of Solution 1 is performed to achieve appropriate luminance profiles along the edges. This thresholding is in particular important close junctions. Recall from Section 4.2 that $z_{i,1}$ is set to zero if the adjacent curve has an opposite orientation. For a large λ, such thresholding is important to avoiding sharp transitions. This thresholding step is performed, in both directions of the list of $z_{i,1}$, by:

$$z_{i,1} = \begin{cases} z_{i-1,1} + \log(t) & \text{if } 10^{z_{i,1}} - 10^{z_{i-1,1}} > t; \\ z_{i,1} & \text{otherwise.} \end{cases}$$

As with the threshold in Solution 1, t should reflect the sampling of the discrete edge. The results presented here were produced with a threshold of 13% of the dynamic range (e.g. 13 if the luminance is defined from 0 to 100). For denser samplings, a lower threshold should be used (e.g. 3% to 20% for spacing between ca. 3 and 20 edge pixels), and vice versa (e.g. between 20% to 70% for spacing between ca. 20 and 70 edge pixels).

7.3.4 Applying Cornsweet surfaces to the original image

Now that the control meshes are created, the Cornsweet surfaces are rendered and applied to the image. The enhancement profiles are defined by the depth buffer, D, when rendering the surfaces. This can be efficiently performed by an off-screen rendering process with OpenGL or any other renderer. Since there were no requirements on performance, my MATLAB implementation uses the naïve recursive evaluation of the NURBS surfaces.

The NURBS surfaces are ensured to exactly align with the boundaries defined by the cubic B-spline curves defined by Q_i. Recall that corners $\hat{Q}_i$ are modelled as triple knots, forcing the curve to interpolate $\hat{Q}_i$. Thus, mesh control points $(\hat{P}_{i,1})$ associated with corners are also modelled as triple knots, forcing the surface to interpolate $\hat{Q}_i$.

The resulting discretised surface will not correspond exactly with the input edge. There will be a small number of pixels that are incorrectly categorised. That is: a pixel on the 'wrong' side of the edge may be adjusted, when it should have been left untouched; or a pixel on the 'correct' side of the edge may be left untouched when it should have been adjusted. Pixels on the wrong side of the curve simply have their adjustment set to zero. They are identified by considering whether the vector between the pixel and its closest curve pixel has the same direction as the normal vector of that curve point. Untouched pixels on the correct side of the curve are defined by bilinear interpolation to the closest drawn pixels. This is sufficient because only a small number of pixels are affected and the bilinear interpolation gives a result very close to the accurate value.

By directly adding the enhancement profile onto the image, hard step edges will be introduced. However, edges in photographs are not typically hard. Edges are smoothed by the camera via lens blur. If a lossy image format, such as JPEG, is used, compression artefacts might also occur. An anti-aliasing mask M is therefore defined to smooth the Cornsweet edges. In the enhancement image, pixels close to the input edges are blurred with a Gaussian filter (filter size 3; $\sigma = 1$). This neighbourhood is defined by the binary input edge image, dilated with a 3×3 structuring element composed of ones.

Dilation is a morphological operation typically used for probing and expanding elements defined in a discrete binary domain. The dilation of A by the structuring element B is defined by:

$$A \oplus B = \bigcup_{b \in B} A_b.$$

Note that an accurate[16;55] anti-aliasing mask will not contribute much for this problem. The reason for this is that the alpha matte is effectively turned into an averaging filter, as differences on the edges in the enhancement image (approximately λ), are much larger than the original differences encoded in the alpha matte. When this alpha matte is scaled to fit the new Cornsweet edges, these original differences turn negligible. Finally, I note that aliasing is a general problem when applying Cornsweet profiles in photographs, as they are defined across hard edges.

The output luminance Y_o is computed by modifying the original luminance Y as follows:

$$Y_o = 10^{D+\log_{10}(Y+1)} - 1 \times M.$$

The image is transformed from xyY to RGB to produce the final output.

7.4 Results and comparisons

The framework was implemented in MATLAB and is therefore not optimised for computational efficiency. Consequently, 1 mega pixel images currently require a few minutes of processing time. However, the performance would decrease significantly in a programming environment better suited to spline operations, as demonstrated in Chapter 5.

In Section 7.2.2, inherent limitations of the unsharp mask were established. In this section, I demonstrate that the related methods aiming to model the Cornsweet effect in photographs, all of which employ the unsharp mask, also face the same problems. The most related method, as it only requires a single input image, is the approach by Trentacoste et al.[101] (see Figure 7.14 for comparisons). The scan-line plots in Figure 7.14 show that the method of Trentacoste et al. saturates where neighbour edges are too close and when edges are curved. Moreover, their profiles do not always level out to the original luminance, whereas our solution does. A question that arises is whether the avoidance of these saturation artefacts has any influence on general photographs. This question is answered in Section 7.6.

Our framework is well suited for reference-guided enhancements since each mesh control point can be independently scaled. This is demonstrated by the textureness measure (Section 7.3.3) which increases enhancement magnitudes in areas of high textureness. Other uses of such

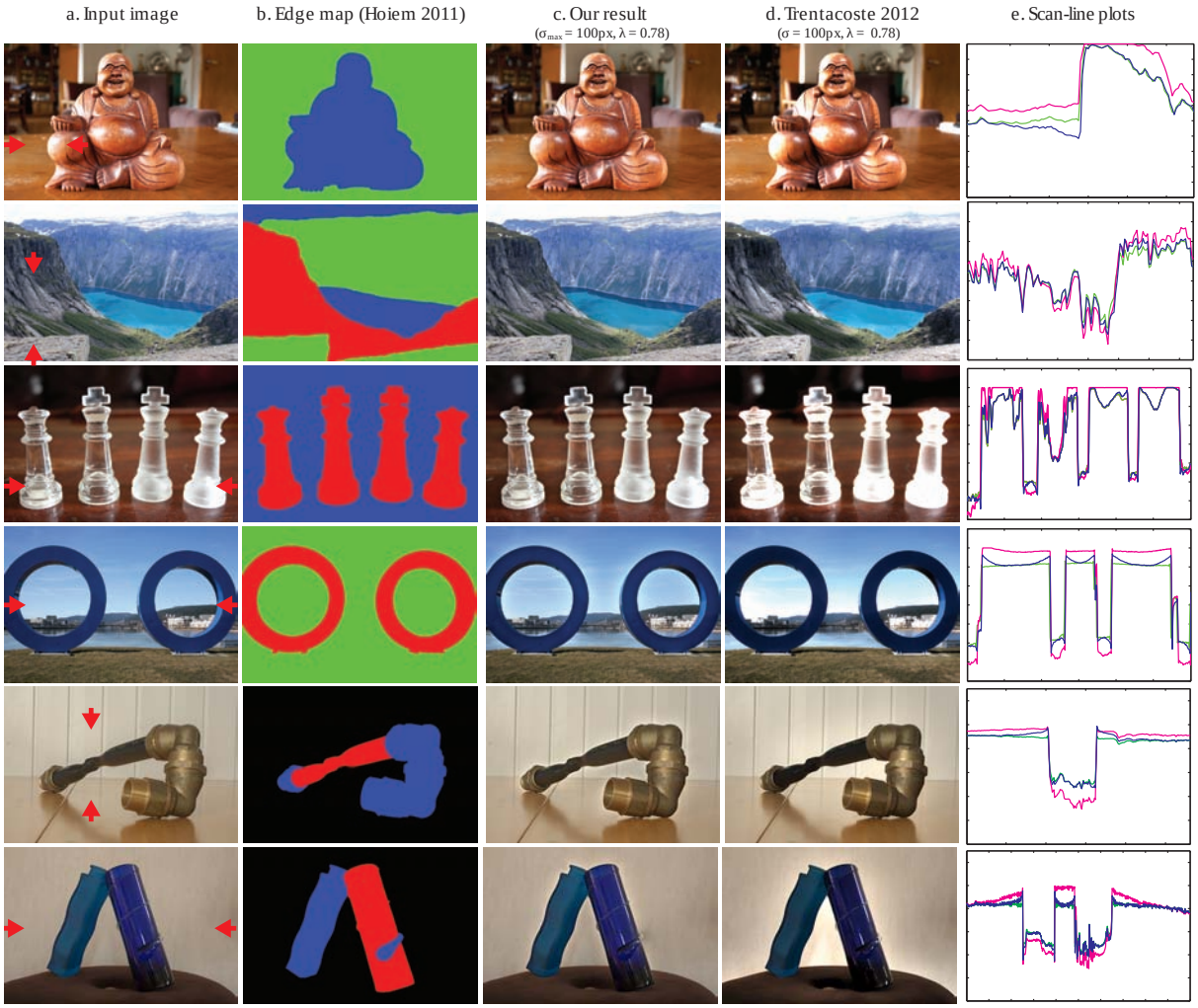


Figure 7.14: (c) Results of our method with mesh extents $\sigma_{\max}$ and with the value of λ computed according to the method of Trentacoste et al.[101]. Column (d) shows the corresponding results using the edge-aware enhancement of Trentacoste et al. (e) The scan-line plots (blue curves: our method, magenta: Trentacoste et al., green: input image) are plotted in luminance between the red arrows in (a). These plots show that the method of Trentacoste et al. saturates where neighbour edges are too close and when edges are curved. Moreover, their profiles do not always level out to the original luminance, whereas our solution does.

guided enhancements include the use of depth maps[62;87;39] and contrast recovery from HDR tonemapping[52].

In Figure 7.15, the textureness measure is complemented with an additional depth difference measure, where adjacent regions of large difference in depth values are emphasised. This effectively increases enhancements between foreground and background objects. Note that Luft et al.[62] also target depth-based contrast enhancement, inspired by neo-impressionism (Figure 7.4). However, as they employ the unsharp mask to solve this problem, they produce the same artefacts as the standard unsharp mask operator.

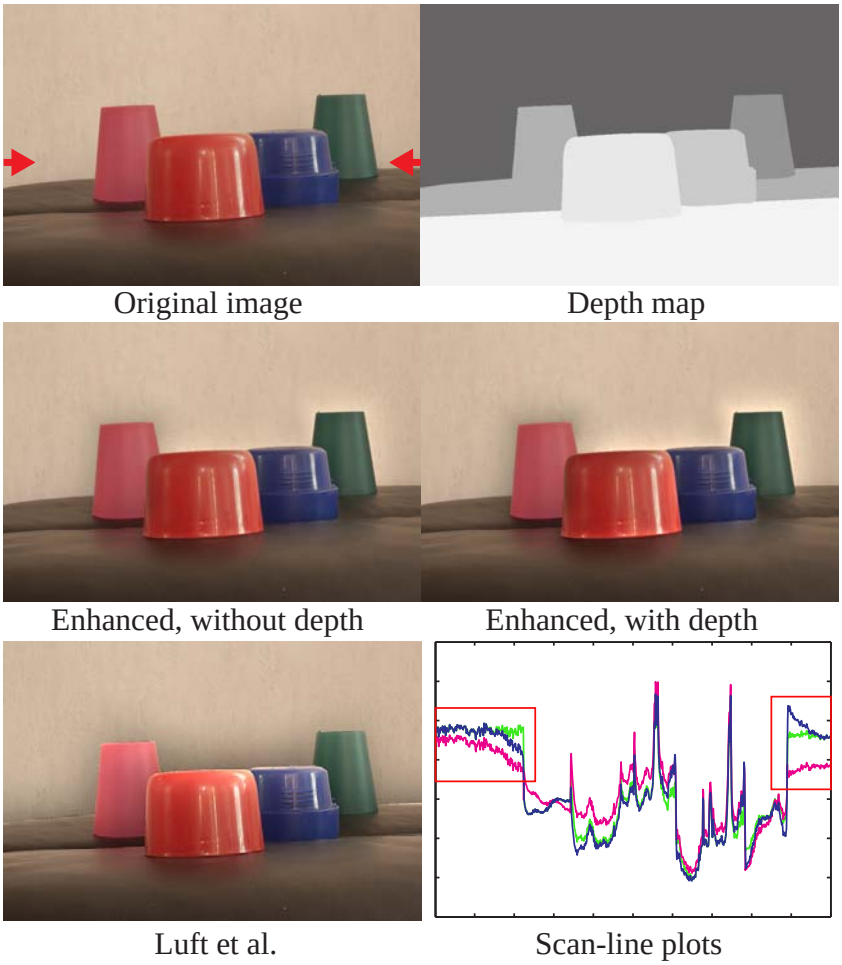


Figure 7.15: Reference-guided enhancement with depth. Object boundaries of large differences in depth are emphasised. Note that the enhancements are amplified to demonstrate the difference. The scan-line plots (blue curves: our method, magenta: Luft et al., green: input image) are plotted in luminance between the red arrows shown in the original image.

Krawczyk et al.[52] employed Cornsweet-style countershading to bring back lost contrast from HDR tone mapping. As with the related methods aiming to model Cornsweet profiles they employ the unsharp mask. Thus, saturation artefacts can occur (Figure 7.16). Such artefacts are typically not as strong as those produced by Luft et al. and Trentacoste et al. since the size of the Gaussian kernel is adjusted for each level of the image scale space, according to the reference image (Section 7.2). Additionally, clipping in luminance was avoided with a post-processing step; however, the correct Cornsweet profiles were not recovered, as highlighted in the scan-line plots in Figure 7.16. Finally, while it is demonstrated that the our method models Cornsweet profiles better than the method of Krawczyk et al., I do not argue that our method better recovers lost contrast from HDR tone mapping. In fact, Krawczyk et al. did not confirm whether their method does recover such contrast. Future work is therefore needed to verify whether Cornsweet-style enhancements do recover such lost contrast from HDR tone mapping.

Figure 7.17 illustrates the difference between general image processing methods and selective Cornsweet-style contrast enhancement like our method: general methods aim to *generally* alter

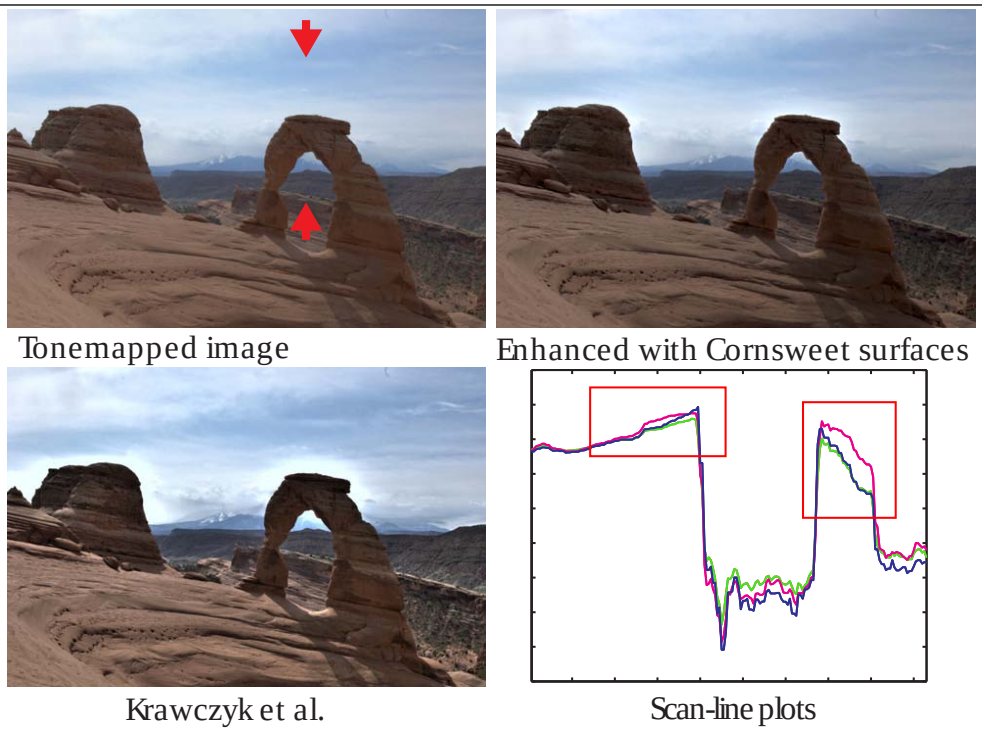
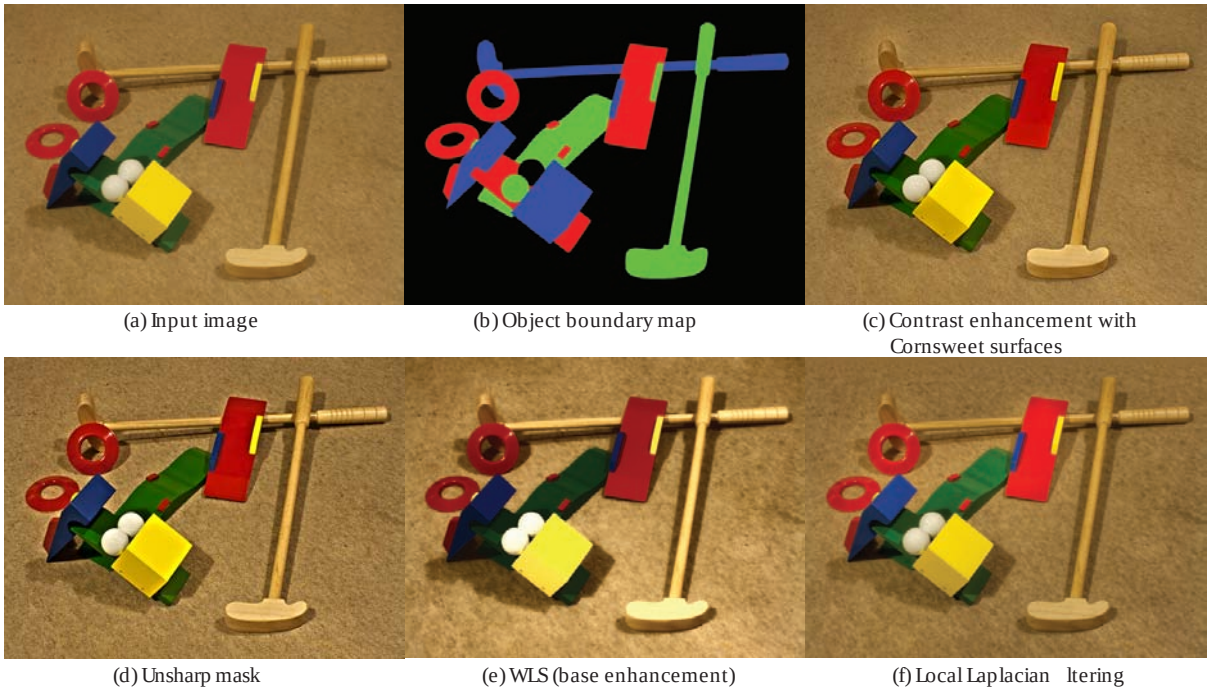


Figure 7.16: Modelling Cornsweet profiles with an HDR image. The Cornsweet surfaces are computed in the original HDR image, which can then be tonemapped. The scan-line plots (blue curves: our method, magenta: Krawczyk et al., green: tonemapped image without Cornsweet enhancement) show that the our method more accurately models the Cornsweet profile than the method of Krawczyk et al.

the input image, while our method aims to apply the Cornsweet effect to a selective set of edges, without altering the original look of the image.

7.5 Discussion

The Cornsweet surfaces produced, via NURBS, can be modelled in both spatial and luminance domains in other ways than presented in this chapter. For example, we decided to model the sign of a given surface by comparing the difference in luminance of the two given image regions (Section 7.3.3). Thus, we aimed to model the Cornsweet surfaces to adapt to the Cornsweet illusion, which states that the luminance of the brighter region should be increased, and vice versa. In some cases, however, other aspects can be incorporated to such modelling decisions. In Figure 7.15, for example, the leftmost cup is modelled with a negative outgoing profile, while the other three cups are associated with positive outgoing profiles. In this example, some would argue that spatial or depth coherence, in addition to following the Cornsweet profile for each edge separately, should dictate the sign of the surfaces. Given that decisions can be defined algorithmically, such algorithmic changes are trivially achieved with the proposed framework, since each surface is explicitly modelled. In the setting of depth coherency (Figure 7.15), the sign of the enhancements can be dictated by depth values, and not luminance, similar to the



(a) Input image (b) Object boundary map (c) Contrast enhancement with Cornsweet surfaces

(d) Unsharp mask (e) WLS (base enhancement) (f) Local Laplacian filtering

Figure 7.17: Comparison with general state-of-the-art image processing methods (d,e,f). While such methods provide a general enhancement of the image, altering the overall look, our method (c) *selectively* enhances a given set of edges defined by object boundaries (b) with the Cornsweet effect whilst preserving the original feel of the image.

approach by Luft et al.

Cornsweet NURBS surfaces are defined with C^0 creases along points associated with sharp corners and junctions, represented as triple knots in the knot vector. While this can be a visible artefact in flat shaded images, such as Figure 7.11, I have not noticed any related artefacts in photographs, as they are disguised by high-frequency image signals. However, for extreme enhancements, employed in artistic applications (Chapter 8), C^0 creases have been noticed. Alternative surface representations, such as subdivision surfaces, can provide smooth surface representations for meshes of both quads and triangles, as demonstrated in Chapter 5. Subdivision surfaces can therefore be employed to avoid C^0 creases. As mentioned in Section 7.3.3, however, the our approach is not directly supported by standard subdivision schemes. Thus, in order to adapt the framework to subdivision, some design decisions should be revised. For example, the fitted curves can be modelled as quadratics, and not cubics, in order to model rational Doo-Sabin subdivision surfaces.

7.5.1 Limitation

Our method is not applicable for edges and regions where the Cornsweet effect is negligible. Such edges are found adjacent to exceedingly narrow regions. Dooley and Greenfield[25] have demonstrated that perceived contrast is *not* altered with profiles of less than 0.2 visual degrees (Figure 7.2 in Section 7.1). Thus, edges surrounding objects spanning only a few pixels in

Figure 7.18: Experimental design.

the image, such as leaves in distant trees, cannot be enhanced with the Cornsweet effect. Approaches aiming to model the Cornsweet effect, such as our method, are most effective where sufficiently large regions are well defined adjacent to relatively hard edges, as the Cornsweet profile can be properly defined in such regions.

7.6 Evaluation

Note: the experimental design and analysis were conducted by Erik Reinhard and Tania Pouli.

In Sections 7.2 and 7.4, I demonstrated that the unsharp mask and related methods create satura-

tion artefacts if edges are not sufficiently straight or adjacent edges are too close. The proposed solution explicitly deals with these problems and does not create such artefacts. A question that arises from these contributions is whether they matter for Cornsweet-style enhancement of general photographs. To answer this question, we conducted a psychophysical experiment comparing the relative merit of our framework and the method of Trentacoste et al.[101]. We only compared with Trentacoste et al. since their method is the only approach aiming to model the Cornsweet effect from single images.

7.6.1 Design

We designed a 2-alternative forced-choice experiment, comparing the two methods at similar levels of enhancement. For each trial, participants were asked to select the *best enhanced image in their opinion* between corresponding results from both methods. The unprocessed image was also shown as a reference to enable participants to determine the extent and quality of each enhancement (Figure 7.18).

Ten different images were processed with each method, using 4 values of enhancement strength $\lambda = (0.5, 1.0, 1.5, 2.0)$ for our method, resulting in 40 trials per participant. The images used were selected to depict a wide variety of objects and scenes, including both indoor and outdoor settings. We opted against including portraits in this comparison as different criteria may be used when assessing imagery with people, which could bias the results. The images we used are shown in Figure 7.18.

To select comparable enhancement strengths between the two methods, a series of images with varying λ values were produced for the method of Trentacoste et al. Matching pairs for each λ were selected such that they minimised the difference between global contrast within the two methods for each image. The enhancement extent was fixed to 4.2 degrees of visual angle (assuming a 0.5m viewing distance), which corresponded to $\sigma = 250$ pixels in our method.

The experiment was designed in MATLAB using the Psychophysics Toolbox and performed on a laptop display with the resolution of 1366×768. 17 participants (7 female, 10 male, age mean 43.6; standard deviation 20.0) took part in the experiment, all reporting normal or corrected to normal vision and normal colour vision. Participants were positioned so they could comfortably view and use the laptop (approximately 0.5m viewing distance), but viewing position and distance were not precisely controlled as we were interested in measuring viewer preferences rather than perceptual phenomena. The order of stimuli as well as the position (left, right) of each method were randomised.

7.6.2 Analysis

In a total of 680 trials, our method was selected 464 times, leading to a normalised mean of 0.68 (standard deviation 0.12). The results were analysed with a one-way ANOVA, finding that our method led to a significantly preferred enhancement overall ($\mathrm{F}(1, 32) = 51.04, p < 0.001$).

To further explore the results, we performed a multi-way ANOVA on the enhancement strength λ, the image and the participant, as well as their interactions. We found that the selection of λ

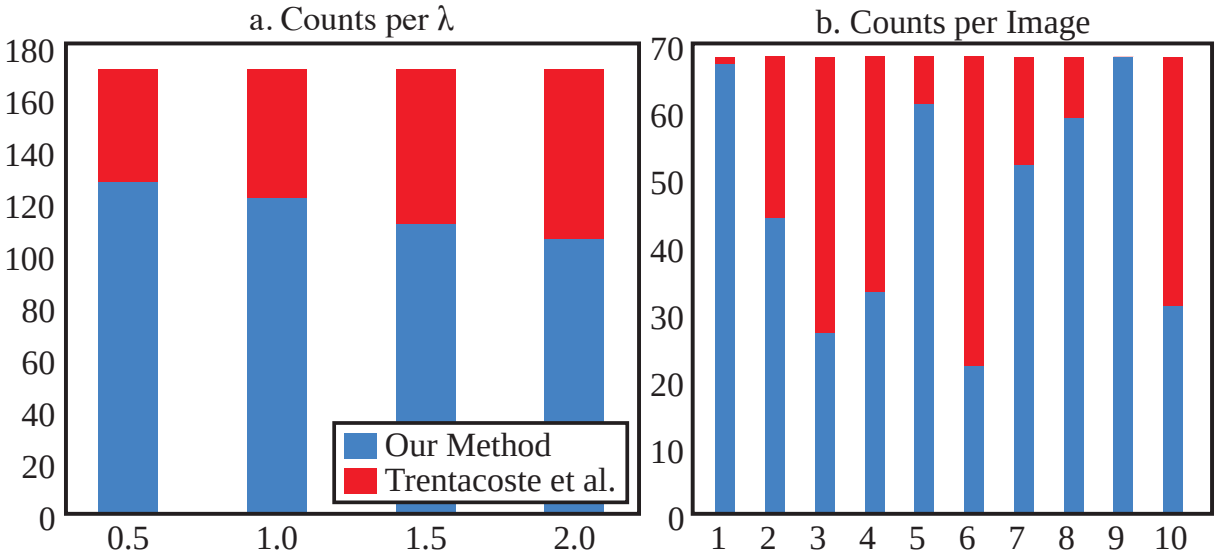


Figure 7.19: Detailed results for our experiment. The stacked bar charts show how often each method was selected, grouped by λ (left) and by image (right). The number of each image corresponds to the numbers given in Figure 7.18.

led to a significant effect in terms of enhancement preference ($\mathrm{F}(3, 676) = 4.88, p < 0.005$), although post-hoc analysis indicated that this was only the case between $\lambda = 0.5$ and $\lambda = 2.0$. Although for all λ values our method was selected more, increasing the strength of the enhancement seemed to progressively reduce the preference towards our method. The detailed results are shown in Figure 7.19(a).

Similarly, significant differences in participant choices were found between the different images ($\mathrm{F}(9, 670) = 37.48, p < 0.001$). Specifically, we found that our method performed significantly better in images where contrast was already high. In such cases, our adaptive adjustment allowed for the contrast to be enhanced where possible, whilst preserving local details. Compared against high contrast regions in the same images enhanced with the method of Trentacoste et al., which were likely to become over- or under-exposed, effectively losing local detail (see Figure 7.14). Detailed results can be seen in Figure 7.19(b).

Although a significant effect was found for participant number, post-hoc tests indicated that most participants responded similarly with only few exceptions. Detailed results for each participant can be seen in Figure 7.20(a) and (b). No interactions were observed between participants and enhancement strength λ, suggesting that the preference for softer enhancements is universal. In contrast, the selection of image led to a significant interaction both with participants and λ values ($\mathrm{F}(26, 653) = 2.94, p < 0.001$ and $\mathrm{F}(13, 666) = 1.97, p < 0.001$ respectively), suggesting that the preferred amount of enhancement varies per image.

Overall, our analysis shows that our adaptive enhancement method leads to contrast enhancements that are preferred by viewers. We have found that participants generally prefer softer enhancements, which aligns with the visual effect obtained with our method: only salient edges are enhanced with the Cornsweet profile being gently embedded into the image.

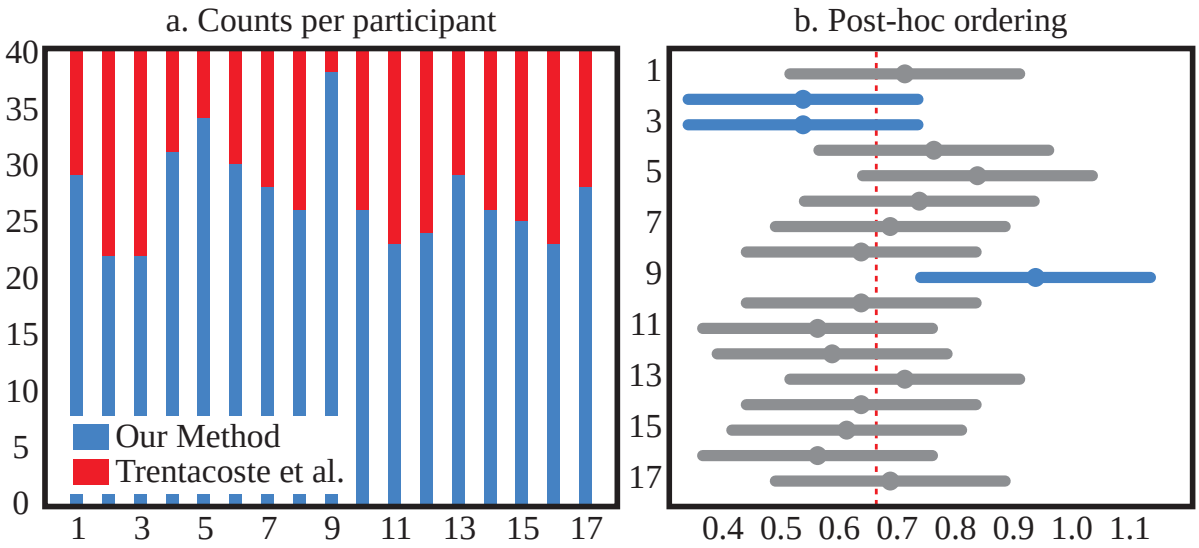


Figure 7.20: Per-participant results. The left bar charts show how often the two methods were selected by each participant, while the right plot shows groupings for the participants as determined by post-hoc analysis. Participants highlighted in blue were significantly different from each other, but not from the main group. As such, they were not considered as outliers. Note that the dashed line in red indicates the overall mean.

7.7 Future work

The selective nature of the contrast enhancement algorithm makes our framework suitable for future work in perceptual psychology. Our experiment (Section 7.6) indicates that the aesthetic value of the images we produce is not worse than that of previous methods. Thus, previous studies can be rerun to accommodate the advantages of our method. For example, the experiments conducted by Trentacoste et al. on whether countershadings are objectionable or not can include additional dimensions. Our algorithm produces absolute renderings of the input parameters (σ and λ), while filtering-based methods can only provide relative enhancements (Section 7.2). Thus, one can now ask questions like whether countershadings of widths of σ pixels (and not filter widths) and enhancement magnitudes of λ luminance (and not relative scalars depending on image content) are objectionable. Additionally, an interesting future investigation is to establish to what extent Cornsweet enhancement can alter the perceived contrast in photographs.

Our experiment suggests that the preferred adjustment varies per image. For example, it might be that countershadings between two regions of similar colours are more preferred over two regions of very different colours (or vice versa). An interesting avenue for future research is whether such countershading can be (machine) learned.

7.8 List of contributions

- I have presented the first approach to apply vector-centric image processing to contrast enhancement.
- Our solution does not encounter saturation artefacts introduced by previous approaches.
- In user trials, our solution is significantly preferred over a state-of-the-art (selective) contrast-enhancement method.

Chapter 8

Spline-based artistic image processing

This chapter presents research that has been published in the following paper:

Cornsweet surfaces for selective contrast enhancement

In this chapter, I continue the investigation of surface modelling in the setting of image processing. While the previous chapter focused on automatic processing of the image, I now present various methods where the artist is included in the algorithmic loop. I do not present any new techniques or explore new *technical* problems. Instead, I present novel artistic imagery using the frameworks of Chapters 5 and 7.

A key difference to traditional pixel-based methods is that my approach is edge-centric. That is, edges are manually selected in the image and associated with any given *effect*. Instead of modelling colour gradients for creating vector graphics (Chapter 5), the artist models various artistic effects out from image edges. In contrast, pixel-based methods typically operate on image regions or colour differences between pixels. I present various applications where an edge-centric approach is more sensible than such pixel-based solutions (Section 8.2). Note that edge-centric image editing has been proposed in the literature. Such related work is compared with my approach in Section 8.3.

8.1 User-assisted spline-based artistic imaging

In this section, I present the various types of user interactions available using my surface modelling frameworks in the setting of photo manipulation. This is an informal discussion of how the input to my frameworks can be defined. I do not argue or discuss whether the chosen type of interaction is better than other methods. Such questions are formally studied in the field of human-computer interaction. As my work is aimed towards new methods and algorithms in computer graphics, such questions are out of scope in this dissertation.

8.1.1 Edge selection

The primary input for the type of user interaction presented in this chapter is image edges. Since it is time consuming for a user to manually select the exact pixels of the desired edge, many approaches have been suggested to speed up this process.

A typical solution, employed by many related methods including DCs, is to 'snap' user-defined strokes to the image edges. This snapping mechanism is based upon *active contours*[46]. In general, the active contour model attracts given splines or edges towards features of interest in the image. In the setting of selecting image edges, a selected spline or edge can be translated along the gradient of the image signal. This process is iteratively performed to place the user-selected edges on image edges.

The edges used in this chapter are defined in a similar process. The user input, defining a single edge, is a single thick stroke (Figure 8.1). The stroke is thick enough for the user to be able to cover the desired edge. The resulting discrete edge is defined by the connected discrete edge along high magnitudes of the gradient within the stroke.

The rationale for using thick strokes, and not pixel-wide strokes, is that it is a natural type of input for different kinds of interactions. For example, strokes with thickness of roughly the index finger can be more natural than thin pixel-wide strokes, especially for touch-based interactions. Additionally, the process is fully automatic after the stroke is defined and there is no need for the manual snapping process employed by some other methods including DCs. Again, I do not claim that an approach with thick strokes is necessarily better than other methods. The given advantages are merely the basis for my design decisions.

Active contours can be employed to place the edges, accurately, on image edges. This is achieved in two steps: extract a connected discrete edge from the thick stroke and align this edge to the closest image edge. (Step 1) The thick stroke is represented as a binary image with 1s at stroke pixels (0s everywhere else). This image can be transformed to represent thin strokes in several ways. In my implementation, I use MATLAB's **bwmorph** function (morphological operations on binary images) with the options 'thin' and 'Inf'. This operation removes pixels so that a region without holes shrinks to a minimally connected edge. (Step 2) This edge is snapped to the image edge by moving pixels along the gradient (similar to the particles in Solution 2, Section 4.2.4). In my implementation, this is only performed a few iterations (3) since I assume that the original stroke is close to an edge.

In addition to forming a thin discrete edge for each stroke, edges from the previous strokes must also be taken into account. More specifically, if the new edge overlaps with a current edge, the new edge is split into two edges. The point of the overlap then forms a junction point.

Finally, the new edges are fitted, creating B-spline curves needed for the creation of control meshes. The fitting procedure is the same as the method described in Section 7.3.2.

Other types of strokes that do not need to be converted into B-spline curves can also be employed. Recall the following termination criterion from Section 4.2.3: '[terminate the ray r_i when] r_i hits any barrier that constrains mesh extents in the image'. This 'barrier' can, for example, be a set of pixels defined by strokes. As these strokes are not converted to B-splines they can be represented as thick strokes (that is, no processing of the input stroke is required).

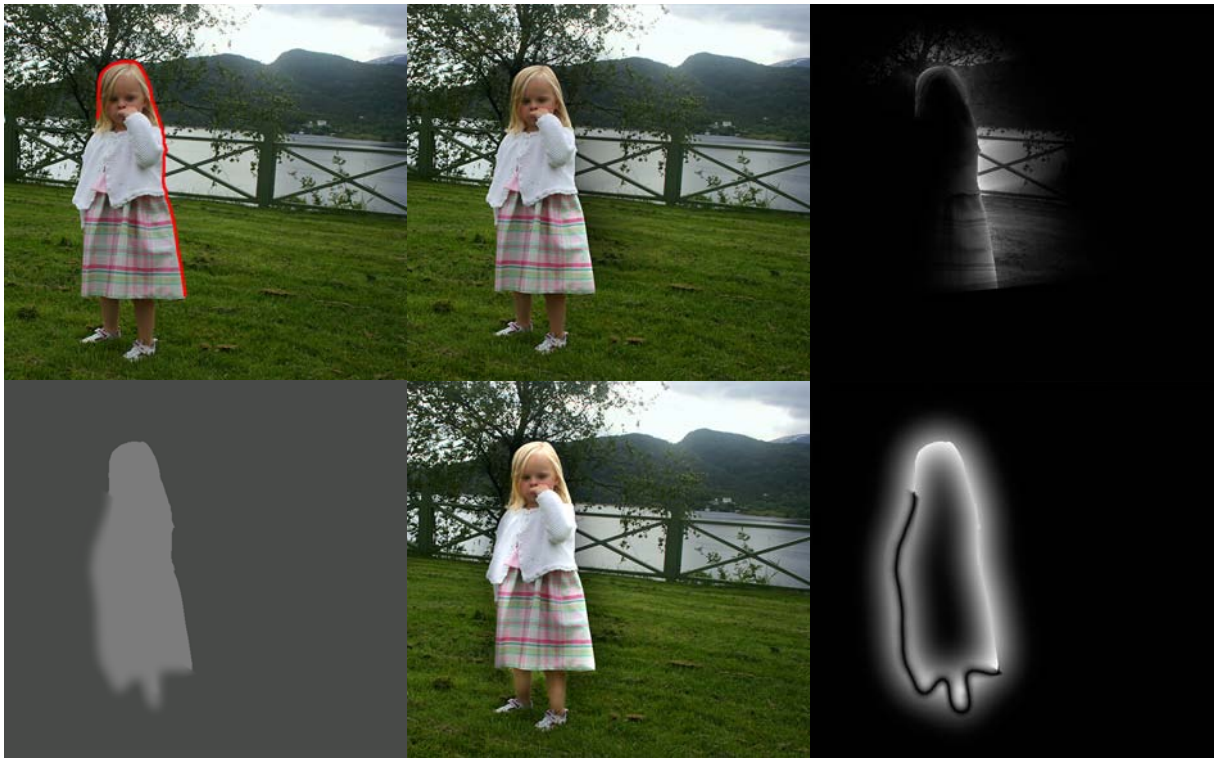


Figure 8.1: Edge selectivity with a thick red stroke related to a Cornsweet edge (top). In comparison, to achieve edge selectivity with the unsharp mask (bottom), image segments must be defined (bottom-left). Edge selectivity can be emulated by blurring hard edges in the segmented image. However, the rightmost images, showing the magnitude of the enhancements, demonstrate that full edge selectivity is difficult to achieve with the unsharp mask.

Figure 8.1 demonstrates the advantage of my edge-based approach over region-centric filtering in the setting of selective Cornsweet-style contrast enhancement. As the Cornsweet effect is associated with edges, and not regions, edge-based user input is most sensible. Edge selectivity can be emulated with filter-based methods, like the unsharp mask, by blurring out the hard edges in the segmented image, as demonstrated in Figure 8.1. However, this is not as convenient as directly selecting the edges and do not achieve full selectivity.

In the remainder of this section, I present ways to interact with photographs, for artistic purposes, using the frameworks presented in Chapter 5 (the shading curve) and Chapter 7 (Cornsweet surfaces).

8.1.2 Image manipulation with shading curves

Figure 8.2 demonstrates the shading curve in the setting of photo manipulation. The interaction with the image and its selected edges is the same as producing shading profiles for vector graphics. That is, cubic profiles are edited, representing how the given effect is propagated out from the edge.

To apply the image defined by rendering the subdivision surfaces, the same issues encountered

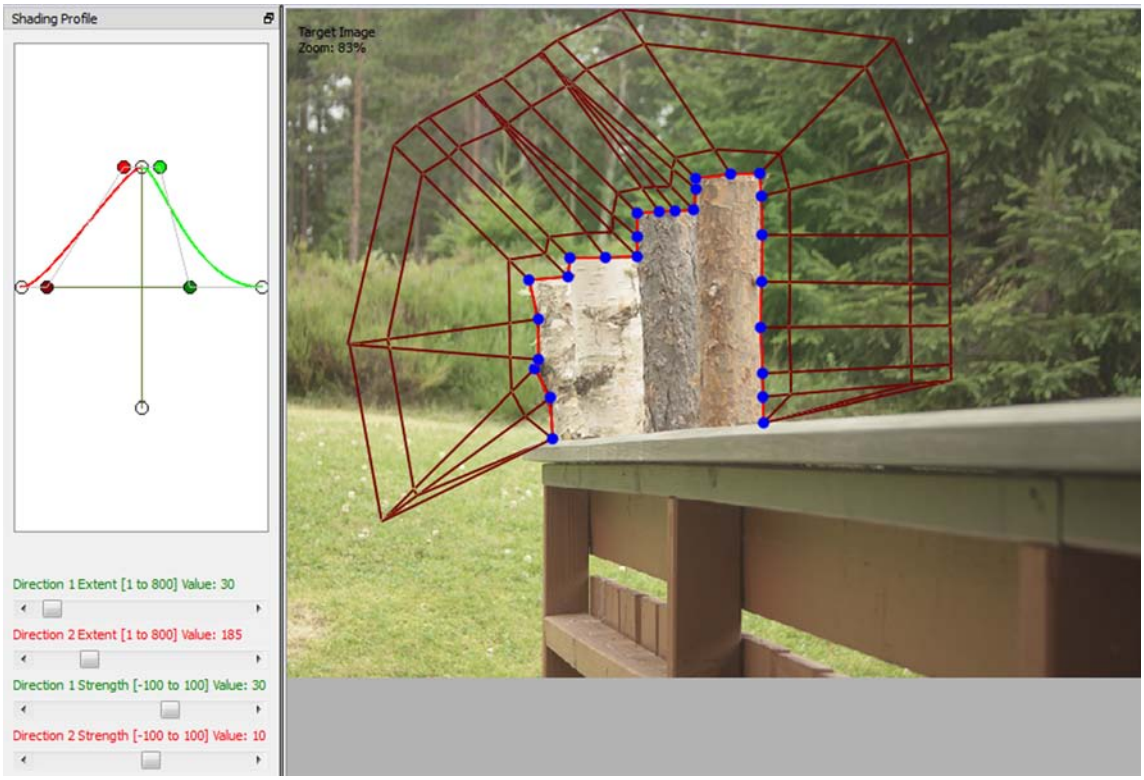


Figure 8.2: Adding a green-coloured profile behind the logs with the shading curve. The red spider-web is the control mesh projected to the image plane. On the left hand side, the shading profile (red curve) and the attributes related to extent and height are manipulated.

in Chapter 7 must be dealt with. That is, the resulting discretised surface will not correspond exactly with the image edge. A pixel on the 'wrong' side of the edge may be adjusted, when it should have been left untouched; or a pixel on the 'correct' side of the edge may be left untouched when it should have been adjusted. As with Cornsweet surfaces, this is solved by identifying incorrect pixels and by making corresponding corrections. If the pixel is on the wrong side, it is zeroed, and if the pixel on the correct side has not been painted, it is bilinearly interpolated.

8.1.3 Image manipulation with Cornsweet surfaces

A disadvantage with photo manipulation with the shading curve is that the artist is concerned with B-spline curves and control points that are only intermediate representations in the photo manipulation pipeline. The framework for defining Cornsweet surfaces (Chapter 7) is well suited to abstract the artist from these representations. Thus, the artist can be provided with a more natural type of input for photo manipulation with strokes and sliders. In order to fully abstract the user from control points and spline curves and surfaces, alternative methods to defining the shape, extent, and magnitude of the surfaces can be defined. In the following, I present ways to manipulate the shape of the surfaces and the extent and height of the meshes.

The **shape** of the NURBS surfaces is manipulated through their weights (Figure 8.3). Thus, in

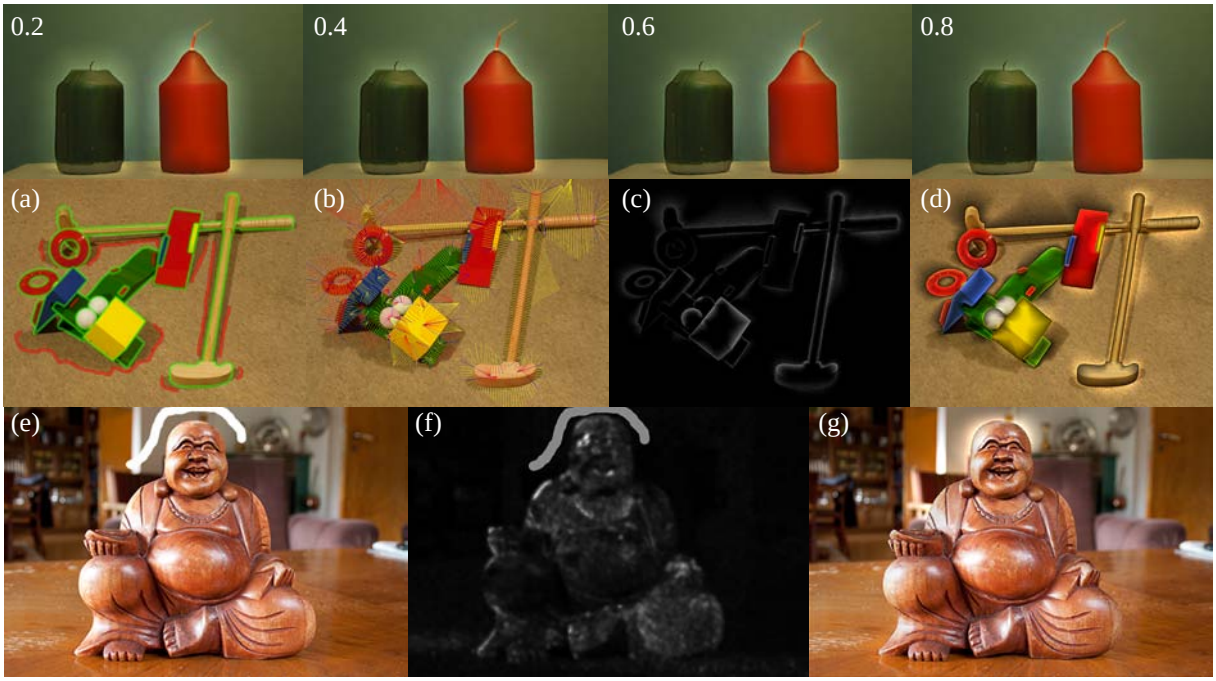


Figure 8.3: Modelling Cornsweet surfaces with slider-friendly input and brush strokes. Top row: globally adjusting the shape of the surfaces with NURBS weights (weights given in the top left corners). Middle row: locally adjusting spatial extents of the surfaces with strokes. (a) Input image with strokes (green: enhancement edges, red: barrier strokes). (b) Resulting extents of the normal vectors of the control points (red: negative enhancements, yellow: positive enhancements, blue: junctions). (c) Magnitude of the enhancement image and (d) final result with extreme enhancement. Bottom row: locally adjusting the magnitude of the enhancements by augmenting the textureness measure. (e) Input image with a user stroke. (f) Augmented textureness measure. (g) Resulting enhancement.

addition to the current global parameters, λ (magnitude or strength) and σ (extent), the weight w can be used to model shapes other than the Cornsweet profile. Note that the Cornsweet profile was modelled as a quadratic function in Chapter 7. While there are no restrictions to the degree of this profile, I have found the quadratic profile sufficient for the showcase applications presented in this chapter. The weight related to the second control point defining this quadratic was set to 0.4 for all of these applications. Finally, some applications, such as shading, might require the shape to be modelled locally in the image. While I did not implement this feature, I imagine this being achieved by having the user specify selected regions where a given weight or profile is applied.

Manipulation of **local extent** of the meshes is achieved with the barrier strokes discussed previously in this chapter. Recall that Solution 1, which creates the meshes of Cornsweet surfaces, supports this feature by incorporating barrier strokes as a termination criterion. Thus, such barrier strokes serve as constraints to the spatial extent of the resulting Cornsweet surfaces. This is demonstrated in Figure 8.3(a-d), where the barrier strokes, shown in red in Figure 8.3(a), are placed on shadow boundaries to ensure that the Cornsweet enhancement is not crossing such edges.

Manipulation of the **height** of the meshes can be achieved by altering the textureness measure used in Chapter 7. Recall from Section 7.3.3 that the textureness measure represents high-frequency changes in the image signal. To account for visual masking, $z_{i,1}$ is scaled according to the local textureness of the related Q_i. Thus, this textureness measure can be used to scale, locally, $z_{i,1}$ and can naturally be used in other settings than to account for visual masking.

An artistic application of implicitly altering the textureness measure is shown in Figure 8.3(e-g). A white stroke is drawn onto the original image (e). This stroke can then be included in the texture measure, as shown in (f). Note that the values are scaled to be defined between 1 and 1.2. Thus, the z coordinates of the related mesh control points $P_{i,1}$ in the local neighbourhood of the stroke are scaled by approximately 1.2. In this example, the alteration (g) produces a strengthened halo-like enhancement in that neighbourhood (Buddha's head).

8.2 Applications

In this section, I present various artistic applications for my frameworks in the setting of artistic photo manipulation. These include enhancement in alternative colour channels, authoring of shade and light in photographs, and masking extreme enhancements with artistic filters. I note that this list is not exhaustive and other applications to spline-based photo manipulation can exist.

8.2.1 Enhancement in alternative colour channels

Figure 8.4 shows Cornsweet enhancement employed in other channels than luminance. Enhancement in colour (RGB space in Figure 8.4) can be employed to model artistic coloured halos, such as the golden halo surrounding Buddha, and more subtle effects, such as the green halo behind the flower enhancing the surrounding greenery.

Manipulation of the chroma and saturation channels is often used to enhance the colourfulness of the image. In this setting, Cornsweet surfaces are well suited for selectively increasing colourfulness, whilst retaining the rest of the image. In Figure 8.4, chroma and saturation are selectively increased to visually separate the Buddha model from the table, and to create an artistic look of the petals of the flower and the surrounding background. Similarly, Cornsweet-style enhancements can be used to lighten or darken selected parts of grey-scale images (Figure 8.4(bottom)).

8.2.2 Manipulating shade and light in photographs

Shading curves are well suited for manipulating shade and light in photographs. Such manipulation is best achieved with *intrinsic images*[3]. The intrinsic image representation is a decomposition of the original photograph, describing scene properties for each pixel, including illumination, reflectance, and optionally orientation and depth. While this decomposition of the

Figure 8.4: Cornsweet surfaces employed in colour, chroma, saturation, and grey scale channels.

image was originally defined for problems within computer vision, it has been demonstrated useful for manipulation of shade and colour in photographs[9].

Altering the shading of the image can be achieved by replacing or modifying the shading component of the intrinsic image. This is demontrasted in Figure 8.5 where shading defined by shading curves replaces the shading image of the intrinsic image. Images from the MIT Intrinsic Images dataset[35] were used for these demonstrations.

8.2.3 Masking extreme enhancements with artistic filters

Artistic filters typically add additional frequencies to the image. Thus, the effect of visual masking, where high frequencies visually hide changes of low frequencies, ensures that extreme enhancements can be added to the image without being objectionable. In art, this effect was experimented with during the neo-impressionist era (Section 7.1).

Figure 8.6 demonstrates this application. The flower and the girl examples create a cut-out effect, where the central objects are better separated from the background with an extreme Cornsweet-style enhancement. In the bottom sphere example, an adjustment is made to add light behind the object. This addition is better hidden with an artistic filter (due to visual masking).

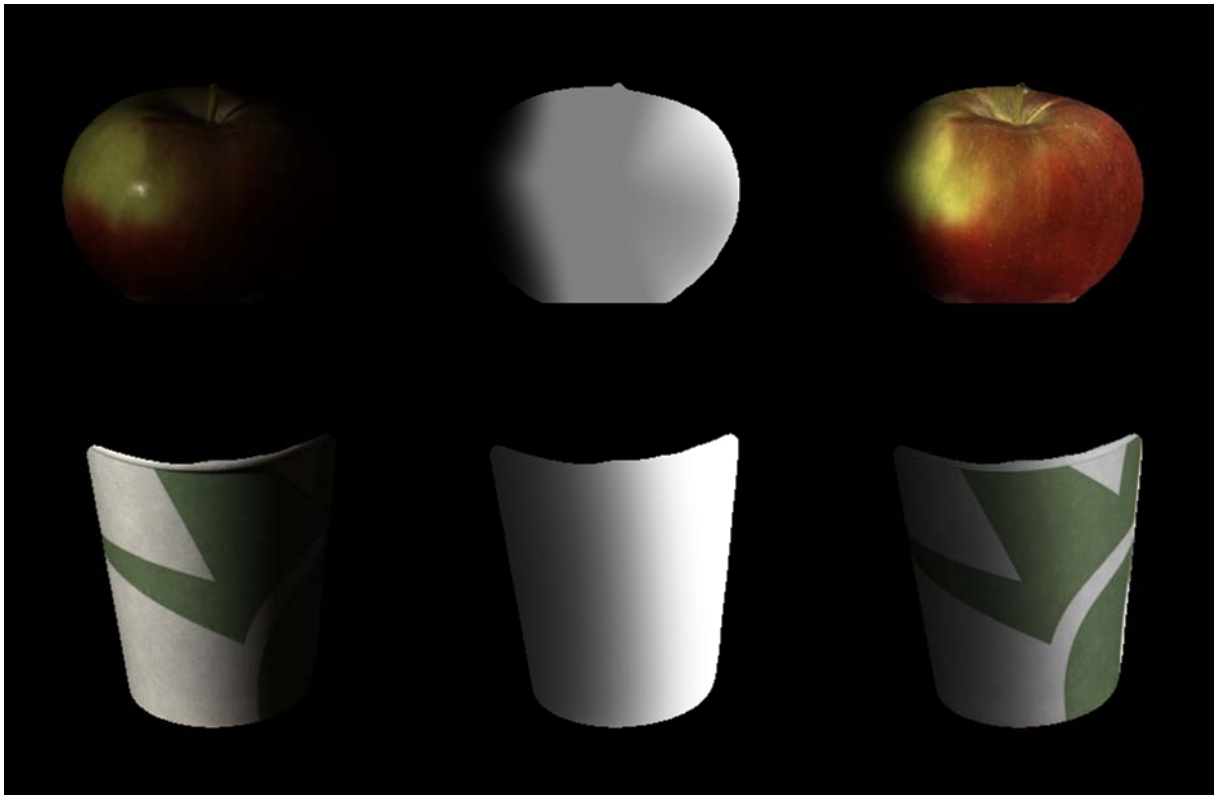


Figure 8.5: Drawing shade and light with shading curves on intrinsic images. Left: original intrinsic images. Middle: shading images produced with shading curves. Right: resulting intrinsic images, using the middle images as shading images.

8.3 Related work

The methods I have proposed in this chapter are not the first targeted towards image edges. In this section, I discuss previous research on edge-centric image editing.

Elder and Goldberg[27] have argued that image editing in the contour domain can provide several advantages compared to pixel-based editing. According to Elder and Goldberg, pixels are artefacts of the sensing process and have no relation to the underlying image content. Due to these issues, pixel-based editing provide challenges in many high-level editing applications like removing specular reflections or deleting image features. By contrast to pixels, edges and contours, they argue, are directly related to image content as they differentiate image boundaries, surface creases, cast shadow boundaries, and other significant visual 'events'. From these observations, they hypothesise that many image editing operations may be facilitated if the user can influence the image by directly manipulating its edges.

To achieve such editing, three issues must be dealt with[27]:

1. Edges must be extracted from the image. In contrast to the edge detection problem for Cornsweet-style contrast enhancement (Section 7.3.1), one would like to identify as many possible edges in the image, and not only boundary edges. Identifying many edges will make a solution for the 2nd issue more accurate (see ref. 73, Section 4.3).

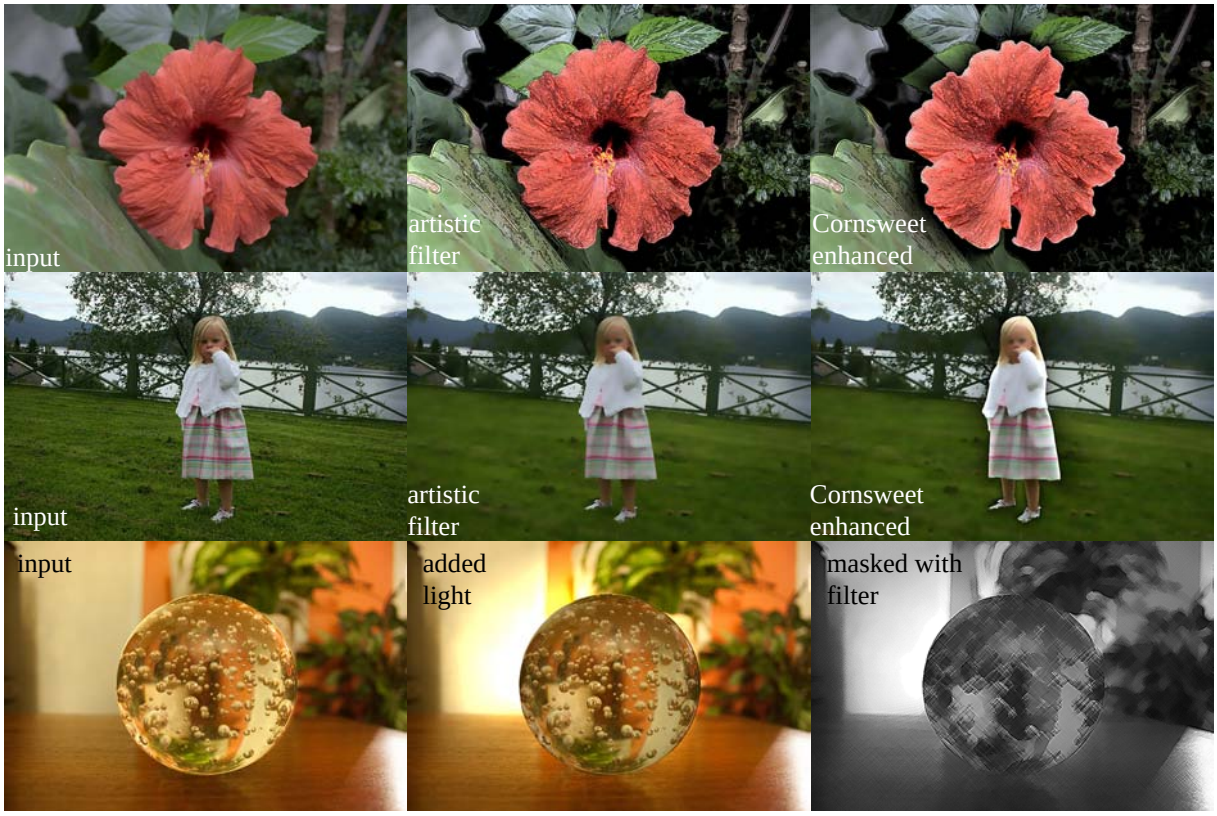


Figure 8.6: Applications of Cornsweet surfaces in the setting of artistic filtering. Due to visual masking, extreme enhancement can be employed without being objectionable.

2. Given these edges, associated with greyscale or colour values, the original image should be reconstructed. Elder and Goldberg solve this problem by defining the greyscale values at the edges as boundary conditions for Laplacian diffusion (that is, solving the equation $\Delta f = 0$). This approach has later been extended by Orzan and colleagues[73] (diffusion curves) to support colour images.

3. To perform image editing operations, edges should be grouped. With extended edges, the user is, according to Elder and Goldberg, not required to perform as many manipulations to achieve a desired effect.

The above approach represents the class of methods aiming to vectorise images. Image edits therefore operates on the vector image. By contrast, my approach is only *vector centric* in the way a vector image, produced by rendering spline surfaces, supplements the original image. Thus, the original image remains a pixel-based image with my method.

While the two approaches are similar since they are both edge-centric, their differences make them suitable for different types of scenarios. The method of Elder and Goldberg (vectorisation) provides all the advantages of vector graphics, like scalability. On the other hand, vectorising images is prone to error, especially at high-frequency image regions (see ref. 73, Section 6.2). By contrast, my approach avoids this issue by blending an intermediate vector image with the original pixel image. As I have demonstrated in Section 8.2, such an approach is particularly useful when a given effect, like the Cornsweet effect, is modelled along a sparse, selective, set

of image edges. In these situations, I therefore demonstrate that there is no need to tackle the challenging problem of vectorisation.

Finally, I note that my method can be associated with methods aiming to paint effects along brush strokes in images. Most related to my method is the approach by McCann and Pollard[65]. They propose two types of brushes for image manipulation: the edge brush and the gradient brush. The edge brush is defined as a diffusion curve (their approach was presented in the same session as diffusion curves[73] at the SIGGRAPH 2008 conference). The gradient brush tool lets a user select an existing edge in the image. The gradient along this brush is captured and rendered along new brush strokes, allowing the user to extend and duplicate edges in the image. This gradient brush stroke is rendered via the Poisson equation ($\Delta f = g$; where g is the gradient captured from the original edge). Similarly to my method, the rendered image is blended with the original image using a selected blending mode, like additive blending.

The main difference to my method, not comparing the rendering procedures (Poisson solution vs. spline surfaces), is that only new strokes are supported; adding effects to existing edges is not allowed. Thus, the aspects of their method related to my work are more targeted towards image painting rather than manipulation of existing content. Note that previous work[76] on the Poisson equation targets image manipulation, but is more targeted towards blending and cloning operations rather than edge-centric gradient editing.

8.4 Contribution

I demonstrate novel approaches for user-assisted artistic image processing to render several types of effects out from image edges. Such effects are challenging to achieve with previous methods.

Chapter 9

Conclusions and future work

I have presented novel methods for drawing 2D vector graphics and I have proposed novel approaches to vector-centric image processing. The novelty of the proposed methods for drawing 2D graphics lies in how users specify colour gradients explicitly and how they can work with gradient meshes of arbitrary manifold topology. The novelty of the proposed approaches for image processing lies in how the vector representation of the effect is *supplemented* with the original pixel image. By contrast, previous vector-centric methods rely on accurate conversion to a pure vector representation of the image which is a challenging problem.

Colour interpolation with vector graphics has been previously concerned with two approaches: diffusion curves and gradient meshes. On one hand, diffusion curves is related with a more natural type of input (freeform curves) compared to the input to gradient meshes (control meshes). In terms of general drawing, it is obvious that drawing with freeform curves is preferable over manually creating control meshes. On the other hand, when colouring vector graphics, diffusion curves are limited compared to gradient meshes. First-order diffusion curves are only concerned with colours at the curves and are therefore restricted in terms of colourings inside a domain. Second-order diffusion curves improve on this limitation with the support for zero-derivative curve and point constraints. However, their colour constraints can saturate the colour function, as bi-Laplacian diffusion does not possess the maximum property, and can therefore make adjustments propagate unnaturally. By contrast, gradient meshes provide more local control and support a more direct specification of the colour gradient compared to the zero-derivative curves of diffusion curves.

Throughout this dissertation, I have argued that a unification of these two methods can provide several advantages. As input, one would like freeform curves. Such curves can be specified directly (like Illustrator's pen tool) or be extracted from brush strokes (like Illustrator's brush tool). While colours can be associated with this type of input, it should not be a requirement. The colouring aspects of vector graphics should be separated from the input curves. One can imagine point constraints for simpler types of colourings and a mesh-based interface for more complex colourings. A mesh-based interface can provide advantages since derivative constraints can more easily be specified (for example, one constraint along each edge of a control point). Additionally, the spatial relationship between colour points is more defined and manageable, via the topological construction of the mesh, compared to a set of disconnected

points. While the current technology, including my contributions, is restricted in terms of this goal, I believe my contributions narrows the gap between diffusion curves and gradient meshes. In the following, I outline these contributions.

9.1 Practical contributions

The shading curve primitive (Chapter 5) extends the basic diffusion-curve representation (colours associated with curves) with a shading profile. This shading profile represents how the colour at the curve is propagated in the perpendicular direction to the curve. The basic first-order diffusion curve primitive only supports a colour attribute to be associated with each side of the curve. To alter the gradient, the image produced by the diffusion process can be blurred along the curve[73]. Alternatively, one can define rational harmonic functions, meaning that a weight can be associated with the curve[4]. This weight alters the relative influence between neighbouring curves. To force the first derivative of the colour function to zero in a given direction, the derivative curves of second-order diffusion curves can be used[31]. In contrast to these previous approaches, colour propagation is manipulated via curves. My contribution therefore represents an additional degree of freedom added to the diffusion curve primitive, where users are able to more explicitly define colour propagation out from curves, using curves rather than scalar weights or pre-defined constraints.

It is challenging to render the proposed shading profile with (bi-)Laplacian diffusion. Bi-harmonic functions, produced by bi-Laplacian diffusion, are influenced globally by their boundary conditions. This means that it is challenging to apply a shading profile locally in the image, without altering other image regions. Additionally, bi-harmonic functions do not possess the maximum property, meaning that derivative constraints cannot be freely altered and at the same time guaranteeing that the colour function does not over-saturate (Section 6.2.2). To render the shading profile correctly, a fundamentally different approach had to be invented. With Catmull-Clark surfaces, which achieve local control, I demonstrated in Chapter 5 that the shading profile can be smoothly applied to vector images. While I demonstrated a wide range of results, I argue that the shading curve is particularly suited for shading, where I suggest following the workflow of the traditional chiaroscuro technique for drawing shade and light.

The gradient meshes I describe in Chapter 6 can be defined with arbitrary manifold topology, along with directional colour weights to emulate derivative constraints of regular gradient meshes. Again, this contribution represents an additional degree of freedom, where the technology now is unrestricted to the mesh topology rather than being restricted to rectangular meshes. The gradient mesh has been previously utilised by researchers (e.g. refs. 95;53) to solve challenging problems related to vector graphics. However, research in recent years has, owing to this restriction, diverted from gradient meshes to diffusion curves[72] and alternative types of solutions, like Loop subdivision[58]. With the rectangular-grid restriction now lifted, I would like to see researchers and developers using my solutions to invent new and interesting tools for computer graphics.

A common view of vector-based image processing is that the image must first be converted to a complete vector representation before the advantages of vector-based manipulation can be utilised[27;58;73;95]. In Chapters 7 and 8, I provide an alternative view: given a set of curves ex-

tracted from the image, an intermediate vector representation of the effect can be created. Then, the manipulated image is defined by supplementing the original image with this intermediate vector image. My approach therefore avoids the challenging problem of creating a complete vector representation of the image.

My approach is particularly useful when an effect defined by a low-frequency profile is added to the image. This is demonstrated in Chapter 7, where I use Cornsweet surfaces to alter the perceived contrast between objects in photographs. The advantage of this approach, compared to standard contrast enhancement, is that the perceived contrast is potentially enhanced, without altering the original look of the image. By comparison, standard contrast enhancement often increases details as well as contrast. This challenging problem, which is previously studied[52;87;101], is inspired by the Cornsweet illusion and related findings in perceptual psychology[50]. From these findings, it is well-known that the Cornsweet profile provides the most effective perceived contrast enhancement between two image regions. While filter-based methods, using the unsharp mask, correctly apply the Cornsweet profile in many settings, they quickly over- and under-saturate the colour or luminance image if image regions are too close to each other. Additionally, the parameter to adjust the enhancement magnitude may provide different magnitudes of enhancement depending on image content. Using a fundamentally different vector-centric approach, I have demonstrated that these issues can be dealt with more robustly by local, explicit, control of the enhancement profile using freeform surfaces.

The framework presented in Chapter 7 is fully automatic and is tuned towards Cornsweet-style contrast enhancement. The framework, however, is highly adaptable, as demonstrated by the similar shading curve framework. To this end, I demonstrate several applications within artistic image manipulation in Chapter 8. Using strokes as input, users can select image edges, adjust local mesh extents, and manipulate the magnitude of the effect with strokes. Additionally, the shape of the effect can be adjusted with the NURBS weight. Of course, the shading curve can also be utilised in this setting. I demonstrate enhancement in alternative colour channels, authoring of shade and light with intrinsic images, and artistic enhancement with extreme magnitudes. Such manipulations are challenging with filter-based methods because the effects are applied locally in the image in a non-linear manner. The types of effects (modelling adjustment profiles of low frequencies) are also not typically associated with vector-based effects, which are mostly concerned with colourisation, shape deformation, and other operations related to geometric properties[58]. Thus, there are no obvious advantages of converting the image into a vector representation to achieve the effects I have demonstrated.

9.2 Technical contributions

In this dissertation, I have presented solutions to two technical problems: creating control meshes from curves and interpolating colours across control meshes of arbitrary manifold topology.

Creating control meshes from curves

My approach to mesh creation, as presented in Chapter 4, is novel: control meshes have previously not been constructed from a set of curves with attributes to explicitly control the shape of the resulting surfaces.

The principal challenge when defining such control meshes is to define the 'outermost' points related to a curve point. These are defined according to the extent attribute associated with that curve point. While several solutions to this sub-problem have been proposed[38;1;31], they are relatively naïve according to my requirements.

I argue that to achieve a robust solution, the projected meshes to 2D should not overlap each other and not fold. I have proposed two solutions to this problem (Section 4.2), both of which satisfy the given conditions as long as the input curves do not intersect. I have demonstrated (Section 4.1.4) that previous solutions do not achieve the same level of robustness; that is, they are not robust to the topology of the input curves.

This problem is ill-posed and is therefore challenging. It is ill-posed because there are two sensible solutions at certain regions. I classify these two types of solutions as a solution at a narrow region and a solution at a connected narrow region (also known as corners). To deal with this issue, I make various assumptions in my solutions. In Solution 1 (Section 4.2.3), I assume corners to be tagged as input and in Solution 2 (Section 4.2.4), I assume that the extent attribute is set at the preferred offset. To achieve robust solutions, I employ mathematical tools related with the distance transform, the medial axis, and Voronoi partitioning.

Gradient meshes of arbitrary manifold topology

The problem of defining surfaces related to control meshes of arbitrary manifold topology is not new and well-established solutions exist. There are two sensible types of approaches: subdivision with 'special' rules at irregular mesh elements (like Catmull-Clark or Doo-Sabin subdivision) or Hermite interpolation, ensuring that the gradient defined by the derivative constraints at each vertex is continuous.

The principal problem I am concerned with in Chapter 6 is to define a surface that emulates Ferguson patches in the regular setting and to extend this behaviour in the irregular setting. A relatively naïve solution is to convert ill-defined derivative constraints at irregular vertices, for example, by fitting a gradient to those constraints, and then perform Hermite interpolation. Such an approach has been demonstrated in the setting of gradient mesh simplification by projecting the gradient of the original vertices to the remaining vertices[57]. I argue that such a solution would not be sufficient in a general setting of gradient meshes: the behaviour of the derivative constraints will not be extended if they are simply averaged together. Moreover, to satisfy such constraints at vertices of arbitrary valency is a significant challenge because an arbitrary set of gradients cannot be achieved at a single point.

For these reasons, I employ Catmull-Clark subdivision surfaces. The standard Catmull-Clark scheme is an approximating scheme, meaning that the original data points will not typically be interpolated, which is required to emulate the behaviour of Ferguson patches. To this end, I have extended the Catmull-Clark scheme so that the resulting surface satisfies the required

conditions. This was achieved by separating the treatment of geometric and colour coordinates and by forcing zero curvature at the original control points in colour space. A special set of initial subdivision rules have been invented for the geometric coordinates to emulate the derivative constraints of Ferguson patches.

9.3 Future work

I would like to see future work that further closes the gap between diffusion curves and gradient meshes. Eventually, I envision a unified framework that uses freeform curves as input, with support for intricate colourings via a mesh-based representation. Throughout this dissertation, I have discussed the potential of such a framework and I have presented several ideas for future improvement. In the following, I summarise these thoughts.

The next barrier that should be dealt with, I argue, is to create control meshes, covering the entire given domain, from freeform curves. This problem is discussed in detail in Section 4.3. The following two aspects should be adressed:

- The type of mesh(es): a tricky aspect that probably involves subjectivity. A sensible argument is that pure quad meshes should be created. Advantages of such meshes are given in Section 4.3. However, I have demonstrated, in Chapter 6, that other types of meshes can produce high-quality colourings. For example, the pepper example (Figure 6.18) is a quad-dominant mesh and the girl example (Figure 6.19) is a quad-dominant mesh with high-valency faces (up to 8-valence).
- Density of meshes. Many would argue that we would like meshes as sparse as possible. For complex objects and colourings, however, a certain density would be needed. Should we allow the user to specify the level of density to begin with or should we create the sparsest mesh possible (which can later be colour-edited via multi-resolution edits)?

Colours should not be the only source of colour editing. To be able to control the propagation of colours is equally important. A question that should be addressed is what type of colour-gradient constraints provides the best type of control? Currently, we have a wide range of options: point constraints, zero-derivative curves, shading profiles, linear gradients, and derivative constraints in polygonal domains and meshes. All of these types of control mechanisms are currently incompatible. In a unified framework, which of these should be included and which of these should remain in a separate tool?

Finally, diffusion curves provide a way to diffuse colours to the entire domain, similar to heat diffusion. By contrast, constructions based on B-splines, or similar types of splines, are constructed from the minimal support property, meaning that surface alterations should only influence the local neighbourhood of the surface (for example, if you alter the front bumper of a car, this should not change the roof). A unified framework should support both of these two types of behaviours. That is, if the user wishes a diffusion-like behaviour, the framework should provide this, and vice versa. An interesting aspect is whether such a framework can support a transition between these two, fundamentally different, types of modes or whether it must provide the user with a binary choice.

Bibliography

[1] P. J. Asente. Folding avoidance in skeletal strokes. In *Proc. SBIM*, volume 7, pages 33–40, Annecy, France, June 2010. Eurographics Association.

[2] S. Bae, S. Paris, and F. Durand. Two-scale tone management for photographic look. *ACM Trans. Graph.*, 25(3):637–645, July 2006.

[3] H. G. Barrow and J. M. Tenenbaum. Recovering intrinsic scene characteristics from images. *Computer Vision Systems*, 3:3–26, 1978.

[4] H. Bezerra, E. Eisemann, D. DeCarlo, and J. Thollot. Diffusion constraints for vector graphics. In *Proc. NPAR*, volume 10, pages 35–42, Annecy, France, June 2010. ACM.

[5] F. Billmeyer and M. Saltzman. *Principles of color technology*. Wiley, New York City, USA, 3rd edition, 2000.

[6] H. Blum. A transformation for extracting new descriptors of shape. *Models for the Perception of Speech and Visual Form*, 1:362–380, 1967.

[7] D. Bommes, B. Lévy, N. Pietroni, E. Puppo, C. S. a, M. Tarini, and D. Zorin. State of the art in quad meshing. In *Eurographics STARS*, Cagliari, Italy, May 2012.

[8] D. Bommes, H. Zimmer, and L. Kobbelt. Mixed-integer quadrangulation. *ACM Trans. Graph.*, 28(3):77:1–10, July 2009.

[9] A. Bousseau, S. Paris, and F. Durand. User-assisted intrinsic images. *ACM Trans. Graph.*, 28(5):130:1–10, Dec. 2009.

[10] S. Boyé, P. Barla, and G. Guennebaud. A vectorial solver for free-form vector gradients. *ACM Trans. Graph.*, 31(6):173:1–9, Nov. 2012.

[11] P. Burt and E. Adelson. The Laplacian pyramid as a compact image code. *IEEE Trans. Communications*, 31(4):532–540, Apr. 1983.

[12] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10(6):350–355, Nov. 1978.

[13] C. D. Cennini. *The Craftsman's Handbook: "Il Libro dell' Arte"*. (D. V. Thompson, Trans.). Dover Publications, Mineola, USA, 1954 (Original work published early 15th century).

[14] G. Chen, V. Kwatra, L.-Y. Wei, C. D. Hansen, and E. Zhang. Design of 2D time-varying vector fields. *IEEE Trans. Vis. Graph.*, 18(10):1717–1730, Oct. 2012.

[15] M. E. Chevreul. *De la loi du contraste simultané des couleurs et de ĺassortiment des objets colorés*. Pitois-Levrault et ce., 1839.

[16] Y.-Y. Chuang, B. Curless, D. H. Salesin, and R. Szeliski. A bayesian approach to digital matting. In *Proc. CVPR*, volume 2, pages 264–271. IEEE, Dec. 2001.

[17] G. Civardi. *Drawing Light & Shade: Understanding Chiaroscuro*. (L. Black, Trans.). Search Press Limited, Tunbridge Wells, UK, 2006 (Original work published 2005).

[18] R. L. Cook and K. E. Torrance. A reflectance model for computer graphics. *ACM Trans. Graph.*, 1:7–24, Jan. 1982.

[19] S. A. Coons. Surfaces for computer aided design. Technical report, Massachusetts Institute of Technology, USA, 1964.

[20] T. N. Cornsweet. *Visual Perception*. Harcourt, San Diego, USA, 1970.

[21] K. J. W. Craik. *The Nature of Psychology*. Cambridge University Press, Cambridge, UK, 1966.

[22] C. de Boor. *A Practical Guide to Splines*. Springer, New York City, USA, 1978.

[23] T. DeRose, M. Kass, and T. Truong. Subdivision surfaces in character animation. In *Proc. SIGGRAPH*, volume 25, pages 85–94, Orlando, USA, July 1998. ACM.

[24] D. Doo and M. Sabin. Behavior of recursive division surfaces near extraordinary points. *Computer-Aided Design*, 10(6):356–360, Nov. 1978.

[25] R. P. Dooley and M. I. Greenfield. Measurement of edge-induced visual contrast and a spatial-frequency interaction of the Cornsweet illusion. *J. Opt. Soc. America*, 67(6):761–765, 1977.

[26] E. Eisemann and F. Durand. Flash photography enhancement via intrinsic relighting. *ACM Trans. Graph.*, 23(3):673–678, Aug. 2004.

[27] J. H. Elder and R. M. Goldberg. Image editing in the contour domain. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(3):291–296, Mar. 2001.

[28] Z. Farbman, R. Fattal, D. Lischinski, and R. Szeliski. Edge-preserving decompositions for multi-scale tone and detail manipulation. *ACM Trans. Graph.*, 27(3):67:1–10, Aug. 2008.

[29] J. Ferguson. Multivariable curve interpolation. *J. ACM*, 11(2):221–228, Apr. 1964.

[30] J. A. Ferwerda, S. N. Pattanaik, P. Shirley, and D. P. Greenberg. A model of visual masking for computer graphics. In *Proc. SIGGRAPH*, volume 24, pages 143–152, Los Angeles, USA, 1997. ACM.

[31] M. Finch, J. Snyder, and H. Hoppe. Freeform vector graphics with controlled thin-plate splines. *ACM Trans. Graph.*, 30(6):166:1–10, Dec. 2011.

[32] M. S. Floater. Mean value coordinates. *Computer Aided Geometric Design*, 20(1):19–27, Mar 2003.

[33] A. Gooch, B. Gooch, P. Shirley, and E. Cohen. A non-photorealistic lighting model for automatic technical illustration. In *Proc. SIGGRAPH*, volume 25, pages 447–452, Orlando, USA, July 1998. ACM.

[34] C. Grant. David Hockney's instant iPad art. BBC News, Technology, Nov 2010.

[35] R. Grosse, M. K. Johnson, E. H. Adelson, and W. T. Freeman. Ground-truth dataset and baseline evaluations for intrinsic image algorithms. In *Proc. ICCV*, volume 12, pages 2335–2342. IEEE, Sept. 2009.

[36] H. Hnaidi, E. Guérin, S. Akkouche, A. Peytavie, and E. Galin. Feature based terrain generation using diffusion equation. *Computer Graphics Forum*, 29(7):2179–2186, Oct. 2010.

[37] D. Hoiem, A. A. Efros, and M. Hebert. Recovering occlusion boundaries from an image. *Int. J. Comput. Vision*, 91(3):328–346, Feb. 2011.

[38] S. C. Hsu, I. H. H. Lee, and N. E. Wiseman. Skeletal strokes. In *Proc. UIST*, volume 6, pages 197–206, San Jose, USA, 1993. ACM.

[39] M. Ihrke, T. Ritschel, K. Smith, T. Grosch, K. Myszkowski, and H.-P. Seidel. A perceptual evaluation of 3D unsharp masking. In *Human Vision and Electronic Imaging XIV*, Proc. SPIE 7240, page 72400R, Jan. 2009.

[40] P. Ilbery, L. Kendall, C. Concolato, and M. McCosker. Biharmonic diffusion curve images from boundary elements. *ACM Trans. Graph.*, 32(6):219:1–12, Nov. 2013.

[41] S. Jeschke, D. Cline, and P. Wonka. Rendering surface details with diffusion curves. *ACM Trans. Graph.*, 28(5):117:1–8, Dec. 2009.

[42] S. Jeschke, D. Cline, and P. Wonka. Estimating color and texture parameters for vector graphics. *Computer Graphics Forum*, 30(2):523–532, Apr. 2011.

[43] P. Joshi and N. A. Carr. Repoussé;: Automatic inflation of 2d artwork. In *Proc. SBIM*, volume 5, pages 49–55, Annecy, France, June 2008. Eurographics Association.

[44] P. Joshi, M. Meyer, T. DeRose, B. Green, and T. Sanocki. Harmonic coordinates for character articulation. *ACM Trans. Graph.*, 26(3):71:1–10, July 2007.

[45] J. T. Kajiya. The rendering equation. In *Proc. SIGGRAPH*, volume 13, pages 143–150, Dallas, USA, Aug. 1986. ACM.

[46] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *Int. J. of Computer Vision*, 1(4):321–331, Jan. 1988.

[47] Keydata. Computer display review. Technical report, Keydata Corp, 1970.

[48] H.-C. Kim. Tool path generation for contour parallel milling with incomplete mesh model. *The Int. J. of Advanced Manufacturing Technology*, 48(5-8):443–454, May 2010.

[49] R. Kimmel, N. Kiryati, and A. M. Bruckstein. Sub-pixel distance maps and weighted distance transforms. *Journal of Mathematical Imaging and Vision*, 6(2-3):223–233, June 1996.

[50] F. Kingdom and B. Moulden. Border effects on brightness: A review. *Spatial Vision*, 3(4):225–262, 1988.

[51] J. Kopf and D. Lischinski. Depixelizing pixel art. *ACM Trans. Graph.*, 30(4):99:1–8, Aug. 2011.

[52] G. Krawczyk, K. Myszkowski, and H.-P. Seidel. Contrast restoration by adaptive countershading. *Computer Graphics Forum*, 26(3):581–590, Sept. 2007.

[53] Y.-K. Lai, S.-M. Hu, and R. R. Martin. Automatic and topology-preserving gradient mesh generation for image vectorization. *ACM Trans. Graph.*, 28(3):85:1–8, July 2009.

[54] I. Leichter and M. Lindenbaum. Boundary ownership by lifting to 2.1D. In *Proc. ICCV*, volume 11, pages 9–16. IEEE, Sept. 2009.

[55] A. Levin, D. Lischinski, and Y. Weiss. A closed-form solution to natural image matting. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(2):228–242, Feb. 2008.

[56] X. Li and J. Zheng. An alternative method for constructing interpolatory subdivision from approximating subdivision. *Computer Aided Geometric Design*, 29(7):474–484, Oct. 2012.

[57] X.-Y. Li, T. Ju, and S.-M. Hu. Cubic mean value coordinates. *ACM Trans. Graph.*, 32(4):126:1–10, July 2013.

[58] Z. Liao, H. Hoppe, D. Forsyth, and Z. Yu. A subdivision-based representation for vector image editing. *IEEE Trans. Vis. Graph.*, 18(11):1858 – 1867, Nov. 2012.

[59] C. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah, USA, 1987.

[60] C. Loop, S. Schaefer, T. Ni, and I. Castaño. Approximating subdivision surfaces with Gregory patches for hardware tessellation. *ACM Trans. Graph.*, 28(5):151:1–9, Dec. 2009.

[61] J. Lopez-Moreno, S. Popov, A. Bousseau, M. Agrawala, and G. Drettakis. Depicting stylized materials with vector shade trees. *ACM Trans. Graph.*, 32(4):118:1–10, July 2013.

[62] T. Luft, C. Colditz, and O. Deussen. Image enhancement by unsharp masking the depth buffer. *ACM Trans. Graph.*, 25(3):1206–1213, 2006.

[63] R. Maharik, M. Bessmeltsev, A. Sheffer, A. Shamir, and N. Carr. Digital micrography. *ACM Trans. Graph.*, 30(4):100:1–12, July 2011.

[64] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. ICCV*, volume 2, pages 416–423. IEEE, July 2001.

[65] J. McCann and N. S. Pollard. Real-time gradient-domain painting. *ACM Trans. Graph.*, 27(3):93:1–7, Aug. 2008.

[66] G. Medioni and Y. Yasumoto. Corner detection and curve representation using cubic B-splines. *Comp. Vis., Graphics and Image Proc.*, 39(3):267 – 278, Apr. 1987.

[67] H. Nebi Gürsoy and N. Patrikalakis. An automatic coarse and fine surface mesh generation scheme based on medial axis transform: Part I algorithms. *Engineering with Computers*, 8(3):121–137, 1992.

[68] F. Neycenssac. Contrast enhancement using the Laplacian-of-a-Gaussian filter. *CVGIP: Graph. Models Image Process.*, 55(6):447–463, Nov. 1993.

[69] M. Nießner, C. Loop, M. Meyer, and T. Derose. Feature-adaptive GPU rendering of Catmull-Clark subdivision surfaces. *ACM Trans. Graph.*, 31(1):6:1–11, Feb. 2012.

[70] V. O'Brien. Contour perception, illusion and reality. *J. Opt. Soc. Am.*, 48(2):112–119, Feb 1958.

[71] M. Okabe, G. Zeng, Y. Matsushita, T. Igarashi, L. Quan, and H. yeung Shum. Single-view relighting with normal map painting. In *Proc. Pacific Graphics*, volume 14, pages 27–34, Taipei, Taiwan (R. O. C.), Oct. 2006.

[72] A. Orzan, A. Bousseau, P. Barla, H. Winnemöller, J. Thollot, and D. Salesin. Diffusion curves: A vector representation for smooth-shaded images. *Commun. ACM*, 56(7):101–108, July 2013.

[73] A. Orzan, A. Bousseau, H. Winnemöller, P. Barla, J. Thollot, and D. Salesin. Diffusion curves: A vector representation for smooth-shaded images. *ACM Trans. Graph.*, 27(3):92:1–8, Aug. 2008.

[74] J. Palacios and E. Zhang. Rotational symmetry field design on surfaces. *ACM Trans. Graph.*, 26(3):55:1–10, July 2007.

[75] S. Paris, S. W. Hasinoff, and J. Kautz. Local Laplacian filters: edge-aware image processing with a Laplacian pyramid. *ACM Trans. Graph.*, 30(4):68:1–12, July 2011.

[76] P. Pérez, M. Gangnet, and A. Blake. Poisson image editing. *ACM Trans. Graph.*, 22(3):313–318, July 2003.

[77] G. Petschnigg, R. Szeliski, M. Agrawala, M. Cohen, H. Hoppe, and K. Toyama. Digital photography with flash and no-flash image pairs. *ACM Trans. Graph.*, 23(3):664–672, Aug. 2004.

[78] B. T. Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6):311–317, June 1975.

[79] J. M. S. Prewitt. Object enhancement and extraction. *Picture Processing and Psychopictorics*, 1:75–149, 1970.

[80] S. J. D. Prince. *Computer Vision: Models, Learning, and Inference*. Cambridge University Press, Cambridge, UK, 2012.

[81] D. Purves, A. Shimpi, and R. Lotto. An empirical explanation of the Cornsweet effect. *J. Neurosci.*, 19(19):8542–8551, Oct. 1999.

[82] W. R. Quadros, K. Ramaswami, F. B. Prinz, and B. Gurumoorthy. Laytracks: a new approach to automated geometry adaptive quadrilateral mesh generation using medial axis transform. *Int. J. for Numerical Methods in Engineering*, 61(2):209–237, Sept. 2004.

[83] F. Ratliff. Contour and contrast. *Proceedings of the American Philosophical Society*, 115(2):150–163, Apr. 1971.

[84] K. C. Redmond and T. M. Smith. *From Whirlwind to MITRE: The R&D story of the SAGE Air Defense Computer*. MIT Press, Cambridge, USA, 2000.

[85] J.-F. Remacle, J. Lambrechts, B. Seny, E. Marchandise, A. Johnen, and C. Geuzainet. Blossom-quad: A non-uniform quadrilateral mesh generator using a minimum-cost perfect-matching algorithm. *Int. J. for Numerical Methods in Engineering*, 89(9):1102–1119, Mar. 2012.

[86] X. Ren, C. C. Fowlkes, and J. Malik. Figure/Ground assignment in natural images. In *Proc. ECCV*, volume 9, pages 614–627, Graz, Austria, May 2006.

[87] T. Ritschel, K. Smith, M. Ihrke, T. Grosch, K. Myszkowski, and H.-P. Seidel. 3D unsharp masking for scene coherent enhancement. *ACM Trans. Graph.*, 27(3):90:1–8, Aug. 2008.

[88] A. Rosenfeld and J. L. Pfaltz. Sequential operations in digital picture processing. *J. ACM*, 13(4):471–494, Oct. 1966.

[89] P. Sampl. Semi-structured mesh generation based on medial axis. In *Proc. Int. Meshing Roundtable*, volume 9, pages 21–32, New Orleans, USA, Oct. 2000.

[90] T. W. Sederberg, J. Zheng, D. Sewell, and M. Sabin. Non-uniform recursive subdivision surfaces. In *Proc. SIGGRAPH*, volume 25, pages 387–394, Orlando, USA, July 1998. ACM.

[91] P. B. Seel. *Digital Universe: The Global Telecommunication Revolution*. Wiley-Blackwell, Hoboken, USA, 2012.

[92] C. Shao, A. Bousseau, A. Sheffer, and K. Singh. Crossshade: shading concept sketches using cross-section curves. *ACM Trans. Graph.*, 31(4):45:1–11, July 2012.

[93] R. M. Shapley and D. J. Tolhurst. Edge detectors in human vision. *The Journal of physiology*, 229(1):165–183, Feb. 1973.

[94] V. Srinivasan, L. R. Nackman, J.-M. Tang, and S. N. Meshkat. Automatic mesh generation using the symmetric axis transformation of polygonal domains. *Proceedings of the IEEE*, 80(9):1485–1501, Sept. 1992.

[95] J. Sun, L. Liang, F. Wen, and H.-Y. Shum. Image vectorization using optimized gradient meshes. *ACM Trans. Graph.*, 26(3):11:1–8, July 2007.

[96] I. Sutherland. *Sketchpad, A Man-Machine Graphical Communication System*. PhD thesis, Massachusetts Institute of Technology, USA, 1963.

[97] D. Sýkora, L. Kavan, M. Čadík, O. Jamriška, A. Jacobson, B. Whited, M. Simmons, and O. Sorkine-Hornung. Ink-and-ray: Bas-relief meshes for adding global illumination effects to hand-drawn characters. *ACM Trans. Graph.*, 33(2):16:1–15, Apr. 2014.

[98] K. Takayama, O. Sorkine, A. Nealen, and T. Igarashi. Volumetric modeling with diffusion surfaces. *ACM Trans. Graph.*, 29(6):180:1–8, Dec. 2010.

[99] T. Tam and C. Armstrong. 2D finite element mesh generation by medial axis subdivision. *Advances in Engineering Software and Workstations*, 13(5–6):313 – 324, Sept. 1991.

[100] W. Tiller. Rational B-splines for curve and surface representation. *IEEE Comput. Graph. Appl.*, 3(6):61–69, June 1983.

[101] M. Trentacoste, R. Mantiuk, W. Heidrich, and F. Dufrot. Unsharp masking, countershading and halos: Enhancements or artifacts? *Computer Graphics Forum*, 31(2):555–564, 2012.

[102] R. Vergne, P. Barla, R. W. Fleming, and X. Granier. Surface flows for image-based shading design. *ACM Trans. Graph.*, 31(4):94:1–9, July 2012.

[103] K. J. Versprille. *Computer-aided design applications of the rational B-spline approximation form.* PhD thesis, Syracuse University, USA, 1975.

[104] E. L. Wachspress. *A Rational Finite Element Basis*, volume 114 of *Mathematics in Science and Engineering*. Elsevier, Amsterdam, Netherlands, 1975.

[105] T. Wachtler and C. Wehrhahn. The Craik-O'Brien-Cornsweet illusion in colour: Quantitative characterisation and comparison with luminance. *Perception*, 26:1423–1430, 1997.

[106] O. Weber, R. Poranne, and C. Gotsman. Biharmonic coordinates. *Computer Graphics Forum*, 31(8):2409–2422, Dec. 2012.

[107] H. Winnemöller, A. Orzan, L. Boissieux, and J. Thollot. Texture design and draping in 2D images. *Computer Graphics Forum*, 28(4):1091–1099, June 2009.

[108] H. Wolfflin. *Principles of Art History: The Problem of the Development of Style in Later Art*. Dover Publications, Mineola, USA, 1986.

[109] T.-P. Wu, C.-K. Tang, M. S. Brown, and H.-Y. Shum. Shapepalettes: Interactive normal transfer via sketching. *ACM Trans. Graph.*, 26(3):44:1–6, July 2007.

[110] T. Xia, B. Liao, and Y. Yu. Patch-based image vectorization with automatic curvilinear feature alignment. *ACM Trans. Graph.*, 28(5):115:1–10, Dec. 2009.

[111] Y.-L. Yang, J. Wang, E. Vouga, and P. Wonka. Urban pattern: Layout design by hierarchical domain splitting. *ACM Trans. Graph.*, 32(6):181:1–12, Nov. 2013.

[112] L. Zhang, G. Dugas-Phocion, J.-S. Samson, and S. Seitz. Single view modeling of free-form scenes. In *Proc. CVPR*, volume 1, pages 990–997, Kauai, USA, Dec. 2001. IEEE.

[113] D. Zorin. Modeling with multiresolution subdivision surfaces. In *SIGGRAPH 2006 Courses*, pages 30–50, Boston, USA, July 2006. ACM.

Appendices

Appendix A

Implementation details

In this appendix, I describe noteworthy aspects of my implementations of the solutions presented in Chapter 4. Recall that I presented two solutions to the given meshing problem (Solutions 1 and 2). In Section A.1, I describe a MATLAB implementation of Solution 1 and in Section A.2, I describe two C++ implementations of Solution 2.

A.1 Solution 1

Noteworthy aspects of my MATLAB implementation of Solution 1 are now described. This implementation was written for the application presented in Chapter 7 (Cornsweet-style contrast enhancement).

The Voronoi partition is defined by a set of branches: $A = \{b_k\}$. In my implementation, this partition is represented as a discrete image defined via MATLAB's **bwdist** function (the discrete distance transform) on the discretised B-spline curves. The partitioning is performed using the indexing, provided by the bwdist function, to the nearest curve segments to define Voronoi cells as image segments. Branches can be trivially deleted from the set by merging their related Voronoi cells. That is, the branches are only implicitly defined via the image segments.

Step 1 traces rays from Q_i in the direction of N_i. This tracing is performed in pixel space in the discrete Voronoi image. The starting point, or pixel, for this tracing operation must be inside the correct cell. This starting pixel is defined to be the pixel next to the closest discrete curve pixel to Q_i (which has already been found by the distance transform).

There are five termination cases. Case (ii) terminates r_i when it hits an edge of the current Voronoi partition. This happens when the tracing enters a Voronoi image segment not associated with the current curve segment. In cases (iii) and (v), the tracing terminates when it hits another discrete edge pixel (must be of the same curve segment). Here, all discrete edges are dilated using the 2×2 structuring element $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$. This makes the edges waterproof, provided that the input edges are connected.

A.2 Solution 2

Noteworthy aspects of my C++ implementations of Solution 2 are now described. I have implemented two implementations of this solution. The first implementation, presented in Section A.2.1, was written for the application presented in Chapter 5 (the shading curve). This implementation traces paths in the discrete distance transform. The second implementation, presented in Section A.2.1, was written for various tests related to the problem presented in Chapter 6 (gradient meshes of abritrary topology). This implementation traces paths in the Voronoi partition of the curve control points.

A.2.1 Implementation 1

Solution 1 can be naïvely implemented by tracing the discrete distance transform (DDT). Note that the particles described in Section 4.2.4 do not need to be explicitly modelled; a single (**while**) loop is sufficient to 'travel' between pixels until a termination criterion is reached.

For a given Q_i and direction N_i, the DDT tracing is initiated at a pixel on the 'correct' side of the curve (that is, the pixel next to the closest discrete curve pixel to Q_i towards N_i, which is given by the DDT function as mentioned in Section A.1). The tracing, along the path of the abstract particle, is performed by iteratively moving the current pixel (or particle) to the pixel with the maximum DDT value in the 1-ring pixel neighbourhood of the current pixel. The tracing stops when the DDT value is above e_i (the given extent has been reached) or the pixel is a local extremum (a stationary point).

Equal values in the 1-ring neighbourhood are handled by reference to the ray r_i (defined in Solution 1). For example, if the curve is a horizontal line, there will be three pixels of the same value in the neighbourhood. This problem is solved by computing the distances from the pixels (all with the maximum DDT value) to r_i and then moving to the pixel with the lowest distance. Such tracing is therefore performed approximately compared to the definition of the gradient of the DT surface, where the tracing is not guaranteed to follow the normal direction exactly to the MA.

This implementation was used to create the images in Chapter 5. While it can be noted that the quantisation error produced by this discrete approach can give rise to uneven spacing of $P_{i,m}$, the cubic B-spline curve defined by $P_{i,m}$ control points is a sufficiently good offset approximation, at least to demonstrate the application described in Chapter 5. An example of such uneven spacing can be seen in Figure A.1(left), where the $P_{i,1} - P_{i,m}$ lines are of different direction to the curve's normal direction.

The quantisation error is obviously due to the fact that the solution is performed in pixel space. As an alternative, I also implemented an implementation that operates on a Voronoi partition, thus avoiding quantisation. This implementation is described next. Figure A.1 illustrates the difference between these two implementations.

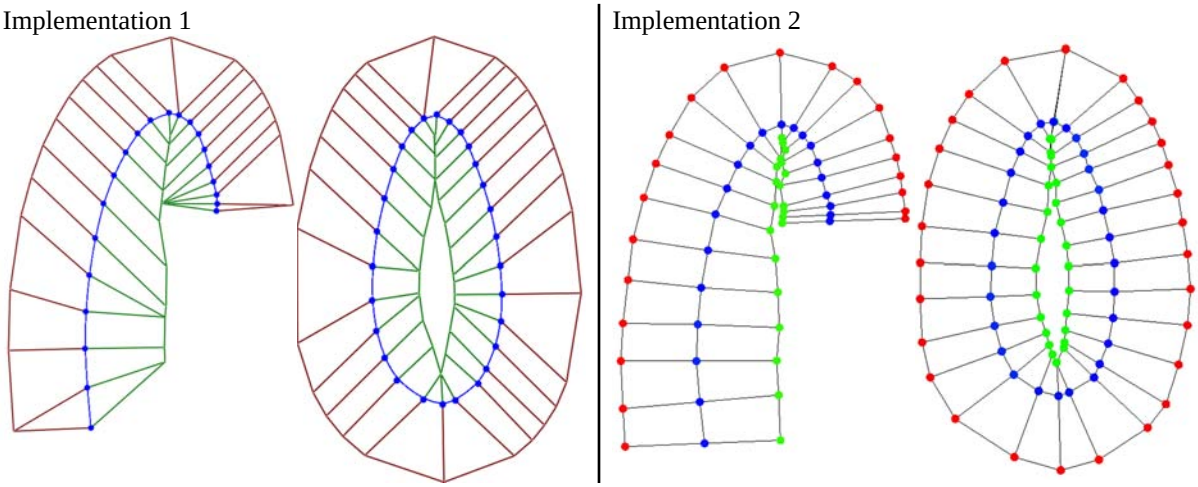


Figure A.1: Comparison between the two implementations of Solution 2.

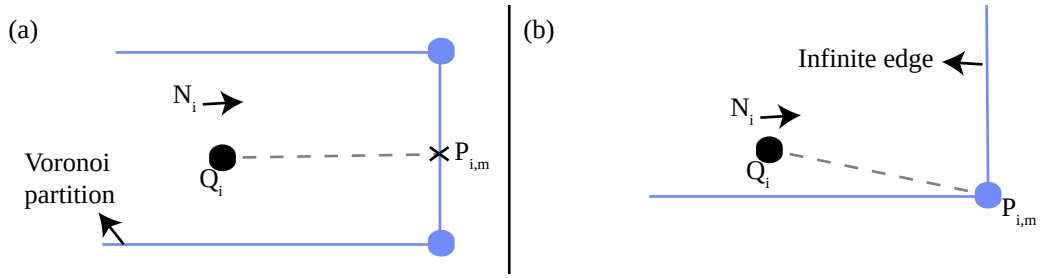


Figure A.2: Two possible cases for Implementation 2: the related Voronoi edge to a curve control point Q_i in the direction N_i is either a finite edge (a) or an infinite edge (b).

A.2.2 Implementation 2

Evaluating Solution 2 in a Voronoi partition, and not directly in the DDT image, introduces additional algorithmic challenges. These challenges depend on the data structure used to represent the Voronoi partition. To this end, I do not present the fine details of my code, but rather provide some suggestions to how to deal with the general problems faced with such an implementation.

There are many available implementations of Voronoi partirtioning of a set of points. I employed the C++ Boost Voronoi library[i]. The Voronoi partition is represented as a half-edge data structure (a doubly-connected edge list). A framework using such a data structure can provide efficient traversal and efficient manipulation of faces, edges, and vertices of the polygonal mesh (in this case a Voronoi partition).

The vertices of the Voronoi partition can provide a good approximation to the MA. From my experience, it can be wise to sample the input curve so that the density of points is sufficiently high for an accurate approximation. This, of course, depends on the density of the input curve control points.

The main challenge of this implementation, compared to tracing the DDT, is to follow the path of the particle by traversing the half-edge data structure. In this description, I point out the

[i] boost.org/doc/libs/1_53_0_beta1/libs/polygon/doc/voronoi_main.htm

following considerations and guidelines:

- At any given point in the domain, it can be instructive to define a procedure that estimates the distance to its closest curve point (that is, the DT value of the point). In Boost, any cell is associated with an input point. Thus, the DT value can be approximated by computing the distance from the given point to the curve control point Q_i of its Voronoi cell.
- The first phase of the movement of the particle is concerned with the line between Q_i and a corresponding point on the MA. It is challenging to exactly locate this MA point with the given data structure. However, the following approximation has shown sufficient when the MA approximation is accurate: any given point Q_i is associated with one or two Voronoi vertices in the direction N_i (Figure A.2). If Q_i is related to a single Voronoi vertex; that is, the related edge extends to infinity, the MA point is defined as this vertex (Figure A.2(b)). If Q_i is related with two Voronoi vertices; that is, they define a finite edge, the MA point is defined as the midpoint of this edge (Figure A.2(a)). Note that more accurate definitions of the MA point can be made, for example by intersecting r_i with the related edge. However, I found the above approach sufficient for my experiments.
- The second phase of the movement of the particle is concerned with the path along the MA. To traverse along the MA, a similar procedure to the DDT-related implementation can be performed. That is: collect approximated DT values for all neighbouring vertices of a current vertex. Then, traverse to the vertex with the highest DT value and iterate. In the limit, this traversal stops at a 'stationary' vertex or at an infinite edge.

In summary, I have presented two different approaches to implementing the Solution 2. The first solution, tracing the DDT, is easy to implement, but suffers from quantisation artefacts. The second solution, traversing the Voronoi partition does not suffer from quantisation artefacts (although the solution is still approximate), but is harder to implement. I can imagine the first approach to be preferred by researchers to demonstrate a concept, as I have done in Chapter 5, and the second implementation to be preferred by developers of commercial products because it is more accurate.

Appendix B

Colour-gradient meshes: additional results

This appendix shows additional visualisations and comparisons. Unless stated otherwise, the interpolating colour function is defined in the CIE Lab colour space.

B.1 Additional results

Figures B.1, B.2, B.3, and B.4 show additional results. In Figures B.1 and B.2, notice how the density of the mesh matters for visual quality.

B.2 Visualisations of underlying control meshes

Figures B.5 and B.6 show computed grids after the ternary subdivision step.

B.3 Comparisons with Illustrator's gradient mesh tool

Figures B.7 and B.8 show comparisons with Adobe Illustrator's gradient mesh tool. Colours were interpolated in the RGB colour space.

B.4 Comparisons with cubic mean-value coordinates

Figure B.9 and B.10 show comparisons with cubic mean value coordinates (CMVC). Colours were interpolated in the RGB colour space.

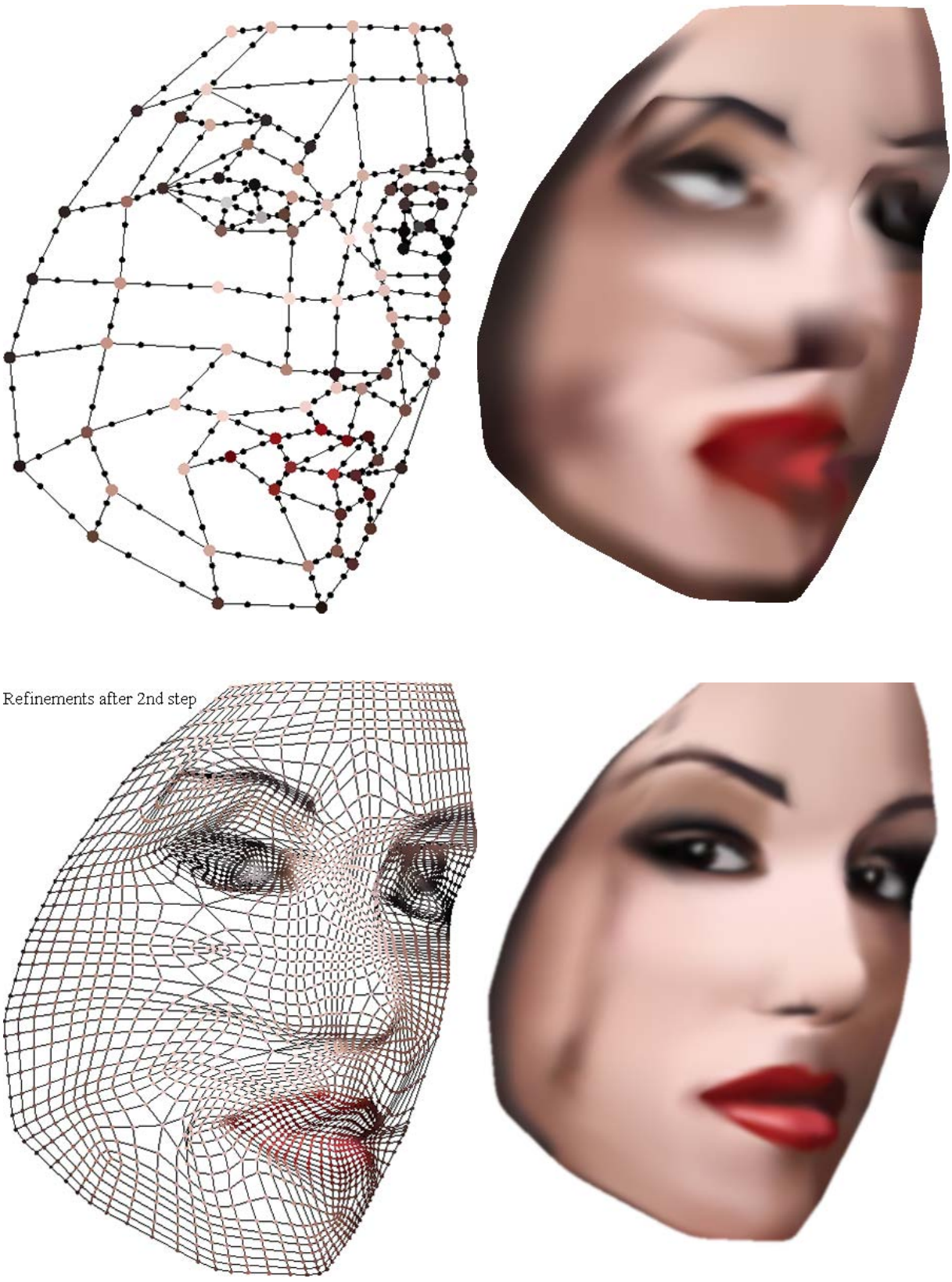


Figure B.1: Additional results.

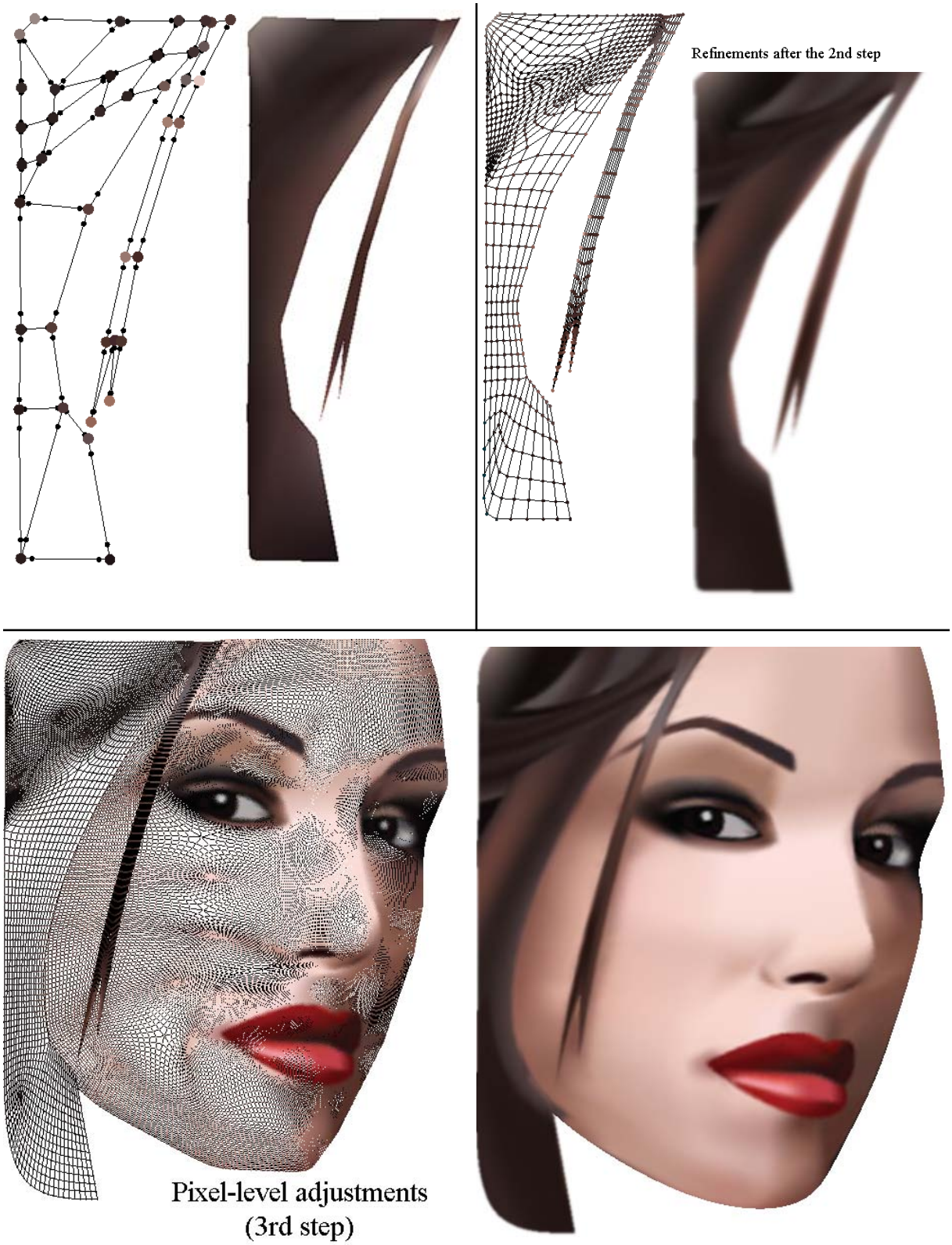


Figure B.2: Additional results.

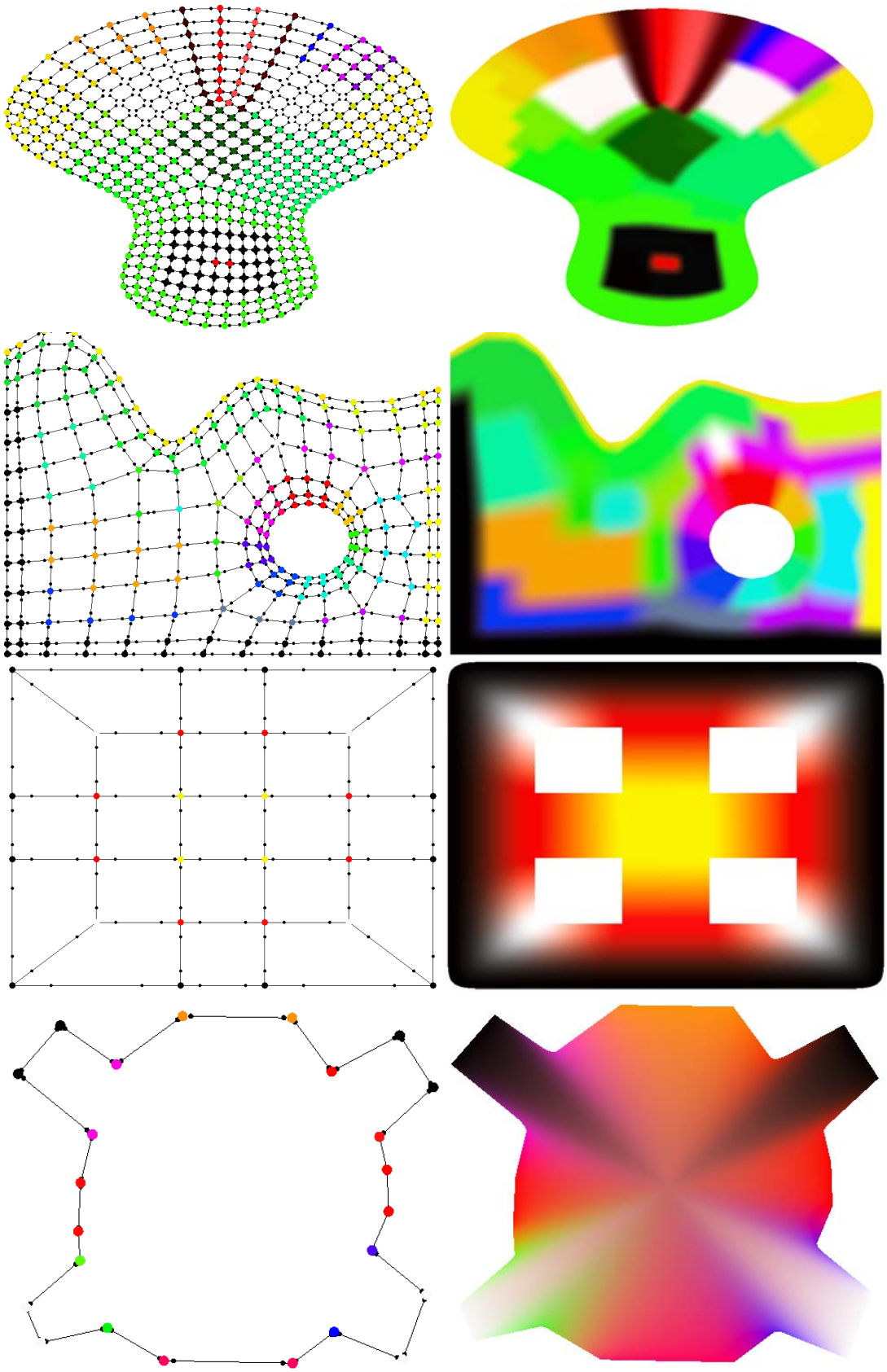


Figure B.3: Additional results: some ‘simpler’ control meshes with primary colours

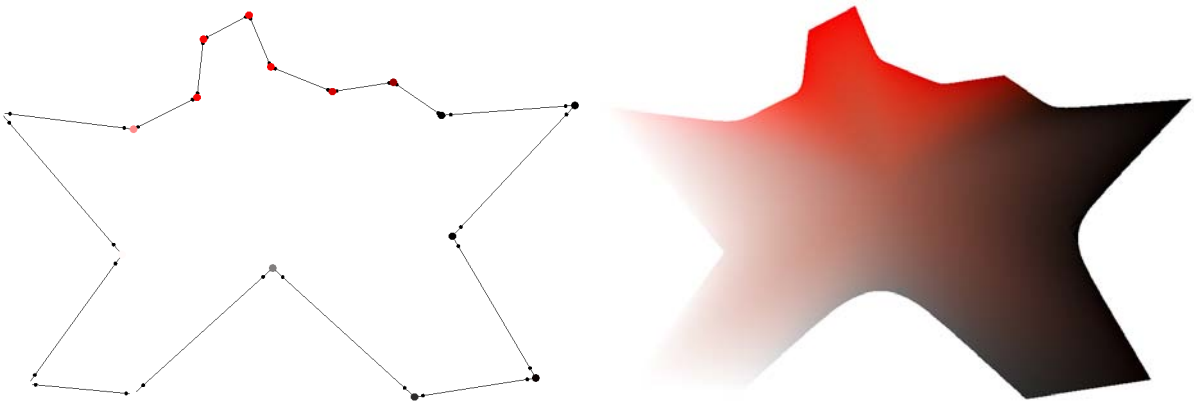


Figure B.4: An additional result.

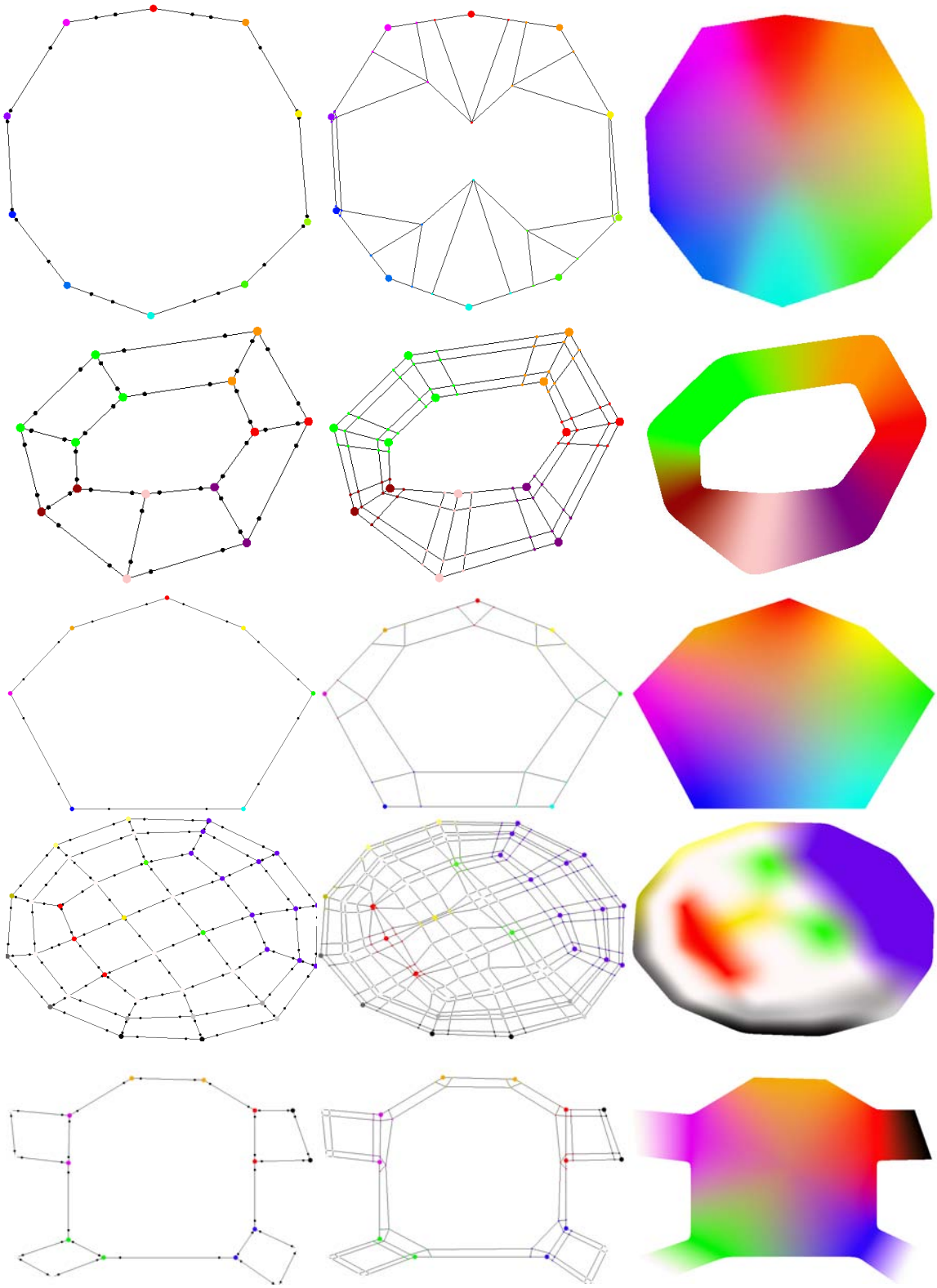


Figure B.5: Grid visualisations. Left: input mesh. Middle: mesh after ternary subdivision. Right: coloured result after two additional steps with Catmull-Clark. The directional colour weights have been manipulated with the smaller black disks (left). These disks represent the location of the new edge points.

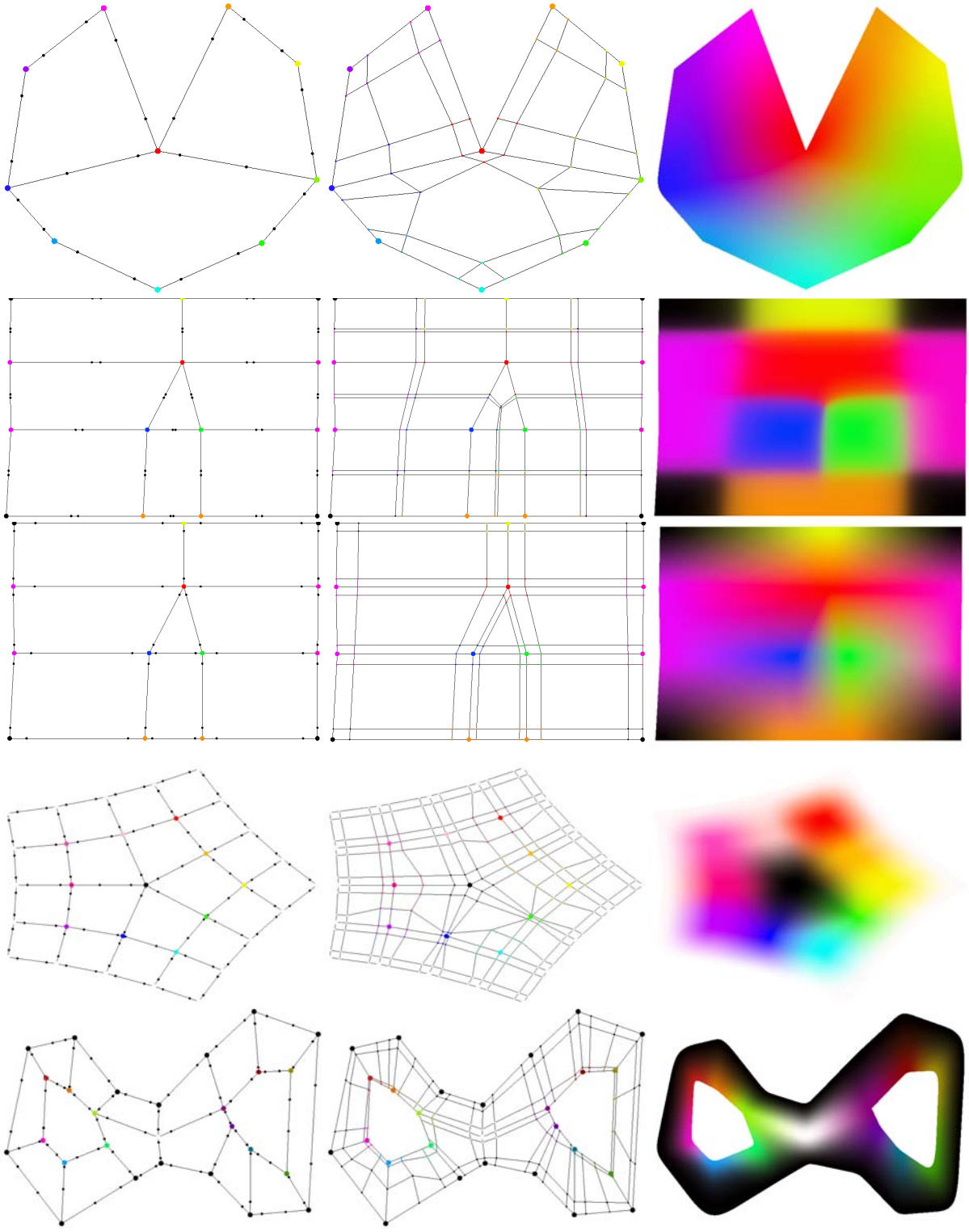


Figure B.6: Grid visualisations. Continued from Figure B.5.

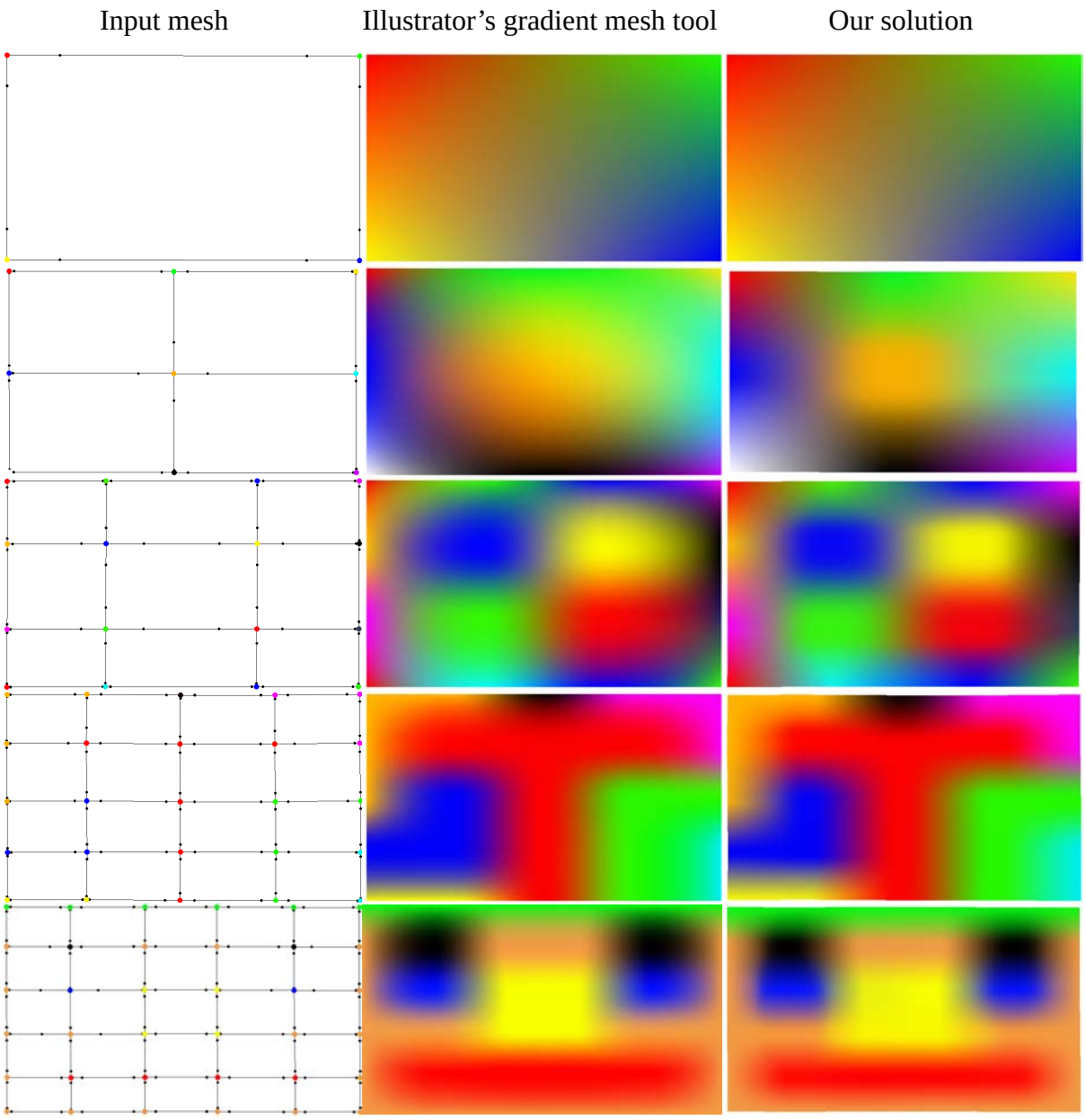


Figure B.7: Comparisons with Illustrator's gradient mesh tool with regular grids. The input is manually defined via both interfaces. The fact that inputs are defined separately in the two interfaces and how they are fed to the interpolation frameworks influence this comparison. For example, Illustrator uses different derivatives for boundary vertices.

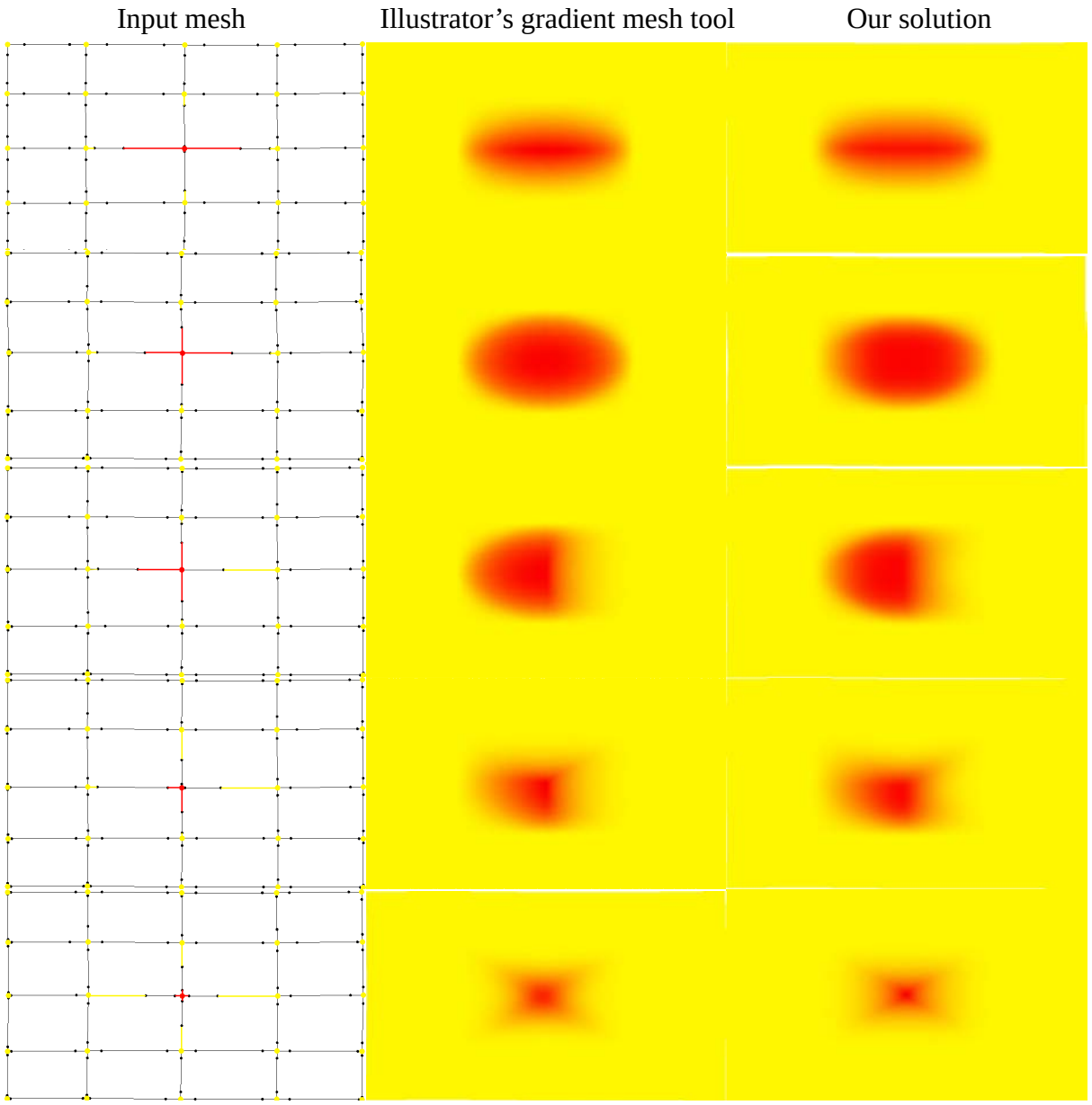


Figure B.8: Various settings for the derivative constraints of Illustrator's tool and comparable settings for the directional colour weights of our method. Edited constraints are highlighted with coloured lines along the edges. We noticed that these derivative constraints in Illustrator are scaled to provide more intuitive colour propagation for the user. Thus, non-identical or mismatched inputs can give rise to visual differences.

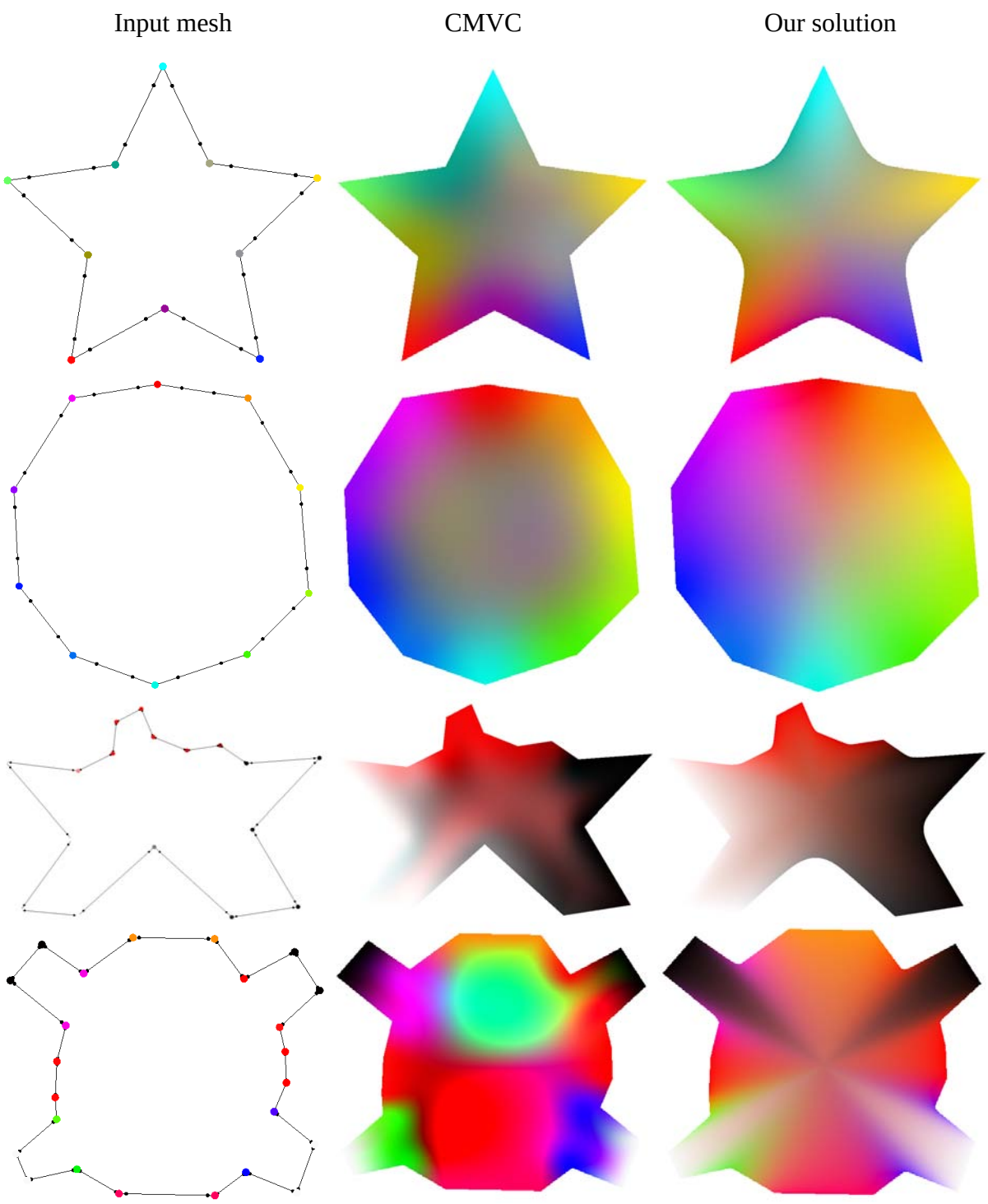


Figure B.9: Comparisons between CMVC and our method with simple polygons. Note that the derivative constraints are set to zero for CMVC. Thus, these images for CMVC can also be viewed as results of mean value coordinates.

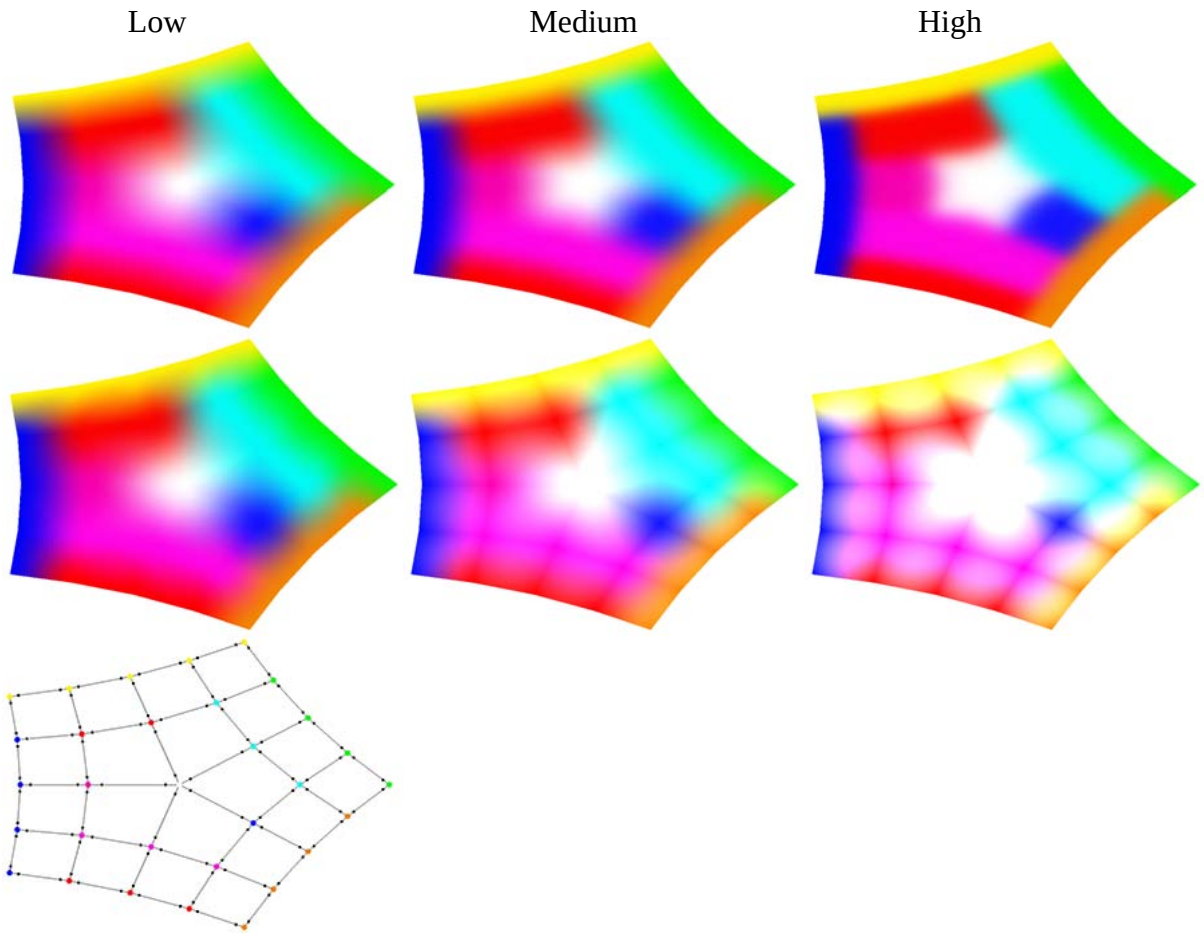


Figure B.10: Comparison between CMVCs' derivative constraints (top) with our directional colour weights (bottom). For CMVCs, the magnitude of the derivatives are increased. The values are increased from left to right. The bottom image shows the input mesh.