# Image Segmentation for Stylized Non-Photorealistic Rendering and Animation

by

Alexander Kolliopoulos

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

# Abstract

Image Segmentation for Stylized Non-Photorealistic Rendering and Animation

Alexander Kolliopoulos

Master of Science

Graduate Department of Computer Science

University of Toronto

2005

This thesis approaches the problem of non-photorealistic rendering by identifying segments in the image plane and filling them using algorithms to render in artistic styles. Using segments as a 2D primitive for non-photorealistic styles is a natural extension of techniques artists often implicitly employ for purposes such as abstraction of unnecessary detail. The problem of segmenting an arbitrary 3D scene in a 2D view using geometric scene information is presented, and a solution based on spectral clustering is proposed. With an acceleration technique, segmentation can be performed in near real-time for interactive, artistic environments. This approach is automatic beyond the setting of segmentation parameters by a user, and it can be extended to temporally coherent non-photorealistic animation by segmenting adjacent frames together. A number of artistic rendering styles are applied within this segmentation framework to demonstrate the effects that such a system makes possible.

# Acknowledgements

This thesis could not have happened without the guidance, ideas, and encouragement of my advisor, Aaron Hertzmann. For reading on such a tight schedule, I owe Allan Jepson my gratitude. I must thank Roey Flor for taking the time to help implement some of the artistic styles that appear in this work. Some DGP name dropping is in order for all the people around the lab who make it a worthwhile place to work: Abhishek, Anand, Anastasia, Azeem, Bowen, Brad, Dan, Eron, Gonzalo, Jack, Jacky, Joe, John, Jonathan, Kevin, Marge, Noah, Patricio, Patrick, Shahzad, Winnie, and all the Mikes. You guys are the best.

# Contents

Chapter 1

# Introduction

*"I may seem to be passionately concerned with the 'hows' of representation, how you actually represent rather than 'what' or 'why'. But to me this is inevitable. The 'how' has a great effect on what we see. To say that 'what we see' is more important than 'how we see it' is to think that 'how' has been settled and fixed. When you realize this is not the case, you realize that 'how' often affects 'what' we see."*

– David Hockney

Visual art is one of the most expressive forms of communication available. Even a novice can create simple drawings that tell a compelling story through little more than a collection of rough strokes that suggest a certain character. Such subtle messages to a viewer are lost in precise photorealistic renderings that preserve physical appearance alone. Some of the first marks left by early humans that survive today still evoke feeling and emotion, even if the original intention of the artist is no longer known. While the ability to produce expressive art comes almost naturally to people, it is still a significant problem to design automatic computational algorithms that produce a wide variety of artistic styles without a great deal of human intervention.

## 1.1 Motivation

Non-photorealistic rendering (NPR) algorithms have been developed both to emulate existing styles of traditional art as well as to allow one to create novel forms of artistic imagery. One interesting use of NPR is to generate artistic renderings from 3D scenes, as opposed to generating renderings based on 3D images. There are a number of ad-

vantages to this approach. For example, rich information can be computed from a 3D scene that might be difficult or impossible to extract from only pixel color information. Moreover, it gives a user the ability to easily generate stylized views of a scene from many different camera orientations. With some degree of frame-to-frame coherency, even non-photorealistic animation is possible. This can give artists the ability to easily generate motion in styles that are otherwise nearly impossible to animate by traditional means. Another interesting use for non-photorealistic animation from 3D scenes is that it can allow an artist to quickly see the effects of rendering a clip in novel styles with little or no additional work. The use of 3D environments also suggests interactive applications, such as games that allow a user to freely explore an artistic world. Finally, any insights gained from designing NPR for 3D might give clues to what techniques are worth pursuing for rendering based on only 2D information.

There is considerable interest in artistic rendering outside of the research community. NPR can be found more and more often in the entertainment industry. Scenes or objects are commonly composited into hand-animated cartoons with cell-style shading and outline contours to emulate the style of manually drawn work, such as in the television series Futurama [42] and Invader Zim [89]. Simple NPR effects are typical in many modern video games as well, with examples of toon shading and contour lines in Jet Set Radio Future [87], Robotech: Battlecry [68], Auto Modellista [14], and countless others. These techniques are fairly straightforward, as the medium encourages simple styles that are not difficult to emulate algorithmically. However, applying artistic rendering algorithms directly to models can result in out of place elements, despite their similar appearance to other hand-drawn elements. For example, motion can be disturbing because it is easy to apply rotations that are more often avoided when animated traditionally on a budget. NPR is finding its way into big budget movies, such as Disney's animated *Tarzan* [13], which uses a system to allow animation through digitally painted backgrounds. Richard Linklater's *Waking Life* [61] and, more recently, *A Scanner Darkly* [60] are rotoscoped,

with a proprietary software system to help animators produce animation from video. Although NPR has been applied in the real world with varying degrees of success, there are many experimental styles of animation that are quite challenging to capture with current automatic techniques. Examples include oil painted short animations by Georges Schwizgebel, such as *L'homme sans ombre* [84], and Alexander Petrov's animated painting adaptation of Hemingway's *The Old Man and the Sea* [75]. Most existing systems, such as those described here, are very simple or require a great deal of interaction with an artist. An exciting area for new work is to design algorithms that automatically produce artistic renderings. Such a system should be capable of real-time interaction, and it should emulate artistic techniques that existing systems are not able to capture. The next section examines many varied styles of art to find a common theme that will form the basis of our system.

## 1.2 Inspiration

An obvious place to look for motivation in designing novel NPR techniques is existing art produced by traditional artists. Furthermore, guidance from artists who are familiar with current work in NPR can provide valuable insight into what is lacking in the field.

### 1.2.1 Segmentation in Art

Examples of the use of segmentation in art are not at all difficult to find, although in some cases it can be subtle. When searching specifically for examples of segmentation in art, one can find a range of applications. Segmentation can appear as part of the process of turning a concept into a finished piece of art. In Figure 1.1, the artist models a watercolor painting after a photograph of a tree. In the first stage of painting shown, it is clear that objects are blocked out to be handled with separate washes of paint, as in the second stage pictured. In the final stage shown, the tree in the foreground still has much

Figure 1.1: Stages of producing a watercolor painting based on a photograph (left), from [74]. Details in the branches, leaves, grass, and background are abstracted away with larger washes that suggest detail without explicitly reproducing it.

of the detail of its leaves and branches abstracted away—it would be tedious for the artist to reproduce every small feature of the tree, and it would likely be visually cluttered to do so. The forest in the background is abstracted away into a single segment, as a wash that flows from one side of the painting to the other. In some areas of this painting, it is difficult to say where one segment ends and another begins, but segmentation certainly played a role in the creation of this piece, and its effects can be seen in the final result.

(a) Wayne Thiebaud's *Around the Cake* (oil on canvas, 1962).

(b) Manually highlighted segments in the painting.

Figure 1.2: Segmentation is based on object features, with strokes built up more heavily around segment boundaries.

Segmentation in art is not limited to watercolor paintings or scenes of nature where there is a great amount of detail for the artist to deal with. Figure 1.2 shows a relatively

(a) *Golden City* by Pat Duke (digital painting, 2004).

(b) Detail of the skyline.

Figure 1.3: Buildings in the background become less detailed and merge together into a single skyline.

simple scene of a cake and slices of cake on plates, made up of only a few fairly regular geometric shapes. Despite the apparent simplicity in the scene, the choice of stroke orientations and texture gives the painting considerable character. Here, the use of segmentation is not only in abstracting away detail, but it is also employed as a stylistic tool, partly due to the medium used.

In Figure 1.3, the artist used segmentation subtly to suggest depth by removing features from buildings that are further from the viewer. However, this reveals another use of segmentation, as detail is not only removed from buildings, but buildings merge together to share shading and a single outline in the background.

A compelling use of segmentation for artistic effect can be seen in Figure 1.4. The artist has merged much of the figure with the background to achieve a unique effect, with each segment primarily a single color, but with texture due to the screenprinting process. Contours between segments increase visual legibility of the segments.

Examples of segmentation can be found in ancient art as well. The black figure style of decorating pottery in Archaic Greece makes use of a clear segmentation of the depicted figures and features. Objects are painted with a black slip over the natural red color of the clay as a first step. Details are then etched in the black areas with a needle, and other

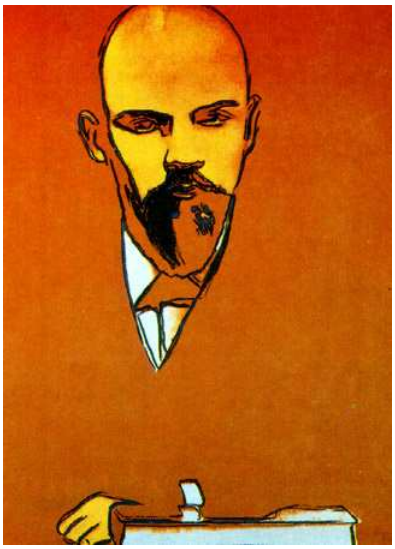


Figure 1.4: Andy Warhol's *Lenin Red* (screenprint, 1987). Few segments are used for artistic effect.

pigments may be added to bring out certain features. The technique creates an effect of striking contrasts, with segments shaded in a single color and separated by the etched contour lines. The constant shading of each segment also suggests a sense of flatness and 2D design similar to that of modern animation. The red figure style developed later but used the same technique of decorating pottery. The main differences in this style are that it reverses the use of slip to indicate background regions instead of foreground features and details are painted rather than etched. The sense of segmentation is just as strong with this approach. 1.5 shows pieces characteristic of each style of art.

Finally, segmentation has been employed by artists to vary detail over time in animation. Figure 1.6 shows a sequence of five frames from an animation where a small town starts in the distance in the initial frame. As the camera draws closer, more detail becomes visible and shapes become more precise. Buildings evolve from a vague collection of shapes to clearly defined structures, and details such as windows appear on the faces of the structures. Similarly, human figures are not rendered until they are within a reasonable distance to the viewer.

Figure 1.5: Left: *The Recovery of Helen by Menelaos*, attributed to Lydos (black figure plate, ca. 560-540 B.C.). Right: *Zeus Chasing Ganymede*, signed by Douris (red figure cup, ca. 480 B.C.).

Figure 1.6: Frames from Georges Schwizgebel's animation, *L'homme sans ombre* (oil and acrylic on cells, 2004). As the viewpoint approaches the town, more detail is painted.

### 1.2.2 Applications of Segmentation

When corresponding with professional illustrator and animator, Pat Duke [29], whose work includes animating for *A Scanner Darkly*, he noted that the most important problem he identifies with automatic NPR systems is the following:

> Level of Abstraction Detail: This is simply an acknowledgment of a traditional artistic convention: artists do not use the same level of abstraction throughout an entire image. To aid in reading a painting, for instance, some elements need to be slightly clearer and more detailed than others. Likewise,

> elements of the image which are unimportant can easily be more abstracted—this helps to focus the viewer on what's really important.

This sentiment is reflected in literature on the subject as well. In *Understanding Comics* [66], artist Scott McCloud explores the importance of segmentation in animation and sequential art as a tool for identifying with viewers. Santella and DeCarlo [83] justify the use of abstraction in NPR based on experiments with eye tracking data, and they give a thorough discussion of the merits of abstraction. They found that viewers tend to spend more time looking at regions that have been abstracted less in artistic renderings, which suggests that abstraction can be used by an artist or NPR system to guide attention to more important regions of an image.

The previous examples have illustrated various ways in which image segmentation is used by artists to achieve abstraction of detail. While there are many other ways an artist may choose to abstract detail, the range of possible abstraction effects that segmentation makes possible indicate that it is a flexible and useful tool for abstraction. Hence, it is reasonable to approach the detail abstraction problem as an image segmentation problem. Applications of segmentation in art other than abstraction of unnecessary detail may be identified from these examples as well. Image segments can create a sense of 2D layout and design, freeing NPR techniques from tightly relying upon 3D content. This can be especially important in animation, where 3D geometry allows animators to create scenes that at times do not feel right, because of the conflicting visual cues of NPR techniques that mimic 2D animation but are rendered on precise 3D geometry. Also, an appropriate segmentation can make the process of producing a piece of art much more efficient, as some segments can be filled with simply a single stroke or wash. Finally, in some styles, segmentation is a consequence of the nature of the medium, such as in batik, woodblock printing, and cartoons, or even in cases such as oil painting, where the artist might build up paint around segment boundaries by being more careful with the strokes. While an artist may not be explicitly aware of the segmentation that results from higher level

artistic processes, it is evident that segmentation can serve a number of useful purposes, which would account for why it can be found in such diverse styles of art.

## 1.3 Contributions

This thesis presents 2D image segments as a primitive for NPR techniques, which can be applied by rendering artistic styles within the segments of a scene. For this purpose, a general, automatic, and fast approach to segmentation from 3D scene information is developed, based in spectral graph clustering techniques from the fields of data clustering and machine vision. Various simple artistic styles that work within this framework of scene segmentation are presented, some of which would be difficult or impossible to develop without an explicit notion of segmentation. The possibility of applying segmentation over a sequence of frames for animation is also explored and evaluated. A strategy of generating frame-to-frame coherence for segments in animation is presented, which is critical for obtaining acceptable animation.

Chapter 2

# Background

*"Art is dangerous. It is one of the attractions: when it ceases to be dangerous you don't want it."*

– Anthony Burgess

---

Non-photorealistic rendering has matured as a field in its own right in the past 15 or so years, while image segmentation has been of interest in computer vision for quite some time. Before we consider a general segment-based approach to NPR, it is beneficial to examine previous related work. This chapter reviews artistic rendering and introduces algorithms for image segmentation. First we review approaches to artistically rendering scenes consisting of 3D models using pen-and-ink and painterly methods. We then evaluate other systems that use segmentation as part of the NPR process. Existing approaches to artistic animation and interfaces for NPR systems are also mentioned. Finally, we compare various data segmentation algorithms.

## 2.1 Non-Photorealistic Rendering

A wide range of techniques have been developed for rendering scenes in artistic styles. A general overview of several techniques is given in [57]. Some image-based algorithms rely solely on pixel data, while other approaches generally work based on 3D scene geometry specified by an artist. Many broad approximations to styles of art are possible, including painterly and pen-and-ink styles. This section presents an overview of many current techniques for generating artistic renderings and animation relevant to this work.

### 2.1.1 Painterly and Pen-and-Ink Rendering of 3D Scenes

Some of the earliest work that approached NPR directly as a means to produce artistic effects using computer renderings includes work by Haeberli [43], which focuses on image-based techniques, such as producing simple painting-like images from photographs, but the author notes that NPR based on raytracing gives one considerably more information about a scene, such as surface normals and depth. This is a fact we will exploit by basing our NPR system on 3D scenes rather than photos or video.

There has been considerable interest in pen-and-ink illustration techniques, which produce artistic renderings of scenes using only black strokes on a white background [31, 48, 81, 94]. Shading is produced in these cases by building up more strokes in darker areas and employing hatching where appropriate. Strokes can also be used to convey shape by using surface curvature information. Some pen-and-ink illustration techniques are appropriate for real-time interaction by preprocessing models [32].

Meier produced some of the earliest painterly renderings from 3D scenes, which use a brush texture painted on particles stuck to the surface of objects [69]. Additionally, reference images such as color, stroke orientation, and stroke size are used to determine properties of the rendered brush strokes. While the technique is conceptually simple, it produces some pleasing results. More convincing painterly effects have been generated by modeling longer, curved strokes [45] and, further, embossing painted strokes to give the painting a feeling of depth [46]. These approaches were applied only to color images, but they could also naïvely be used to paint rendered 3D scenes. Complicated physical models for watercolor simulations are also possible [23], if a proper model for applying brush strokes and washes to a 3D scene is developed.

These systems are largely complementary to that presented in this work—they mainly describe various rendering styles, whereas here we describe techniques for automatically assigning styles to different parts of a scene. Next, we examine more closely techniques that either produce a segmented appearance or explicitly employ segmentation as part

of the rendering process, then we review contour rendering and user interaction in NPR.

### 2.1.2 Segmentation in Artistic Rendering

There has been work in using segmentation as a component of some NPR techniques, whether explicitly approached as such or not. Many image-processing operators, such as posterization, anisotropic diffusion, and vector quantization yield painterly effects with a segmented appearance.

Decorative mosaics can suggest underlying segmentations by changing the orientation of tiles to match contour edges and varying tile color and size gradually within regions [44]. Layering artistically rendered areas can also lead to a segmented appearance [23, 65]. Some media which are naturally segmented, such as batik, can only be simulated well by modeling regions of different colors, due to the physical processes involved in dying the cloth [97]. Levin et al. [58] propose a method of automatically coloring grayscale images based on color hints provided by a user, which gives a segmented appearance; and, in fact, they note that their cost function is minimized by the segmentation cost function we will use in Chapter 3 under different constraints.

Recently, a number of authors have directly used image segmentation algorithms for NPR. DeCarlo and Santella [26] create abstracted image representations based on image segmentation, using eye-tracking data to determine where more detailed segmentation is needed. Each segment is smoothed and rendered as a solid color with occasional edge strokes. Gooch et al. [39] segment an image into very small regions and fit a single paint stroke to each region. Bangham et al. [5] use scale-space filters to attempt to preserve important edges in photographs, then each segment is rendered with a solid color. All of these papers produced very high-quality, appealing results; however, most of the effort in these has gone into obtaining usable segmentations for NPR. As a result, each provides only a small number of rendering styles, if any more than one specific style; and these styles are tightly coupled to the result of segmentation.

This work is different from these previous efforts in three ways. First, the segmentation algorithm used here is deigned for rendering 3D scenes, incorporating 3D geometric information. Second, segmentation is approached as a general primitive for artistic rendering, rather than being tied to a specific style. Third, most of the previous systems use some form of user intervention to produce each image or video. In contrast, the system described in Chapter 3 requires a user to author a 3D scene and rendering style, or use those already available, but it can then automatically render arbitrary new views.

### 2.1.3 Contour Rendering

Since we enhance contour rendering using segmentation, it is helpful to review current approaches to this NPR technique. Contours, sometimes called silhouettes, are curves drawn to convey shape of objects by marking edges between visible and backfacing parts. Here, we will follow the convention of calling these curves contours, while the term silhouette is used to denote curves following outlines of objects that separate them from the background or other distinct objects behind them. A good overview of contour rendering techniques that goes into more detail than we do here is available in [50], but note their definitions of contour and silhouette are reversed from what we use.

Contour rendering has its roots in some of the earliest computer graphics work. In 1967, Arthur Appel introduced contour rendering [3]. Although this work focuses on hidden line removal, it does present some compelling, yet very simple scenes rendered with contour lines drawn. It even goes so far as to allow some special tagged edges, such as creases, to be drawn to increase readability of the image. In 1990, Elber and Cohen [33] developed hidden line removal algorithms for nonuniform rational B-spline surfaces, producing very literal outlined renderings of models that only hint at future artistic applications. Early research in contour rendering as an artistic device can be traced to another work presented in 1990, where Saito and Takahashi [80] increase the visual legibility of rendered objects by using contour lines. In this case, the authors rely

on a number of rendered reference images to produce simple artistic effects, with contour edges being computed directly from a depth image. While this approach is simple and often effective, there are some shortcomings to it. Cases such as a nearly flat object that crosses itself in the image can result in a depth image with very little variation at the crossing. Consider a model of a ribbon, for example. This will cause the edge at the crossing to not be detected. Also, such an image-based method makes accurately detecting contour chains more difficult. A contour chain is a set of adjacent contour edges that can be viewed as a single, long curve for placing stylized strokes. Without these, the range of stylization possible on contours is very limited. However, the depth-image based algorithm is fast and conceptually simple, making it a good first step in contour rendering.

A fast object space algorithm for finding contour edges of polygonal meshes as part of an artistic rendering system is first presented in [64], which essentially makes some improvements to Appel's earlier algorithm. This approach is somewhat complicated, since it works directly on models rather than reference images, but it overcomes problematic cases that can occur otherwise, and it directly gives one contour chains. Most modern work in contour rendering relies on finding contour curves in object space rather than image space [12, 25, 51, 53, 71], as this gives more reliable results, and contours chains can be used to produce effects evocative of different media. For example, the usual solid lines that represent contours can be replaced with textures that appear to give a wavy hand drawn appearance or painted brushed strokes. Furthermore, with object space contours, more nuanced decisions about what to do with visibility information may be made—hidden contours can be rendered in a different color, or as dashed lines, for example.

There are a couple of problems that can arise by directly rendering all contours of an image. First, very detailed scenes can become a mess of dense contours, making it difficult or impossible to tell what an image is supposed to represent in the first place.

Second, it is sometimes computationally prohibitive to model all the detail necessary in a scene to get the contours desired by an artist. Examples of such detail are shingles, bricks, or leaves. Often, solving this second problem by explicitly modeling all the detail necessary leads right back to the first problem of cluttered scenes.

Work in resolving the issues of complexity in contour renderings can be traced back to work by Appel in 1979 [4], where contour lines are haloed as illustrators often do to increase readability of complicated scenes. This just means than important curves can have some safe perpendicular distance within which no other features will be drawn, to help bring them out of the image. However, complexity reduction has not been approached to increase artistic effect until recently.

Work in pen-and-ink illustration is closely related to contour rendering, as pen-and-ink techniques presented by Winkenbach and Salesin [94] can be used to create the appearance of detailed contours on architectural renderings, even when such detail is not actually modeled as geometry. Elber [30] demonstrates pen-and-ink techniques with isoparametric curves that can have more detail focused around contour areas, creating an outlined appearance even though no contours are explicitly modeled. Beyond this, the clutter problem is addressed with procedural textures that can omit detail to better match target tones. This approach is limited mainly to large, regular surfaces, such as brick walls and the like.

Deussen and Strothotte [27] focus on the specific problem of rendering trees, which can be a source of complexity in contour strokes due to the number of leaves that would be necessary to model a convincing tree. They use depth discontinuities to determine where leaves are needed, which are modeled with abstract modeling primitives that attempt to match target tone by drawing more in areas of shadow and visual abstraction by scaling primitive size with distance. While the results are promising, they require explicit modeling of the entire tree, and they are also very domain specific. Another approach is to use graftals [56], which do not require an artist to explicitly model every leaf, and

they can also be applied to other complex models, such as fur or grass. Still, this solves a very specific problem of rendering complexity with contours that is not applicable to many domains.

More recently, Wilson and Ma [93] used a complexity map and several view dependent renderings to find regions of high stroke density for simplification in pen-and-ink renderings. The complexity map in this instance is a blurred rendering of the scene with all contours drawn, which is used to determine where there is a high degree of clutter, and this is used to attempt to match contour density to target tones, similar to the approach of [27]. Grabli et al. [40] also use such a measure of *a priori* density, and their work keeps track of stroke orientations and causal density created by drawing strokes to produce more regular appearing contour renderings.

Most current general approaches to rendering contours on complex scenes rely on keeping track of some form of explicit stroke density measure. Such an approach is compatible with the system we propose, but Chapter 5 proposes an alternative strategy to complexity reduction in contour rendering based on segmentation.

### 2.1.4 Non-Photorealistic Animation

There are a number of problems unique to non-photorealistic animation, due largely to frame-to-frame coherency issues and the amount of manual work that can be necessary to produce animation. Many of the efforts in NPR address such problems related to animation by introducing some automatic coherency tracking or simplifying the user input to generate many frames of renderings.

Work in improving correspondence between features of NPR renderings between frames has been applied to animated movies as well as to completely interactive environments. The work of Meier [69], which uses a particle system on object surfaces, is designed for animation, since the particles stick to a fixed position on an object, creating an effect of strokes being carefully placed to correspond to the same point in space in

adjacent frames. A random selection of points could otherwise lead to distracting noise as strokes pop in and out between frames. While this might be an interesting effect in itself, it would likely become tiring quickly. The principles described by Meier were extended by Daniels [24] in the Deep Canvas system used to animate Disney's *Tarzan*. With this, artists paint backgrounds on a computer, and the strokes are projected to correspond to points on 3D models. Klein et al. [55] use NPR textures at different scales to created a seemingly painted, interactive environment. Contour rendering introduces its own coherency problems, as stroke textures need to correspond between frames, even though contour chains can appear, disappear, bifurcate, or merge between frames. Kalnins et al. [52] handle such problems by propagating contour positions in small neighborhoods while keeping track of stroke direction. This works quite well for keeping contour appearance consistent between frames.

Much of the work in reducing the amount of up-front user input for animation has focused on automating the process of rotoscoping. Rotoscoping is the process of producing frames of animation by tracing objects in a reference sequence of images. This technique has its roots in early hand-drawn animation, where reference footage was traced frame by frame [36]. This traditional approach of tracing by hand requires a tremendous amount of manual work, yet the concept forms the basis of a compelling way to reduce the amount of work necessary for animation. By automatically rotoscoping from video, one can potentially generate animations with little more than a video camera and a computer. This has the advantage of requiring no modeling skill on the part of the user, and equipment is inexpensive. In one approach that requires no motion tracking, successive frames of animation may have strokes painted over only in regions that have changed significantly; an alternative is to use optical flow to move strokes in the direction of scene motion [47, 62]. This is similar to Meier's model of using particles on the surface of objects to track stroke positions between frames. An approach using video segmentation is described by Collomosse and Hall [20], where a video sequence is partitioned into

space-time regions, and each segment is rendered with a solid color, with some temporal coherency between segments. Other works propose animation from video techniques that more heavily involve the user. For example, SnakeToonz [1] requires a user to indicate a few curves in the source video, which are then modeled as Bézier snakes to be tracked from frame to frame. Curve endpoints snap and close to create closed regions that are shaded with solid colors. In [2], the user can iteratively run a process to track curves and hand-tune the resulting curves if necessary. Further, the curves can be stylized as brush strokes and used to fill in regions to create more interesting styles. With video tooning [91], a user outlines objects at keyframes, and a segmentation algorithm is applied to the video volume to track regions, which can then be stylized by a user. Collomosse et al. [19] explore the automatic addition of motion lines and other effects of traditional animation, which could integrate well with other automatic rotoscoping techniques. While the advantages in ease of content creation with animation from video are obvious, it is also currently a limited medium without significant input from artists. However, insights from techniques developed for constructing and working with animation from video can be valuable to any artistic animation system.

### 2.1.5 Interaction

An important problem that must be taken into consideration when designing techniques for NPR is in providing user controls over artistic style. One promising approach is to allow an artist to draw directly within an artistic virtual world. This has the compelling advantage of being intuitive, since one draws directly on objects, and it allows an artist to see the results of changes to a scene immediately. Cohen et al. [17] build a scene based on strokes and gestures drawn directly in a NPR environment, which allows users to creatively manipulate the world by painting on billboards and deforming the ground plane with gestures. The system limits a user to a small range of viewpoints, since the billboarding effect becomes obvious at extreme angles. Bourguignon et al. [8] present

a similar interface, where a user draws on a plane in 3D. In this case, however, the resulting scene consists of strokes in 3D that deform and disappear according to the viewpoint. Since the strokes are not tied to any underlying model, they sometimes deform in strange or unexpected ways, but worlds can still be flexibly navigated. With WYSIWYG NPR [53], a user to draws contour strokes and decals directly on surfaces of 3D models. Features can be annotated at different levels of detail, and some subtle fading as the user transitions between detail levels allows one to navigate the NPR scenes quite freely. The system's interface is quite natural; and it allows one to take shortcuts, such as by giving an example stroke to be used for all contours on an object. While the number of possible styles is limited, the range of variation in style demonstrated with the stroke-based interface is impressive.

Alternatively, NPR styles can be defined by procedural shaders, as [41] proposes a shader system for contour rendering. While this can provide a powerful way to describe contour behavior, it does require the user to write algorithmic scripts, and it is limited to contour rendering. Procedural surface textures can be applied to surfaces in a scene as well, to produce very detailed pen-and-ink renderings [95]. A good interface to a similar system allows users to specify stroke directions on a surface and give example strokes which are used to fill in detail [82]. The user simply specifies direction fields and example strokes, which the system uses to automatically fill in detail on an object.

All of these approaches have their merits, although they are tied tightly to the style of rendering they are designed for. Automatically rendering novel viewpoints is a particularly difficult problem for any interface intended to give an artist control. Some of the works described here handle this well, but there is also the problem of giving an artist too much control. This can lead to a daunting task of annotating large scenes. Some degree of automation can help, but care must be taken, otherwise rendered scenes may appear uniform and monotonous. No previous work has specifically addressed problems of interacting with a NPR system in the context of segmentation, but many of the techniques

described here could be useful when designing artistic control for such a system.

## 2.2 Image Segmentation

Image segmentation is one of the classic problems in machine vision. The problem is to divide an image plane into independent partitions that are perceptually significant. Since there is no unique solution to segmenting an arbitrary image, there have been several proposed approaches to it. Some methods treat points in the image as vectors in some feature space which can be clustered based on a measure of proximity. Another class of methods that has grown in popularity in the past decade treats the image as a graph [96]. Doing so makes it trivial to apply graph theoretic partitioning algorithms to segment images. This section elaborates on some of the details of different approaches to image segmentation.

### 2.2.1 Point-Based Segmentation

There are a variety of algorithms that segment data in a feature space with a metric defined on it. One of the simplest of such algorithms is $K$-means, first suggested in [63]. This is an iterative process that begins with $K$ randomly or user-assigned centers. In each iteration, two steps occur. First, data points are assigned to the nearest center. The second step is to move each center to be the mean of the data points assigned to it. This process continues until convergence, when point assignments no longer change. One problem with this algorithm is that all data points are weighted equally, so outliers can cause problems by pulling means away from a good center. That is because of the strong assumptions $K$-means makes about how the data is distributed. It is only appropriate for very tight, well separated clusters of points that are in roughly hyperspherical shapes. Also, the process is very sensitive to the placement of the initial centers. A poor selection of centers can cause clusters of points to be split between two segments, or some centers

may have no points assigned to them at all. $K$-means is not often applied directly to data to be segmented, but it is sometimes used as a step within more complicated clustering algorithms, due to its simplicity.

An approach that addresses many of the shortcomings of $K$-means is the Expectation-Maximization (EM) algorithm for generating mixtures of Gaussians. EM attempts to optimize the likelihood that the given data was generated by a combination of Gaussian distributions. Like $K$-means, EM requires the number of segments to be determined in advance, and an iterative process fits $K$ Gaussian means and variances to data points. Once the algorithm is stopped, data can be segmented by thresholding to the most likely Gaussian. This algorithm is an improvement over $K$-means in that outliers are better dealt with, and data can lie in long, thin strips without being divided into several segments. However, this approach assumes that data was generated by a number of Gaussians, and very often this is not the case. Hence, any points that lie along a curve other than a line will often be overly segmented. EM has been applied to images in [98], with limited success.

One way to overcome the assumptions of $K$-means and Gaussian mixture models is to use support vector machines, as proposed by Ben-Hur et al. [6]. In support vector clustering, a nonlinear projection by a Gaussian kernel is applied to the data, bringing it into a high-dimensional space. A minimal bounding hypersphere is then computed in this space. Projecting this hypersphere back to the original feature space results in a number of contours enclosing regions corresponding to clusters of points. By ignoring outliers in the high-dimensional space, robust segmentation of point data is possible. Clusters need not conform to simple shapes, sets of points along curves may be clustered appropriately by support vector clustering. Ben-Hur et al. show no results on images, but the algorithm makes a promising candidate for image segmentation.

Another robust approach to segmentation of point data in images is that of mean shift [21]. This uses a density gradient estimation to get a mean shift vector for small

windows of data, which points in the direction of maximum increase in density. This has the effect of shifting local means to a nearby region where most points are. Hence, points may be clustered by iteratively computing mean shift vectors and moving windows until convergence. Since this approach only relies upon the density of data and not any assumptions about the process that generated the clusters, it handles arbitrarily shaped segments of points. Furthermore, the algorithm tends to converge rapidly, making it computationally inexpensive. Mean shift is a very powerful approach to segmenting point data, but like any other point-based approach, it is depends on a measure of distance between points. With graph-based approaches, one has more flexibility to easily define constraints or weights on segmentation that would be difficult to represent with only a distance metric in a feature space. Next, we examine segmentation on graphs and their application to segmenting images.

### 2.2.2 Segmentation as Graph Partitioning

First, we make a few definitions related to graph partitioning clear. In this thesis, a *graph*, $G$, is always considered to be a set of $N$ *nodes*, $V$, and weighted, undirected *edges*, $E$. Pairs of nodes may be connected by an edge, with the weight on an edge being a positive real number which represents the affinity between two nodes. The more similar two nodes are considered, the greater the weight on the edge between them should be. If a node is not reachable from some other node by any path of edges in $G$, the two nodes are said to be in different *components*.

To partition a graph, one typically starts with a graph made up of a single component and removes edges to produce multiple components by making some groups of nodes unreachable from some other groups nodes, which may be called partitions, segments, or clusters. There are two basic approaches to this: removing edges from a graph, or starting with no edges and selecting which edges to add back to the graph. Most algorithms applied to image segmentation use the former method, more specifically using

*graph cuts*. A cut is a set of edges that, when removed, completely partitions a single component into at least two components. Thus, it is not possible to reach any node in one component from any node in the other component by traversing graph edges.

Regardless of the partitioning method used, one must assign some relationship between an image and a graph for any graph segmentation algorithms to be useful. It is common for nodes of a graph to be created such that they have a one-to-one correspondence with pixels in an image. Edges may be created between adjacent pixels, forming a grid, so at most any node has four edges; but it is also possible to let edges connect all pixels within a small neighborhood, at a computational cost for the increased graph size. Creating many edges to nodes within small neighborhoods can be useful for large, noisy images, though. Weights must be selected for the edges as well. These edge weights can be as simple as inverse distance between pixels in a feature space, to create larger weights between pixels that are similar in color and smaller weights between pixels that are dissimilar. Weights might be modified based on user input or some other information to guide the segmentation process. This is one of the strengths of graph-based image segmentation—a metric can be modified locally, for pairs of pixels, without affecting the relationship of these pixels to any others.

Given the open-ended nature of the problem of optimally segmenting a graph, there are a number of algorithms that have been developed to solve this problem. Some techniques use mainly local criteria to quickly make decisions about which edges to keep, such as that of Felzenszwalb and Huttenlocher [34]. This is a constructive algorithm; that is, it selects edges to add to a final graph, rather than removing edges with graph cuts. By greedily adding edges to the graph based on some notion of internal variation within a segment, they satisfy a global notion of preventing oversegmentations and undersegmentations. This algorithm is simple, yet it automatically selects the number of clusters, since it is only concerned with decisions on adding single edges. The results of segmentations on example images tend to be susceptible to noise despite manual pa-

rameter selection, oversegmenting some regions, while others are left undersegmented. However, this approach does indicate that simple algorithms on graphs can reflect global information reasonably well. Algorithms that globally optimize cuts on graphs have the potential to produce improved segmentations by using more information about the graph at once. Some popular graph cut algorithms are those based on maximum flow networks and those based on spectral clustering. We review both techniques next.

### 2.2.3 Minimum Cut Algorithms

A critical factor in any graph partitioning algorithm is the choice of objective function, since there is no one way to define the "optimal" partitioning of a graph. Maximum flow techniques are based on the observation that finding the edges of maximum flow in a network is equivalent to finding the *minimum cut* on an undirected graph [37]. The minimum cut on a graph between two nodes, called *terminals*, is the cut that completely separates the two terminals while having a minimal sum of edge weights. Hence, the minimum cut on a graph between two nodes can be found by finding edges that become saturated when flow is propagated from one terminal to the other. This can be computed in polynomial time by a variety of algorithms [9].

Requiring two terminal nodes is problematic, because there is no straightforward way of selecting them. One might add the terminals as new nodes in the graph and somehow select edge weights from these terminals to nodes in the graph. Another possibility is selecting two nodes already in the graph to act as terminals, but it is not clear which are appropriate. This means we need some *a priori* knowledge of which nodes should belong in the different clusters. Some approaches get around this limitation by requiring user interaction, for example, letting the user label a few background and foreground pixels in an image [11, 78]. While the results can be of a reasonably high quality, there are often times when an image should logically be segmented into more than two partitions, and demanding so much input from a user is undesirable.

There are ways to get around the two-terminal requirement without making assumptions or demanding user input, but they tend to be too slow to consider their use in interactive applications. For example, one could instead solve the multi-terminal problem, which simply takes the minimum cut among all possible pairs of nodes. However, any operation that must examine all possible pairings of nodes is certain to be computationally expensive. Naïvely, this requires solving $T_{N-1}$ maximum flow problems, where $T_x$ is the triangular number equal to the binomial coefficient $\binom{x+1}{2}$. To generate more than two segments, the algorithm can be run again on the resulting segments until some threshold condition is met, which slows down the process even more. Boykov et al. [10] present an approach to the multiple labeling problem in terms of energy minimization. They use an iterative minimum cut process that finds an approximation to the solution of the global optimization problem, but it requires on the order of seconds to segment small images on a modern machine [9]. In [96], Wu and Leahy segment images using a graph representation with the Gomory-Hu algorithm [38], which requires only the solution of $N - 1$ maximum flow problems to solve the multi-terminal minimum cut problem exactly, given a graph with $N$ nodes. This is a significant computational savings over $T_{N-1}$ maximum flow problems; but it is still impractical for real-time interaction, considering that $N$ will typically be on the order of tens or hundreds of thousands. Veksler takes a different approach, finding nested segments by computing closed contours of a small cost around each pixel; but even with optimizations, this approach is fairly slow for small images [90].

Perceptually, a minimum cut is meant to correspond to a boundary between pixels of high variation. However, in its basic form, the minimum cut criteria favors removing fewer edges, since it is not normalized by cut size in any way. It can be optimal to cut edges from a single node rather than cutting a long edge between two clusters of nodes. This can cause very small, spurious segments due to noise. Also, the minimum cut algorithm makes no attempt to maximize similarity within segments, so a great deal

of information in the graph is effectively ignored. For these reasons, and the difficulties in quickly finding global optima for an arbitrary number of segments, we consider another graph-based segmentation technique.

### 2.2.4 Spectral Clustering

In the last few years, spectral graph clustering methods have become popular as a promising approach to graph partitioning. It has been observed that many properties of a graph can be revealed by inspecting the eigendecomposition of a matrix derived from the graph [16]. Spectral clustering methods are those that group nodes based on one or more eigenvectors of a such a matrix corresponding to a graph. An *affinity matrix* (or weight matrix) is constructed, with each row and column corresponding to a numbered node in the graph. The values in this graph reflect the edge weights between pairs of nodes. If nodes have no edge between them, this matrix entry is zero. Typically nodes are not allowed a circular edge, so the diagonal values are all zero as well. Depending on the algorithm, this matrix may be manipulated further, for example by multiplying it by other matrices. Then, eigenvectors of the resulting matrix are computed, and some or all of them are analyzed to partition the graph nodes.

One direct approach, proposed by Perona and Freeman, is to use the eigenvector corresponding to the largest eigenvalue of the affinity matrix [73]. The nodes corresponding to nonzero entries of this eigenvector belong to one segment, while the remaining nodes with zero entries are assigned to the other segment. This is possible due to the observation that if affinity between elements of distinct groups is zero, then the largest eigenvector will have the property of separating elements into two distinct zero and nonzero groups. In practice, of course, the affinity between groups will be greater than zero, and the authors show that the zero eigenvector entries are instead on the order of the affinity between the groups. Hence, in practice, some small threshold on the largest eigenvector may be used to segment a graph if there is sufficient separation between groups to

begin with. This tends to work well on obvious partitionings, but it becomes less clear where to threshold the eigenvector when images do not have a clearly dominant segment. Furthermore, this approach tends to favor tight clusters. In [86], some examples of poor separation by the dominant eigenvector of the affinity matrix are given. Note that in this work, the authors refer to this algorithm as the *average association* formulation of spectral clustering.

Kannan et al. [54] present an approach relying on singular vectors with the goal of maximizing conductance in a clustering, which is meant to increase the importance of vertices that have many similar neighbors. Ng et al. [70] give an algorithm that uses the $K$ largest eigenvectors of a matrix that is related to the affinity matrix's Laplacian. That is, for weight matrix $\mathbf{W}$ and $\mathbf{D} = \text{diag}(\mathbf{v})$, where $\mathbf{v}$ is a vector with each element set to the sum of elements in the corresponding row of $\mathbf{W}$, the eigendecomposition of $\mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$ is computed. After forming a matrix from these $K$ eigenvectors and normalizing the length of each row, the authors treat each row corresponding to a graph node as a point, and cluster them using $K$-means. This approach is based on the fact that in the ideal case where all clusters are completely separated from each other, these points in $K$-dimensional space corresponding to graph nodes will be orthogonal to each other when they belong to different clusters. Some analysis shows that perturbing the weight matrix from this ideal case leads to a rather well behaved perturbation of these points, so that a simpler clustering method can be applied. This seems to give better results on point data than some other techniques, including that of Kannan et al., and it has the bonus of finding $K$ segments simultaneously, for a specified value of $K$. Zelnik-Manor and Perona [100] extend this approach by introducing local scaling and attempting to automatically find the number of segments. They attempt to recover a rotation of the eigenvectors that aligns them with the canonical axes and use this to assign points to clusters. Gradient descent is used to recover this alignment by finding an optimal rotation, which works for cases where separation of segments is clear. The number of

segments is selected by trying many different values of $K$ and selecting that which seems to give the best clustering of points to the canonical axes. On image data, their fully automatic algorithm seems to perform well, but it does tend to pick up small, isolated segments at the image boundary, probably due to the local scaling used. However, for both the $K$-means approach and gradient descent, it is easy to generate many cases on point data that result in very poor segmentations.

One of the most often cited formulations of spectral clustering is that of normalized cuts [86], which is based on a continuous relaxation of a discrete optimization problem. In this case, the eigenvector for the second smallest eigenvalue of $\mathbf{D}^{-1/2}(\mathbf{D}-\mathbf{W})\mathbf{D}^{-1/2}$ is used to partition the graph nodes. It can be shown that this formulation uses both the largest and second largest eigenvectors of a normalized affinity matrix [92]. This extra information leads to much more robust bipartitionings than relying only on the largest eigenvector directly. It can also be shown to be related to the algorithm of Ng et al., which ties together approaches based on matrix perturbation and optimization. In fact, a multiclass variation of normalized cuts exists [99], which solves for $K$ segments simultaneously. This turns out to use exactly the same algorithm as that of Ng et al., although they approach the solution in completely different ways. Since normalized cuts and its multiclass form are the algorithms we have selected for segmenting the image, the next chapter describes them in more detail.

These spectral clustering techniques are not appropriate for real-time interaction because calculating eigenvectors of an $N \times N$ matrix can be quite slow for large $N$, since this operation is on the order of $O(N^3)$ for dense matrices. Most implementations dealing with large graphs use sparse matrix eigendecomposition algorithms such as the Lanczos algorithm, which is designed to give an approximate solution for sparse, symmetric matrices. This has a complexity of $O(mN)$, where $m$ is the maximum number of matrix-vector computations required, typically less than $O(\sqrt{N})$ [86]. Despite the relative speed of the Lanczos algorithm for sparse, symmetric matrices, spectral clustering algorithms

can take several seconds or even minutes on a fast machine, since $N$ can quickly grow to hundreds of thousands. Some have attempted to deal with this by oversegmenting the graph with a fast, linear time algorithm, and treating the segments as nodes for normalized cuts [76]. As we will see in the next chapter, this direct approach to reducing the graph size is flawed. Others have approached normalized cuts with a multiscale algorithm [85], but this too runs on the order of seconds. One compelling approach is to compute eigenvectors on a coarse representation of the original matrix and interpolate to get an approximation to the eigendecomposition, as in the work of Chennubhotla and Jepson [15]. While this proves to be significantly faster than other sparse matrix eigenvector solvers, the running time on a fast machine is on the order of seconds. Although spectral clustering presents challenges in its computational complexity, it is possible to accelerate the algorithm significantly to the point of interactive rates on simple scenes. This is demonstrated in the next chapter.

Chapter 3

# Segmenting the Image Plane

*"The artist is nothing without the gift, but the gift is nothing without work."*

– Emile Zola

---

In this chapter, we present an approach to segmenting an image based on 3D scene information. We review an approach to graph segmentation based on spectral clustering, and it is adapted to work at near-interactive rates.

## 3.1 A Face Adjacency Graph

Since we are not segmenting photographs, but rather rendered 3D scenes, we have much more information available than typical image or video segmentation. A scene is rendered from a single mesh made up of any number of separate objects. The faces of the mesh are always triangular, with textures specified by texture coordinates at the face vertices. By rendering a triangle ID image, we can identify the face and object that occupies any pixel. Other auxiliary buffers may also be rendered, such as a depth image, as in [80].

### 3.1.1 Edge Weights

We first consider what the weight on edges should be if we are segmenting an image with additional information provided by the explicit knowledge of the 3D scene being rendered. There is one graph node for each pixel in the image, and edges are introduced between nodes corresponding to adjacent pixels. Edge weight is determined by pixel

*affinity*—the more strongly-related two pixels are, the greater the weight on their shared edge.

Typically, the affinity between pixels in image segmentation is set to some variation on $\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|/(2\sigma^2))$, where $\mathbf{x}_i$ and $\mathbf{x}_j$ are feature vectors corresponding to nodes $i$ and $j$ [58, 73, 86, 92, 100]. This feature vector can be as simple as the vector in RGB color space for each pixel. Since we have a complete description of the 3D geometry, we can modify the feature vector to take advantage of this information. We also take advantage of the fact that pixels are represented in a graph, which does not require that nodes be related by a distance metric that satisfies the triangle inequality. This allows us to specify precise changes in weight between two nodes without affecting the relationships of those nodes to any other nodes in the graph.

To calculate affinities between pixels, we render a number of reference images: a color image of the scene, a depth map, a triangle ID reference image, and an importance map. In the triangle ID reference image, each face in the mesh is rendered with a unique color, so that we can quickly determine what face of the mesh is rendered to any pixel of the image, as in [56]. From the triangle ID, a lookup table can be used to determine which scene object each pixel belongs to. This table is computed once when the mesh is loaded. For the importance map, each object in the scene may be optionally tagged in advance as being more or less "important." The importance map is generated by rendering the scene with each pixel shaded in proportion to its importance, so unimportant elements are rendered in black and important elements are white. The user tags objects by painting to grayscale texture maps, so objects can even vary in importance over their surface.

Given these reference images, we can define a feature vector $f_i$ for each pixel $i$:

$$f_i = \left( w_c r_i, w_c g_i, w_c b_i, \frac{w_z}{z_i + \beta} \right)^T . \tag{3.1}$$

where $(r_i, g_i, b_i)$ is the color of the pixel in the unit cube, $z_i$ is the depth of the pixel,

$w_c$ and $w_z$ are user-defined weights, and $\beta$ is a user-defined bias. Expressing depth in the feature vector as $1/(z_i + \beta)$ treats objects that are far from the viewer as having similar depth, even though the true difference in depth between them may be greater than that of objects closer to the viewer. This models how artists often group distant objects together, even if their relative depths differ greatly.

The weight on the edge between graph nodes $i$ and $j$ is

$$w(i,j) = \exp\left(-\left(\|f_i - f_j\|^2 + c\right)\sigma_{ij}\right). \tag{3.2}$$

The constant $c$ expresses the fact that no two adjacent pixels are exactly the same, since they occupy different positions in 2D. The scaling parameter $\sigma_{ij}$ consists of three terms: $\sigma_{ij} = o_{ij}g_{ij}s_{ij}$. The weight $o_{ij}$ is used to separate different objects in the scene—if pixels $p_i$ and $p_j$ belong to different objects, as determined using the triangle ID reference image and object lookup table, then $o_{ij}$ is set to $\exp(w_o)$; if the object IDs are the same, then $o_{ij} = 1$. That is,

$$o_{ij} = \begin{cases} 1 & \text{if } p_i \text{ and } p_j \text{ belong to the same object,} \\ \exp(w_o) & \text{where } w_o \geq 0 \text{ otherwise.} \end{cases} \tag{3.3}$$

This has the effect of weakening edges connecting nodes between two different objects in the scene when the user sets $w_o > 0$. We use $\exp(w_o)$ rather than $w_o$ directly so that setting $w_o$ to zero will result in $o_{ij} = 1$ everywhere. Hence object IDs will be ignored completely in calculating edge weight when $w_o = 0$, making the user-set parameters more consistent.

Group IDs are used with $g_{ij}$ in a similar fashion. Objects may be tagged with a group ID by the user, by simply selecting them while navigating a scene in 3D and inputing a group ID number. Then, if $p_i$ and $p_j$ belong to different groups, $g_{ij} = \exp(w_g)$, otherwise

$g_{ij} = 1$. Hence,

$$g_{ij} = \begin{cases} 1 & \text{if } p_i \text{ and } p_j \text{ belong to the same group,} \\ \exp(w_g) & \text{where } w_g \geq 0 \text{ otherwise.} \end{cases} \tag{3.4}$$

For example, this allows the user to specify that a group of bushes should be segmented together before being segmented with another nearby object that happens to have the same color.

Finally, the parameter $s_{ij}$ is used to emphasize important objects by encouraging them to be segmented, and to prevent unimportant objects from being overly segmented. Each pixel $p_i$ has an associated importance value $s_i \in [0, 1]$, as determined from the importance map. The weight is defined as $s_{ij} = (\max(s_i, s_j))^{w_s}$, where $w_s$ is the user-specified weight for importance. Hence, an edge between two nodes with small $s_i$ and $s_j$ will be strengthened, but there will be little or no effect on the edge if either of the nodes are considered important.

This gives a user a total of six parameters to set to control the segmentation of a scene, in addition to defining any group IDs and object importance. However, a user can ignore any parameters that are not considered to be necessary—one could simply set all but $w_c$ (the weight for color) to zero and have affinity equivalent to that which is usually used in image segmentation. The other parameters may be viewed as tools to refine a segmentation, as color alone cannot be expected to give reasonable results all the time.

### 3.1.2 Constructing a Face Adjacency Graph

Rather than segment an a scene at the pixel level, we create a *face adjacency graph*. In this graph, each node is associated with a visible face of the mesh that makes up the scene. Edges connect nodes of faces that are adjacent in the image plane. Hence, our primitive for segmentation is a face of the mesh rather than a pixel of the image. This

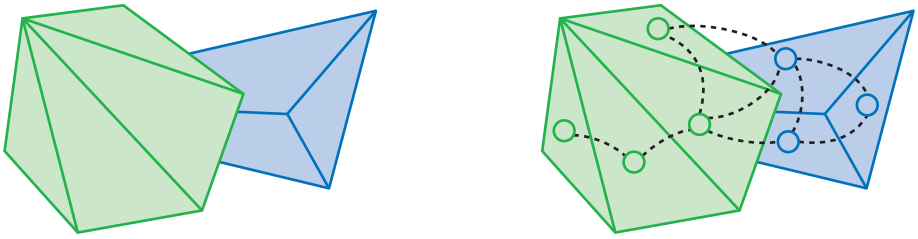
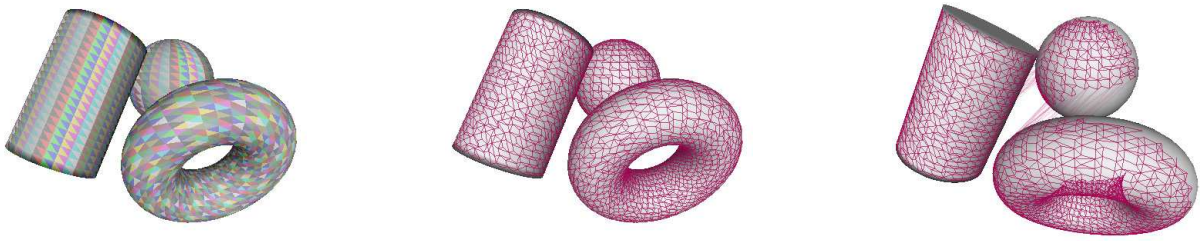


Figure 3.1: An example of a face adjacency graph for two objects. Circles represent nodes in the graph corresponding to faces and dashed lines represent edges between faces that are adjacent in image space.

prevents faces from being split between two or more segments, which can be convenient when rendering some artistic styles. Adjacent faces in the image can be visited simply by walking through the graph, even if the faces belong to completely different objects. This also results in a graph that will require much less memory if rendered faces tend to take up more than a single pixel. This makes operations on the graph faster, since the number of nodes and edges is typically reduced.

To quickly generate the face adjacency graph, we render a triangle ID reference image, where each pixel is given a color that indicates the unique identification number of the triangle that is rendered to that pixel for the current camera orientation. This triangle ID image can be scanned over quickly to find which triangles are adjacent in the image plane. For each pixel, if the pixel to the right or below has a different triangle ID, those faces' nodes are connected with an edge in the face adjacency graph. The time required to build the graph this way is negligible compared to the time taken to calculate eigenvectors and segment the graph. Figure 3.2 shows a face adjacency graph generated using this technique, with 4,791 edges rather than the 319,200 edges that would be required for full 400x400 images, about a 98.5% reduction in edges.

This technique for building the face adjacency graph is a quick approximation to the planar map employed by Winkenbach and Salesin [95]. Since the planar map the authors generate is not dependent upon the rendering resolution, it provides more precise



(a) The faces of a mesh, shown in different colors.

(b) The adjacency graph for this view.

(c) The graph rendered from another angle.

Figure 3.2: An adjacency graph for a simple scene. Nodes of the graph are faces of the mesh, rendered as the average position of the vertices of each face, and edges connect nodes that are adjacent in the image plane.

adjacency information. For example, faces that occupy less than a pixel might not be rendered, so they would be ignored in the graph in our implementation. Another minor error in our algorithm is that occasionally triangles that meet at a vertex will have one pair of adjacent pixels, even though they share no edge. However, the overhead of determining adjacency information with scenes consisting of a large number of triangles is prohibitive to real-time interaction. The faults in the image-based approach to building a face adjacency graph are not significant enough to warrant using a more precise algorithm, since they correspond to very small details that have little impact on the resulting graph.

## 3.2 Normalized Cuts

The normalized cut criterion for graph partitioning, introduced by Shi and Malik [86], minimizes the ratio of the value of a cut to a segment's measure of self-similarity. In contrast, max flow algorithms only minimize the cut and make no guarantees about the relative affinity between pixels within a segment. Previous spectral techniques only minimized cuts with respect to the number of nodes within the segments generated, but this addresses the number of nodes in each segment, not the similarity between nodes in the same segment. The approach of balancing cut cost and region self-similarity is

similar to that of ratio regions, which seeks to minimize the cost of a cut divided by the sum of edge weights of a segment produced by the cut [22].

Let $A$ and $B$ be any two disjoint sets of nodes that partition a graph $G$. The *cut* between $A$ and $B$ has a value defined as

$$\mathsf{cut}(A, B) = \sum_{u \in A, v \in B} w(u, v), \tag{3.5}$$

where $w(u, v)$ is the weight on the edge between nodes $u$ and $v$. If there is no edge between $u$ and $v$, $w(u, v) = 0$. Similarly, the *association* between $A$ and $V$ is

$$\mathsf{assoc}(A, V) = \sum_{u \in A, t \in V} w(u, t), \tag{3.6}$$

where $V$ is the set of all nodes in $G$, so $V = A \cup B$. The definition of the value of association is exactly the same as the value of a cut, the only difference in these two values is in semantics. The *normalized cut* between $A$ and $B$ is

$$\mathsf{Ncut}(A, B) = \frac{\mathsf{cut}(A, B)}{\mathsf{assoc}(A, V)} + \frac{\mathsf{cut}(A, B)}{\mathsf{assoc}(B, V)}. \tag{3.7}$$

To segment a graph using the normalized cut criterion, one partitions the graph into segments $A$ and $B$ in a manner that minimizes $\mathsf{Ncut}(A, B)$. This discourages small, isolated segments from being created by simultaneously minimizing the value of a cut while maximizing the association of each segment, ensuring that they both have a suitable number of similar nodes.

Finding the minimal normalized cut is NP-complete, but an approximate solution can be obtained by using a spectral method. Let $\mathbf{x}$ be an indicator vector with $N$ elements, one corresponding to each node in $G$, where $x_i = 1$ if node $i$ is in segment $A$, and $x_i = -1$ otherwise. Let $\mathbf{D}$ be the *degree matrix*, a diagonal $N \times N$ matrix with $\mathbf{d}$ the vector of $\mathbf{D}$'s diagonal elements: $d_i = \sum_j w(i, j)$. Then $\mathsf{Ncut}(A, B)$ may be written in terms of $\mathbf{d}$,

$\mathbf{x}$, and edge weights $w(i,j)$, since

$$\begin{aligned} \mathsf{cut}(A,B) &= \sum_{x_i>0,x_j<0} w(i,j), && (3.8)\\ \mathsf{assoc}(A,V) &= \sum_{x_i>0} d_i, \text{ and} && (3.9)\\ \mathsf{assoc}(B,V) &= \sum_{x_i<0} d_i, && (3.10) \end{aligned}$$

we have

$$\mathsf{Ncut}(A,B) = \frac{\sum_{x_i>0,x_j<0} w(i,j)}{\sum_{x_i>0} d_i} + \frac{\sum_{x_i>0,x_j<0} w(i,j)}{\sum_{x_i<0} d_i}. \quad (3.11)$$

Let $\mathbf{W}$ be the symmetric *weight matrix*, with $w_{ij} = w(i,j)$. Using this matrix, we can derive $\mathsf{Ncut}(A,B)$ without the need for summations. For notational convenience, let $\mathsf{Ncut}(\mathbf{x}) = \mathsf{Ncut}(A,B)$, $\mathsf{cut}(\mathbf{x}) = \mathsf{cut}(A,B)$, $\mathsf{assoc}_1(\mathbf{x}) = \mathsf{assoc}(A,V)$, and $\mathsf{assoc}_{-1}(\mathbf{x}) = \mathsf{assoc}(B,V)$.

First, we find $\mathsf{cut}(\mathbf{x})$ in terms of $\mathbf{d}$ and $\mathbf{W}$. Consider that

$$\begin{aligned} \mathsf{cut}(\mathbf{x}) &= \sum_{x_i>0,x_j<0} w_{ij} && (3.12)\\ &= \sum_{x_i>0} d_i - \sum_{x_i>0,x_j>0} w_{ij}. && (3.13) \end{aligned}$$

In other words, if we take the sum of edge weights from all nodes in $A$ and remove those edges that connect to other nodes in $A$, this is equal to the sum of edges between $A$ and $B$. Let $\mathbf{z} = \mathbf{x} + \mathbf{1}$, where $\mathbf{1}$ is an $N \times 1$ vector of ones. Hence, $z_i = 2$ when $x_i > 0$ and $z_i = 0$ when $x_i < 0$. Then, we have

$$\begin{aligned} \sum_{x_i>0} 4d_i - \sum_{x_i>0,x_j>0} 4w_{ij} &= \sum_{i\in[1,N]} d_i z_i^2 - \sum_{i,j\in[1,N]} w_{ij} z_i z_j && (3.14)\\ &= \mathbf{z}^T\mathbf{D}\mathbf{z} - \mathbf{z}^T\mathbf{W}\mathbf{z} && (3.15)\\ &= \mathbf{z}^T(\mathbf{D}-\mathbf{W})\mathbf{z}. && (3.16) \end{aligned}$$

Combining Equations 3.12 through 3.16, we have

$$\mathsf{cut}(\mathbf{x}) = (\mathbf{1}+\mathbf{x})^T(\mathbf{D}-\mathbf{W})(\mathbf{1}+\mathbf{x})/4. \tag{3.17}$$

Next, with $k = \left(\sum_{x_i>0} d_i\right) / \left(\sum_i d_i\right)$, we can derive the following:

$$\begin{aligned}
\mathsf{assoc}_1(\mathbf{x}) &= \sum_{x_i>0} d_i && (3.18)\\
&= \frac{\sum_{x_i>0} d_i}{\sum_i d_i} \sum_i d_i && (3.19)\\
&= \frac{\sum_{x_i>0} d_i}{\sum_i d_i} \mathbf{1}^T\mathbf{D}\mathbf{1} && (3.20)\\
&= k\mathbf{1}^T\mathbf{D}\mathbf{1}, && (3.21)
\end{aligned}$$

and

$$\begin{aligned}
\mathsf{assoc}_{-1}(\mathbf{x}) &= \sum_{x_i<0} d_i && (3.22)\\
&= \frac{\sum_{x_i<0} d_i}{\sum_i d_i} \sum_i d_i && (3.23)\\
&= \frac{\sum_i d_i - \sum_{x_i>0} d_i}{\sum_i d_i} \sum_i d_i && (3.24)\\
&= \left(1 - \frac{\sum_{x_i>0} d_i}{\sum_i d_i}\right) \sum_i d_i && (3.25)\\
&= \left(1 - \frac{\sum_{x_i>0} d_i}{\sum_i d_i}\right) \mathbf{1}^T\mathbf{D}\mathbf{1} && (3.26)\\
&= (1-k)\mathbf{1}^T\mathbf{D}\mathbf{1}. && (3.27)
\end{aligned}$$

Therefore, combining Equations 3.17, 3.21, and 3.27, we may rewrite Equation 3.7 as

$$\mathsf{Ncut}(\mathbf{x}) = \frac{(\mathbf{1}+\mathbf{x})^T(\mathbf{D}-\mathbf{W})(\mathbf{1}+\mathbf{x})}{4k\mathbf{1}^T\mathbf{D}\mathbf{1}} + \frac{(\mathbf{1}+\mathbf{x})^T(\mathbf{D}-\mathbf{W})(\mathbf{1}+\mathbf{x})}{4(1-k)\mathbf{1}^T\mathbf{D}\mathbf{1}}. \tag{3.28}$$

It can be shown that, if we relax the problem to allow elements of $\mathbf{x}$ to take on any

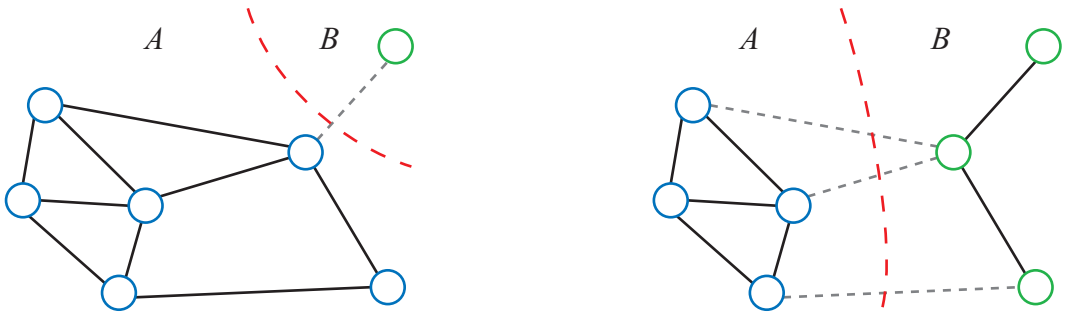



Figure 3.3: Examples of possible cuts on a graph. *Left:* A result that the minimal cut might give, isolating a single node. *Right:* A result typical of normalized cuts, taking segment association into account to give a more reasonable partitioning.

real value, $\mathsf{Ncut}(\mathbf{x})$ is minimized by the eigenvector corresponding to the second smallest eigenvalue of the matrix $\mathbf{D}^{-1/2}(\mathbf{D}-\mathbf{W})\mathbf{D}^{-1/2}$ [86]. A detailed derivation of this result can be found in [76]. This solution can be converted into a graph partition to find an approximate minimized normalized cut by thresholding $\mathbf{x}$, so that element of the eigenvector above the threshold are set to 1 and the rest are set to $-1$. Since there are at most $N-1$ possible threshold values, it is usually reasonable to search over all possible values for that which minimizes $\mathsf{Ncut}(\mathbf{x})$.

To segment a graph into more than two partitions using this technique, we can first compute a two-way partition of the graph, and then recursively apply the same normalized cut algorithm to each segment, as in [88]. The process stops when $\mathsf{Ncut}(\mathbf{x})$ exceeds a user-defined threshold, and no cut is made. This approach of selecting the segments might be troubling because of its greedy nature. At each step, the most optimal cut is made, which may not lead to the best global partitioning once all recursive cuts have been completed. What is needed is a measure of fitness of all cuts in a scene rather than individual cuts and a way to quickly optimize this measure. An extension of normalized cuts to handle the case of an arbitrary number of segments is explored in Section 3.4.

## 3.3 Condensed Graphs

Segmenting graphs at different scales has been applied with some success to speed up the segmentation process significantly at a loss of some accuracy [85]. Here, we consider a simple approach to reducing the graph size without dramatically affecting the final segmentation.

### 3.3.1 Condensing a Graph

Even with the reduced size of the face adjacency graph, the graph will probably be too large to be segmented within a few seconds for meshes with any detail. In the worst case of a large, detailed scene, it might be that no face takes more than a single pixel, resulting in essentially the same graph as the regular grid on the pixels.

To cope with these potentially large adjacency graphs, we introduce another graph, the *condensed graph*. This graph serves the same purpose of speeding up segmentation as the condensed graph described in [86], however we generate it by using a simple region-growing algorithm. Each edge in the face adjacency graph is examined, and if it is strong enough, the two nodes it connects are added to the same condensed node. Once all face adjacency graph nodes are assigned to a condensed node, edges are added between condensed nodes if they have face graph nodes with edges between them. In other words, edges with weight above a threshold are collapsed. This corresponds to an assumption that edges that are "strong enough" usually will not be cut. While there can certainly be cases where this assumption does not hold, in practice, a reasonable threshold can be selected to get a good segmentation while vastly reducing the graph size. By increasing this threshold, a user can obtain faster performance, although at the cost of some possible under segmentation.

We would like to weight the edges of the condensed graph such that a normalized cut on it is equivalent to the corresponding normalized cut on the full face adjacency

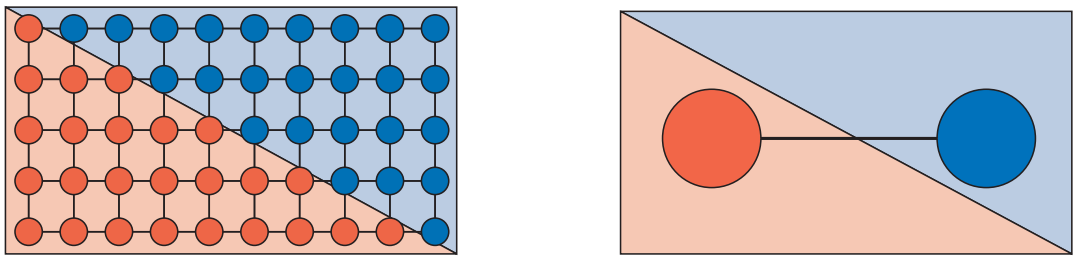


Figure 3.4: *Left:* Image graph produced from a rendering of two triangles. Each node in the graph corresponds to a pixel, and edges connect adjacent pixels. *Right:* Naïvely condensed graph, with only an edge between a node for each region.

graph it represents. Simply weighting the condensed edges as the sum of the weights of the edges they represent is not sufficient, as can be seen with a simple example. A naïve approach to clustering this condensed graph would be to apply the normalized cuts algorithm directly. Unfortunately, this gives a poor approximation to clustering the original graph. Consider an image made up of a red region and a blue region, as in Figure 3.4. In general, the optimal normalized cut for this image should separate the two polygons. However, suppose we condensed this graph to two nodes, one corresponding to each polygon, with a single condensed edge with weight $w$ between them. In this case, there is one possible cut, which has a value of

$$\begin{aligned} \mathsf{Ncut}(A,B) &= \frac{\mathsf{cut}(A,B)}{\mathsf{assoc}(A,V)} + \frac{\mathsf{cut}(A,B)}{\mathsf{assoc}(B,V)} && (3.29)\\ &= \frac{\mathsf{cut}(A,B)}{\mathsf{cut}(A,B)} + \frac{\mathsf{cut}(A,B)}{\mathsf{cut}(A,B)} && (3.30)\\ &= 2. && (3.31) \end{aligned}$$

Thus, this normalized cut has a value of 2, regardless of the edge weights in the underlying adjacency graph. Since this is the maximum possible cost of a cut in every graph, this edge will never be cut, as it will exceed any sensible user-specified threshold for the cost of a cut. However, one should expect the edge between the two regions to be cut—the

edges between the regions have a small weight, and edges within each region have a very large weight. Hence, the cost of the cut would be low relative to the association of either region, resulting in a low normalized cut value.

From our simple example, it is easy to see that the condensing resulted in throwing away information from the majority of the edges in the graph. The solution is to keep track of each condensed node's *internal association*; that is the sum of the weights on the edges of the adjacency graph internal to the condensed node. The next section justifies this and shows how to use the internal association in the context of normalized cuts to segment a condensed graph.

### 3.3.2 Condensed Normalized Cuts

Let $G' = (V', E')$ be a condensed graph that represents a graph $G = (V, E)$ at a reduced size. Each node in $V'$ (called a *condensed node*) corresponds a set of nodes in $V$. For nodes $u \in V$ and $u' \in V'$, we say that $u \in s(u')$ if $u$ is assigned to condensed node $u'$. That is, $s(u')$ is the set of nodes in $V$ assigned to the condensed node $u'$. If nodes $u, v \in V$ are connected by an edge and $u \in s(u')$, $v \in s(v')$, where $u' \neq v'$, then $u'$ and $v'$ are connected by a *condensed edge*. Suppose we have arbitrary partitions $A'$ and $B'$ of $G'$. Then, there are corresponding partitions $A$ and $B$ of $G$ such that $u \in A$ if and only if $u \in s(u')$ and $u' \in A'$. We use the notation $s(A')$ to refer to the union of $s(u')$, for all $u' \in A'$. Hence, $u \in s(u')$ and $u' \in A'$ implies that $u \in s(A')$. With weights on condensed edges equal to the sum of the weights of their corresponding set of edges, we have $\mathsf{cut}(A', B') = \mathsf{cut}(A, B)$. This can be shown by the following:

$$\mathsf{cut}(A, B) = \sum_{u \in A, v \in B} w(u, v), \tag{3.32}$$

and

$$\begin{aligned}
\mathsf{cut}(A', B') &= \sum_{u' \in A', v' \in B'} w(u', v') && (3.33)\\
&= \sum_{u' \in A', v' \in B'} \left( \sum_{u \in s(u'), v \in s(v')} w(u, v) \right) && (3.34)\\
&= \sum_{u \in s(A'), v \in s(B')} w(u, v) && (3.35)\\
&= \sum_{u \in A, v \in B} w(u, v). && (3.36)
\end{aligned}$$

Thus, since we know that, in general, $\mathsf{Ncut}(A', B') \neq \mathsf{Ncut}(A, B)$, it must be the case that $\mathsf{assoc}(A', V') \neq \mathsf{assoc}(A, V)$. Note that $\mathsf{assoc}(A, V) = \mathsf{assoc}(A, A) + \mathsf{cut}(A, B)$, and $\mathsf{assoc}(A', V') = \mathsf{assoc}(A', A') + \mathsf{cut}(A', B')$. Then we have $\mathsf{assoc}(A, A) \neq \mathsf{assoc}(A', A')$, in general. This is due to the fact that we lose weights from edges between nodes that are assigned to the same condensed node. More formally,

$$\begin{aligned}
\mathsf{assoc}(A, A) &= \sum_{u, v \in A} w(u, v) && (3.37)\\
&= \sum_{u, v \in s(A')} w(u, v) && (3.38)\\
&= \sum_{u', v' \in A'} w(u', v') + \sum_{u' \in A'} \left( \sum_{u, v \in s(u')} w(u, v) \right) && (3.39)\\
&= \mathsf{assoc}(A', A') + \sum_{u' \in A'} \left( \sum_{u, v \in s(u')} w(u, v) \right). && (3.40)
\end{aligned}$$

Hence, making a slight modification to the measure of association between the condensed nodes in a partition and all condensed nodes for $G'$ will give us the same value for any

corresponding cut on $G$. That is,

$$\begin{aligned} \mathsf{Ncut}'(A', B') &= \frac{\mathsf{cut}(A', B')}{\mathsf{assoc}(A', V') + \mathsf{intassoc}(A')} + \frac{\mathsf{cut}(A', B')}{\mathsf{assoc}(B', V') + \mathsf{intassoc}(B')} \quad (3.41) \\ &= \mathsf{Ncut}(A, B), \quad (3.42) \end{aligned}$$

where

$$\mathsf{intassoc}(A') = \sum_{u' \in A'} \left( \sum_{u,v \in s(u')} w(u, v) \right). \quad (3.43)$$

Therefore, the value of a condensed normalized cut on a condensed graph, using the definition of $\mathsf{Ncut}'$ given, is the same as the value of the corresponding normalized cut on the underlying graph. That is, $\mathsf{Ncut}'(A', B') = \mathsf{Ncut}(A, B)$. Further, the minimal condensed normalized cut will correspond to the minimal normalized cut under the constraint that no edges may be cut that have both nodes in the same condensed node. This means that minimizing condensed normalized cuts is equivalent to minimizing normalized cuts under certain cut constraints on strong edges.

This suggests an approach to using the graph for the full image, which is never explicitly built, when generating the face adjacency graph. The graph for the full image may be condensed to the face adjacency graph as it is constructed, without loss in the normalized cuts segmentation algorithm. Edge weights are calculated on the edges between pixels, as described in Section 3.1.1. Pixels that belong to the same face in the mesh may be condensed into the same node in the face adjacency graph, keeping track of any internal association of the rendered face. Once it is constructed, the face adjacency graph can be condensed further. This condensed graph can be segmented, propagating the segmentation back to the face adjacency graph when complete.

In practice, we build the face adjacency graph directly by iterating over the image, introducing new nodes whenever a new triangle ID is visited, and adding weights to either the graph edges for each node or their internal association. We then collapse edges between faces with large average weight, since edges can span several pixels, to

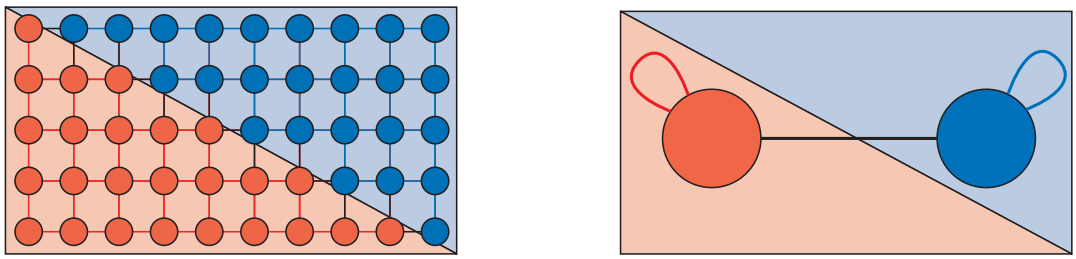


Figure 3.5: A condensed graph corresponding to its underlying graph by including self-referencing edges to condensed nodes.

find the nodes of the condensed graph. Each node in the face adjacency graph stores the condensed node it corresponds to. Then, each edge in the face adjacency graph is scanned over, and its weight is added to the appropriate condensed edge if necessary. If the nodes it joins belong to the same condensed node, the weight is added to the condensed node's internal association.

Another way to look at this solution is as augmenting $G'$ by adding an edge from every condensed node to itself, as shown in Figure 3.5, and setting the weight of this edge to the sum of the edges between the collapsed nodes:

$$w(u', u') = \sum_{i,j \in s(u')} w(i, j). \tag{3.44}$$

The cost of these self-edges are automatically included when computing $\mathsf{assoc}(A', V')$ or $\mathsf{assoc}(B', V')$. Applying the normalized cuts algorithm to this augmented $G'$ is straightforward, with the only necessary modification being that nodes can have an edge to themselves. This is verified in Section 3.3.3.

In [76], the author takes a similar approach to quickly oversegmenting the graph before applying normalized cuts, however it is assumed that the condensed graph can be treated as an approximation of the full graph. As will be demonstrated with experiments in Section 3.5, however, this assumption can lead to significant errors in clustering without

using our definition of condensed normalized cuts.

### 3.3.3 Condensed Spectral Clustering

Now we consider what changes are required to Shi and Malik's spectral approach to normalized cuts to handle condensed graphs with internal associations. For condensed normalized cuts, we have essentially replaced the standard definition of association with the condensed association,

$$\begin{aligned} \mathsf{assoc}'(A', V') &= \sum_{u' \in A', t' \in V'} w(u', t') + \sum_{u' \in A'} \sum_{u, v \in s(u')} w(u, v) && (3.45) \\ &= \mathsf{assoc}(A', V') + \mathsf{intassoc}(A'), && (3.46) \end{aligned}$$

where $A' \subseteq V'$. We first give definitions of $\mathbf{D}'$ and $\mathbf{W}'$ that satisfy this definition of condensed association when used in Equation 3.28. We then show that these matrices have the property that $\mathbf{D}'^{-1/2}(\mathbf{D}' - \mathbf{W}')\mathbf{D}'^{-1/2}$ is positive semidefinite. Then the rest of the derivation of finding the optimal partition in [86] follows from these two properties.

For a condensed graph, set the diagonal elements of $\mathbf{W}'$ equal to their corresponding nodes' internal association, and compute $\mathbf{D}'$ from $\mathbf{W}'$. That is, $w'_{ij} = w(i, j)$, for $i \neq j$, but set the diagonal elements to $w'_{ii} = \mathsf{intassoc}(i)$. For the diagonal of $\mathbf{D}'$, compute the degrees from the rows of $\mathbf{W}'$, so $d'_{ii} = \sum_j w'_{ij}$. We motivate this solution as follows.

Let $\ddot{\mathbf{D}}$ be the degree matrix and $\ddot{\mathbf{W}}$ the weight matrix derived from the naïvely condensed graph without internal associations. That is, $\ddot{\mathbf{W}}$ has zeros on its diagonal and $\ddot{d}_{ii} = \sum_j w(i, j)$. Since the association depends only on the degree matrix (Equations 3.21 and 3.27), we modify $\ddot{\mathbf{D}}$ to include internal associativity for each condensed node. Instead of $\ddot{d}_{ii} = \sum_j w(i, j)$, the condensed degree matrix $\mathbf{D}'$ is the diagonal matrix with diagonal elements

$$d'_{ii} = \sum_j w(i, j) + \sum_{u, v \in s(i)} w(u, v). \tag{3.47}$$

Then, if $\mathbf{x}'$ is the segment indicator vector for condensed nodes and $k' = \left(\sum_{x'_i>0} d'_i\right) / \left(\sum_i d'_i\right)$, we have

$$\begin{aligned} k'\mathbf{1}^T\mathbf{D}'\mathbf{1} &= \frac{\sum_{x'_i>0} d'_i}{\sum_i d'_i} \sum_i d'_i && (3.48)\\ &= \sum_{x'_i>0} d'_i && (3.49)\\ &= \sum_{x'_i>0} \left( \sum_j w(i,j) + \sum_{u,v\in s(i)} w(u,v) \right) && (3.50)\\ &= \sum_{x'_i>0} \sum_j w(i,j) + \sum_{x'_i>0} \sum_{u,v\in s(i)} w(u,v) && (3.51)\\ &= \mathsf{assoc}(A', V') + \mathsf{intassoc}(A'). && (3.52) \end{aligned}$$

This is exactly what we want. A similar derivation follows for the other segment's association, $(1-k')\mathbf{1}^T\mathbf{D}'\mathbf{1}$.

However, the degree matrix is also used to calculate the value of $\mathsf{cut}(A', B')$ (Equation 3.17), so it will be affected by the changes to $\mathbf{D}'$. To fix this, we set the diagonal elements of $\mathbf{W}'$ to the internal association of the corresponding condensed nodes. The changes to the diagonal elements in $\mathbf{D}'$ are subtracted out, since

$$\mathbf{D}' - \mathbf{W}' = (\ddot{\mathbf{D}} + \mathbf{P}) - (\ddot{\mathbf{W}} + \mathbf{P}) = \ddot{\mathbf{D}} - \ddot{\mathbf{W}}, \tag{3.53}$$

where $\mathbf{P}$ is the diagonal matrix with the internal association for each corresponding node on the diagonal. That is, $p_{ii} = \mathsf{intassoc}(i)$ and $p_{ij} = 0$ for $i \neq j$. Therefore, we have the correct value for any cut on a condensed graph using $\mathbf{D}'$ and $\mathbf{W}'$:

$$\begin{aligned} (\mathbf{1}+\mathbf{x}')^T(\mathbf{D}'-\mathbf{W}')(\mathbf{1}+\mathbf{x}') &= (\mathbf{1}+\mathbf{x}')^T(\ddot{\mathbf{D}}-\ddot{\mathbf{W}})(\mathbf{1}+\mathbf{x}') && (3.54)\\ &= 4\mathsf{cut}(A', B'). && (3.55) \end{aligned}$$

From Equations 3.52 and 3.55, we have that $\mathbf{D}'$ and $\mathbf{W}'$ give us exactly the value of

condensed normalized cuts as defined in Equation 3.41.

We now show that $\mathbf{D}'^{-1/2}(\mathbf{D}' - \mathbf{W}')\mathbf{D}'^{-1/2}$ is positive semidefinite. Consider that $\ddot{\mathbf{D}} - \ddot{\mathbf{W}}$ is already known to be positive semidefinite. Then, from Equation 3.53, so too is $\mathbf{D}' - \mathbf{W}'$. Thus, $\mathbf{D}'^{-1/2}(\mathbf{D}' - \mathbf{W}')\mathbf{D}'^{-1/2}$ is also positive semidefinite because for any vector $\mathbf{v}$ of appropriate size,

$$\begin{aligned} \mathbf{v}^T\mathbf{D}'^{-1/2}(\mathbf{D}' - \mathbf{W}')\mathbf{D}'^{-1/2}\mathbf{v} &= \mathbf{v}^T(\mathbf{D}'^{-1/2})^T(\mathbf{D}' - \mathbf{W}')\mathbf{D}'^{-1/2}\mathbf{v} && (3.56) \\ &= (\mathbf{D}'^{-1/2}\mathbf{v})^T(\mathbf{D}' - \mathbf{W}')(\mathbf{D}'^{-1/2}\mathbf{v}) && (3.57) \\ &= \hat{\mathbf{v}}^T(\mathbf{D}' - \mathbf{W}')\hat{\mathbf{v}} \geq 0, && (3.58) \end{aligned}$$

for $\hat{\mathbf{v}} = \mathbf{D}'^{-1/2}\mathbf{v}$. Therefore, these definitions for the condensed weight matrix $\mathbf{W}'$ and the condensed degree matrix $\mathbf{D}'$ allow us to use the standard approach of the spectral normalized cuts algorithm. These definitions are also consistent with the idea of adding self-referencing edges to condensed nodes, weighted by their internal association, and using the matrices derived from that graph without modification.

## 3.4 Simultaneous Multiclass Segmentation

We have so far only considered two-way cuts that separate graphs into the two dominant regions. As was discussed earlier, this greedy approach to segmentation can give poor global segmentations. Yu and Shi have suggested a multiclass approach that finds $K$ segments simultaneously by optimizing one objective function [99]. We examine this approach and suggest improvements to it in this section.

### 3.4.1 Multiclass Normalized Cuts

In multiclass normalized cuts, rather than selecting two partitions to minimize the cut value and maximize their associations, we select $K$ partitions that optimize a measure

of average fitness for each partition. Yu and Shi define this measure for a segment $A$ as $\mathsf{linkratio}(A)$, where

$$\mathsf{linkratio}(A) = \frac{\mathsf{assoc}(A, A)}{\mathsf{assoc}(A, V)}. \tag{3.59}$$

The value of $\mathsf{assoc}(A, A)$ is high when a segment is very self-similar, while a smaller value of $\mathsf{assoc}(A, V)$ corresponds to a better separation between the segment and the rest of the graph. Hence, increasing $\mathsf{linkratio}(A)$ for a segment $A$ encourages it to contain similar points while encouraging a cut between dissimilar points. Therefore, the objective function is

$$\mathsf{knassoc}(\Gamma) = \frac{1}{K} \sum_{l=1}^{K} \mathsf{linkratio}(V_l, V_l), \tag{3.60}$$

the *$K$-way normalized association*, where we aim to maximize this function by partitioning a set of graph nodes $V$ into subsets of nodes $\Gamma = \{V_1, V_2, \ldots, V_K\}$.

We can see that, in fact, this definition of multiclass normalized cuts is exactly two-way normalized cuts for the case where we set the number of segments to find to 2. Let $K = 2$ and $\Gamma = \{A, B\}$. Then we have

$$\begin{aligned}
\mathsf{knassoc}(\Gamma) &= \frac{1}{2}\left(\mathsf{linkratio}(A, A) + \mathsf{linkratio}(B, B)\right) && (3.61)\\
&= \frac{1}{2}\left(\frac{\mathsf{assoc}(A, A)}{\mathsf{assoc}(A, V)} + \frac{\mathsf{assoc}(B, B)}{\mathsf{assoc}(B, V)}\right) && (3.62)\\
&= \frac{1}{2}\left(\frac{\mathsf{assoc}(A, V) - \mathsf{cut}(A, B)}{\mathsf{assoc}(A, V)} + \frac{\mathsf{assoc}(B, V) - \mathsf{cut}(A, B)}{\mathsf{assoc}(B, V)}\right) && (3.63)\\
&= \frac{1}{2}\left(1 - \frac{\mathsf{cut}(A, B)}{\mathsf{assoc}(A, V)} + 1 - \frac{\mathsf{cut}(A, B)}{\mathsf{assoc}(B, V)}\right) && (3.64)\\
&= 1 - \frac{1}{2}\left(\frac{\mathsf{cut}(A, B)}{\mathsf{assoc}(A, V)} + \frac{\mathsf{cut}(A, B)}{\mathsf{assoc}(B, V)}\right) && (3.65)\\
&= 1 - \frac{1}{2}\mathsf{Ncut}(A, B). && (3.66)
\end{aligned}$$

Therefore, maximizing the $K$-way normalized association for $K = 2$ is equivalent to minimizing the normalized cut value. This suggests a spectral approach to optimizing the objective function as well as using graph condensing to reduce the size of the data

to work with.

The condensing strategy used for two-way normalized cuts works as expected for $K$-way cuts. We can justify this with the following. For a set of nodes $V$ with partitions $V_l$ for $l \in [1, K]$, let $V'$ be a condensed set of nodes, and let $V_l'$ be a partition of $V'$ such that $s(V_l') = V_l$. Then,

$$\begin{aligned} \mathsf{knassoc}(\Gamma) &= \frac{1}{K}\sum_{l=1}^{K}\frac{\mathsf{assoc}(V_l, V_l)}{\mathsf{assoc}(V_l, V)} && (3.67)\\ &= \frac{1}{K}\sum_{l=1}^{K}\frac{\mathsf{assoc}(V_l', V_l') + \mathsf{intassoc}(V_l')}{\mathsf{assoc}(V_l', V') + \mathsf{intassoc}(V_l')} && (3.68)\\ &= \frac{1}{K}\sum_{l=1}^{K}\frac{\mathsf{assoc}'(V_l', V_l')}{\mathsf{assoc}'(V_l', V')}. && (3.69) \end{aligned}$$

This can be summarized as follows. Assume that each condensed node has an associated set of nodes that all belong to exactly one segment. Then association of a segment with itself is the same as the association of the corresponding condensed segment with itself plus the internal association within its condensed nodes. The association of any segment with the entire graph is similarly defined as the association of that segment's corresponding condensed nodes with the condensed nodes of the full graph, plus the internal association within the segment's condensed nodes. The association within condensed nodes outside the segment can be ignored because any edges from inside the segment to outside nodes are captured in the condensed edges. In both cases, these values match the definition of condensed association given earlier. Hence, adding circular edges to condensed nodes with internal association as their weight gives us exactly the same optimization problem, under the constraint that no cuts may be made within condensed nodes. This allows us to accelerate $K$-way cuts by working on a reduced size graph as before, which can result in great computational savings in any optimization algorithm.

Now we briefly introduce existing spectral clustering techniques that others have applied to find an approximate optimal partitioning for multiclass normalized cuts. Posing

the optimization problem in terms of the weight matrix $\mathbf{W}$ and degree matrix $\mathbf{D}$ is perhaps more straightforward for multiclass normalized cuts than two-way normalized cuts. Instead of using a single indicator vector that is thresholded to give segment assignments to nodes, we use a binary $N \times K$ partition matrix $\mathbf{X}$, which is 1 exactly once in each row, corresponding to the segment ID assigned to the node for that row.

If $\mathbf{X}_l$ is the vector formed by the column of $\mathbf{X}$ corresponding to segment $V_l$, then we have

$$\mathsf{assoc}(V_l, V_l) = \mathbf{X}_l^T \mathbf{W} \mathbf{X}_l. \tag{3.70}$$

To see why this is so, consider that $\mathsf{assoc}(V_l, V_l) = \sum_{u \in V_l} \left( \sum_{v \in V_l} w(u, v) \right)$, that is, the total sum of weights for each node in $V_l$ of edges that have another node in $V_l$. Then $\mathbf{W}\mathbf{X}_l$ gives us a column vector where each element is the sum of edge weights from the corresponding node to all nodes in $V_l$,

$$\mathbf{W}\mathbf{X}_l = \begin{bmatrix} w(1, v_1) + w(1, v_2) + \cdots \\ \vdots \\ w(N, v_1) + w(N, v_2) + \cdots \end{bmatrix} = \begin{bmatrix} \sum_{v \in V_l} w(1, v) \\ \vdots \\ \sum_{v \in V_l} w(N, v) \end{bmatrix}, \tag{3.71}$$

where $v_i \in V_l$. Multiplying this vector by $\mathbf{X}_l^T$ on the left selects the sum of only those rows that correspond to nodes in $V_l$, resulting in the correct summation.

We can also find that

$$\mathsf{assoc}(V_l, V) = \mathbf{X}_l^T \mathbf{D} \mathbf{X}_l. \tag{3.72}$$

Since $D$ is a diagonal matrix where each diagonal entry is the sum of edge weights from the corresponding node, $\mathbf{D}\mathbf{X}_l$ is a vector where nodes in $V_l$ have an entry equal to their degree, and entries for nodes not in the segment are zero. Multiplying this by $\mathbf{X}_l^T$ on the left selects the sum of degrees of nodes in $V_l$. This is exactly the sum of all edge weights from each node in the segment.

Thus, the optimization problem can be stated as for a fixed $K$, maximizing

$$\mathsf{knassoc}(\mathbf{X}) = \frac{1}{K}\sum_{l=1}^{K}\frac{\mathbf{X}_l^T\mathbf{W}\mathbf{X}_l}{\mathbf{X}_l^T\mathbf{D}\mathbf{X}_l}, \tag{3.73}$$

subject to the constraint that $\mathbf{X}$ is a binary partition matrix.

Yu and Shi show that by manipulating Equation 3.73, this may be relaxed into a continuous domain to find an global solution using eigenvectors of the Laplacian as with two-way normalized cuts in [99]. The spectral algorithm requires the eigenvectors corresponding to the $K$ largest eigenvalues of $\mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$. This matrix appears quite different from that used for greedy normalized cuts, but they are in fact solving the same problem. Consider that the generalized eigendecomposition problem for the multiclass case is

$$\mathbf{W}\mathbf{x} = \mu\mathbf{D}\mathbf{x}. \tag{3.74}$$

For the greedy case, we instead have $(\mathbf{D}-\mathbf{W})\mathbf{x} = \lambda\mathbf{D}\mathbf{x}$. However, with some small manipulations, we can find the following:

$$\begin{aligned}
(\mathbf{D}-\mathbf{W})\mathbf{x} &= \lambda\mathbf{D}\mathbf{x} && (3.75)\\
\mathbf{D}\mathbf{x}-\mathbf{W}\mathbf{x} &= \lambda\mathbf{D}\mathbf{x} && (3.76)\\
\mathbf{W}\mathbf{x} &= \mathbf{D}\mathbf{x}-\lambda\mathbf{D}\mathbf{x} && (3.77)\\
\mathbf{W}\mathbf{x} &= (1-\lambda)\mathbf{D}\mathbf{x}. && (3.78)
\end{aligned}$$

In other words, since $(1-\lambda) = \mu$, the eigenvectors in the multiclass formulation are exactly the eigenvectors of the greedy algorithm, just taken in reverse order of eigenvalue.

The $K$ largest eigenvectors of $\mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$ are used to form the columns of a matrix, where each row represents a point in $\mathbb{R}^K$ corresponding to a node of the graph. These rows are then normalized to have a length of one, so that all points lie on the surface of a hypersphere, resulting in a continuous global optimum $\tilde{\mathbf{X}}^*$. Any rotation of this solution

will also be optimal, so the problem reduces to finding a rotation of $\tilde{\mathbf{X}}^*$ that gives a binary discretization close to the global optimum.

This turns out to be solving the multiclass spectral clustering problem in exactly the same way as Ng et al. in [70], although the approach used to justify the algorithm is quite different in each case. These works further differ in that Ng et al. suggest using $K$-means to cluster the points on the hypersphere, while Yu and Shi favor a more complicated but much better-performing method. Observing that groups are indicated by orthogonal axes, we can attempt to find a rotation that best aligns the data with the canonical axes in $K$-dimensional space. Yu and Shi accomplish this by a two step iterative process. First, they assign each point to its nearest axis by non-maximum suppression. This simply sets all elements in a vector to 0 except a single largest element, which is set to 1. This has the effect of thresholding points to the nearest axis. Second, singular value decomposition is used to find a rotation that best matches the points to this assignment of axes, and the points are rotated. These two steps are repeated until convergence. Using the orthogonality information in this way greatly improves the quality of the discretization over $K$-means; and the algorithm tends to converge very quickly, making it fast as well.

### 3.4.2 Selecting Orthogonal Clusters

While the non-maximum suppression/rotation algorithm for assigning points to clusters tends to give high quality results near a global optimum, it does tend to fail in some cases, namely when the non-maximum suppression step assigns no points to an axis. This causes the segmentation algorithm to return fewer than the requested $K$ number of segments. In practice this is rare, but common enough to be problematic when applying segmentation to frames of animation. We can not fall back on $K$-means, as it can suffer the same problem with stranded centers that are assigned no points, and the quality in clustering results tend to be unacceptable in all but the simplest of cases. Hence, we suggest an alternative approach based on Hochbaum-Shmoys clustering [49].

The Hochbaum-Shmoys algorithm takes a greedy approach to grouping point data. It is designed to approximately minimize the maximum distance between the center of a cluster and any point belonging to it. Note that the algorithm is not specifically designed for spectral clustering. There are two steps to this algorithm—selecting centers and assigning points to centers. First, a random point is selected and labeled as the first center. Each of the remaining $K-1$ centers are then selected by picking a point that has the maximum smallest distance to any of the centers that have been selected so far. That is, center $\mathbf{x}_i$ is a point $\mathbf{p}$, selected by

$$\mathbf{x}_i = \arg\max_{\mathbf{p}} \left\{ \min_{j \in [1,i-1]} d(\mathbf{x}_j, \mathbf{p}) \right\}, \tag{3.79}$$

where $d$ is a function that returns some measure of distance between points. All points are then assigned to their nearest centers.

It is clear that Hochbaum-Shmoys guarantees at least one element in each of the $K$ segments, unlike $K$-means and non-maximum suppression. Furthermore, even with a random starting point and fixed centers, this algorithm performs very well on point data. Non-maximum suppression tends to give slightly higher (better) numerical scores for knassoc in our experiments when it does not fall into a degenerate case, but the difference is usually only in a handfull of points in ambiguous regions that, perceptually, could make sense in either grouping. Furthermore, one can construct cases where the Hochbaum-Shmoys clustering method gives a higher numerical score. Hochbaum-Shmoys is a good candidate for automatic segmentation because it guarantees non-degenerate clusterings, performs reasonably well, and completes in a predictable polynomial time.

We may use the orthogonality constraint on our data to further improve the quality of clusterings, however. Rather than selecting fixed centers that are as far apart in a Euclidean measure of distance from each other, we can select centers that are nearest to being orthogonal to each other as possible. Using Euclidean distance to pick furthest

centers often selects points that are outliers, as far from the true center of a cluster as possible. These poor choices of centers can then result in points on the other side of the actual center of a cluster being misclassified. Maximizing orthogonality between centers better reflects the true distribution of segments, given that we know centers should be approximately orthogonal. This can be achieved by setting the distance measure to

$$d(\mathbf{y}, \mathbf{z}) = -|\mathbf{y} \cdot \mathbf{z}|. \tag{3.80}$$

Hence, when points are orthogonal, their dot product will be zero, the maximum value possible; and this value will decrease as the angle between points diverges from orthogonality. Once a near-orthogonal set of centers is selected, points can be assigned to the nearest center as usual. This tends to slightly improve the results of clustering at no additional computational cost over the standard Hochbaum-Shmoys algorithm.

One problem with this approach is that the selection of centers is highly dependent upon the first choice, which is usually random. In some cases, especially with ambiguous regions, this can lead to poor clusterings. Figure 3.6 demonstrates a simple two-dimensional case where a poor initial center selection can lead to a lopsided segmentation. This problem can be prevented by a good choice of an initial center, which ideally is a point in the middle of a cluster rather than an outlier. To pick an initial point then, we can run one or more iterations of the Yu-Shi non-maximum suppression/rotation algorithm to align data approximately with the canonical axes. The point nearest to any canonical axis can then be used as the first center for orthogonal Hochbaum-Shmoys. The result of such an approach is shown in Figure 3.6.

In our implementation, we run the Yu-Shi algorithm until convergence before selecting centers. This almost always occurs within two or three iterations. We then select as the first center the point with the largest single element, which will be the point closest to an axis. Finally, orthogonal Hochbaum-Shmoys is run to cluster all points.

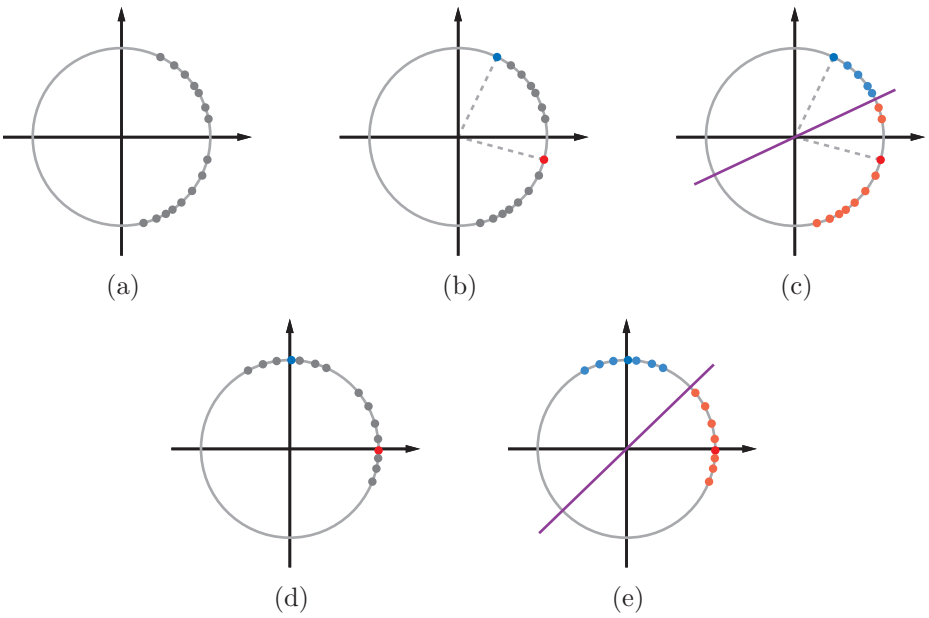



Figure 3.6: A possible problem with random initial center selection for orthogonal Hochbaum-Shmoys. For data with noise lying on the unit circle in (a), a poor center choice is the red point in (b) that lies between the two clusters, which results in the clustering in (c), with mislabeling of two points. By first trying to align data with the axes in (d), this problem is avoided in the clustering (e).

### 3.4.3 Selecting the Number of Clusters

In most previous work on multiclass clustering, it is assumed that the user will set the number of clusters $K$ manually. This, however, would be a tiring interface for animation, where the number of segments should be free to change from frame to frame as necessary without explicit input from a user. Zelnik-Manor and Perona attempt to determine the number of segments completely automatically in [100] by searching over all reasonable $K$ for the number of clusters that seems to give the best correspondence of points on the hypersphere to canonical axes after an aligning rotation. This method gives good results when the clusters are clearly separated, but in ambiguous cases that are common in images, it will tend to select very few clusters. Furthermore, it does not give a user any influence over $K$ should the automatic selection be unsatisfactory.

The $K$-way normalized association value cannot be used directly to choose an optimal $K$ because this value tends to decrease as $K$ increases. In fact, the continuous global optimum monotonically decreases as $K$ increases. The discrete binary approximation does not necessarily decrease monotonically, but it rarely increases. To see why this is so, consider that increasing $K$ must either introduce a new segment or make some change to the previous configuration of the segments while adding a new one. In the first case, all that has occurred is a split in an existing segment. This will result in two smaller segments, so the ratio of their self association to their total association will usually be less than that of their parent. This is analogous to the fact that in the continuous case of a polygon, area decreases quadratically while perimeter decreases linearly when the shape's size is reduced. Hence, the average value of $\mathsf{linkratio}(V_l)$ for a segmentation will tend to be reduced when the number of segments is increased. In the second case, where a segment is added to a different set of initial $K$ segments, the average link ratio of these $K$ segments can be no better than an optimal partitioning. Splitting a segment here will tend to reduce this value further for the same reason as in the other case. This results in a $K$-way normalized association value for $K+1$ no better than that for $K$. Therefore, the $K$-way normalized association value is not a reliable indicator for an optimal selection of $K$. While we could select $K$ as that which maximizes $\mathsf{knassoc}(\Gamma) + f(K)$ for some function $f$ that penalizes small $K$, this lacks a direct relationship to the actual quality of the segments in a given segmentation. In experiments, setting $f(K) = -\alpha K$ for $\alpha > 0$ gave unpredictable results in animation. The behavior of the $K$-way normalized association is not consistent for different types of scenes, other than that it decreases fairly regularly (see Figure 3.15 for the graph of this value on two different scenes).

Instead, we use a measure of suitability for each segment produced by a multiclass partitioning and select the smallest $K$ for which all segments meet the suitability criteria. A segment is considered suitable if all of its nodes are sufficiently similar. We already have a good way of measuring this, by applying two-way normalized cuts. If an optimal

normalized cut value within a segment is below the threshold set by the user, the segment is considered too dissimilar with itself, and $K$ can be increased. This approach is an improvement over setting $K$ directly because it allows the number of segments to increase and decrease naturally in frames of an animation without user intervention. However, it also gives the user some control over the general aggressiveness of the segmentation. There is an additional benefit in that this allows us to directly compare greedy normalized cuts with multiclass normalized cuts using the same threshold value.

We can avoid searching over a large number of candidate values for $K$ by choosing a reasonable initial guess for $K$. Let $\kappa$ be the initial guess for $K$, where $\hat{\kappa}$ is the number of segments to be found. If $K = \kappa$ results in no segments that may be reasonably subdivided, then $K$ is reduced until at least one segment can be cut with the user-set normalized cut threshold, and $K = \hat{\kappa} - 1$. It is generally safe to assume that further reducing $K$ will not result in an improved segmentation, although in some cases this might not be true. One could search for smaller $K$ to ensure that a single poor segment was not just the result of a bad discretization for one value of $K$. Otherwise, if setting $K = \kappa$ results in some segments that can be further subdivided, we increase $K$ until this is not the case, and this gives us $\hat{\kappa}$. In animation, we have a good initial value for $K$, i.e. its final value in the previous frame. Even if $\hat{\kappa}$ is not found immediately, the computational costs of searching are not too high, since the eigenvectors for the full condensed graph only need to be calculated once.

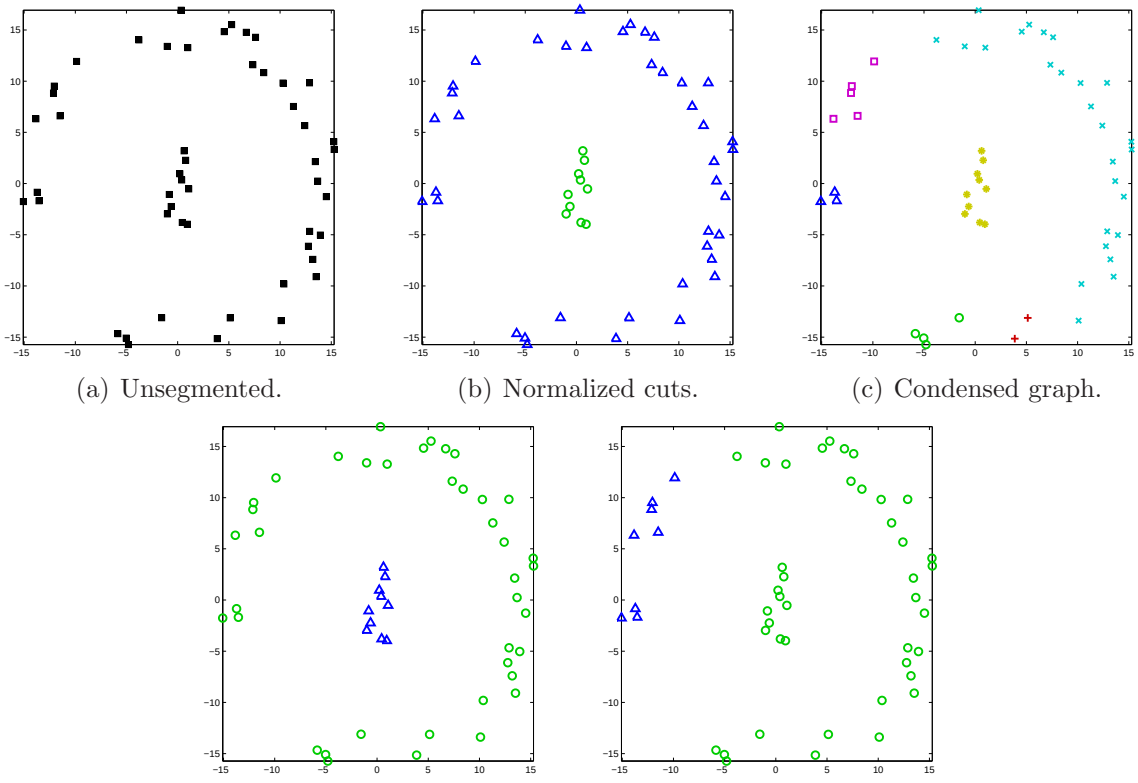
## 3.5 Results

We now consider several examples of normalized cuts using the algorithms described in this chapter on randomly generated point data and a number of simple 3D scenes.

### 3.5.1 Experiments on Point Data

To demonstrate that condensed normalized cuts give reasonable segmentations, we first examine experiments on point data. This makes it easy to visualize affinity between points, since the affinity between nodes is a function of the distance between points. Consider Figure 3.7(a), where there are 50 points arranged in roughly a circle enclosing a set of points at its center. Data laid out in this fashion is notoriously difficult to segment using models such as Gaussian mixture models, since both clusters share the same approximate mean. Normalized cuts, on the other hand, performs quite well on such distributions of points.

The affinity between points here is set to $\exp(-\delta)$, where $\delta$ is the Euclidean distance between points. In this case, normalized cuts produces the segmentation one would expect, with one segment for the points in the center and another for the points lying roughly along a circle (Figure 3.7(b)). The original 50 points are reduced to 6 condensed nodes with a fairly aggressive condensing threshold of 0.01. Figure 3.7(c) shows the groups of nodes that are collapsed into condensed nodes in the same shape and color. Notice that any two points that should be in different segments are assigned to different condensed nodes. This is important, as it allows the condensed normalized cuts algorithm to separate the points into different segments. Only nodes that obviously should belong to the same segment are collapsed into a condensed node. The condensed normalized cut algorithm gives the same partitioning as the full normalized cut algorithm (Figure 3.7(d)), with an optimum normalized cut value of 0.000132. The only difference in this particular case is that the segments are inverted, but this is an equivalent solution. In contrast, the naïve condensed algorithm (without internal association) fails to capture the proper segmentation on the 6 nodes, and its optimal value is 0.000659. As shown in Figure 3.7(e), a set of points of the circle slightly too far from the rest of the points in the circle ends up segmented out, and the center cluster of points ends up in the same segment as the rest of the circle.



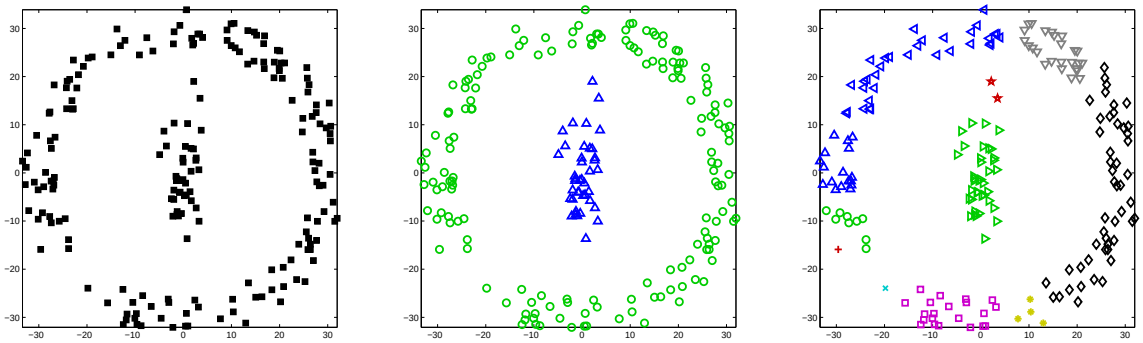
(a) Unsegmented. (b) Normalized cuts. (c) Condensed graph.

(d) Condensed normalized cuts. (e) Naïve condensed normalized cuts.

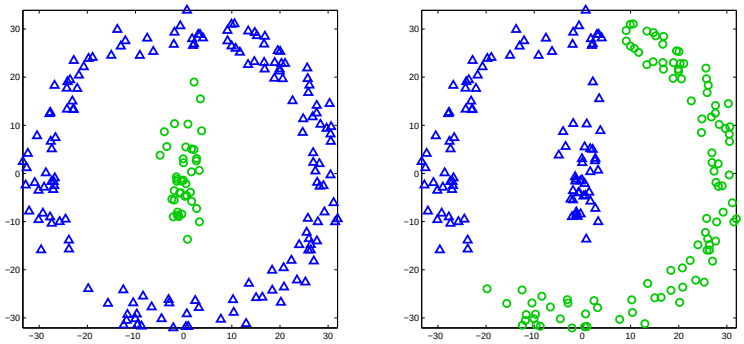
Figure 3.7: Segmentation of a small set of point data.

Another advantage of segmenting a condensed graph can be seen in this example. Typically, edge values below a certain threshold must be set to zero to obtain a sparse enough matrix to make eigenvector decomposition feasible. With a reasonably condensed graph, there is no need to threshold many small edge weights to zero; completely dense graphs can be segmented quickly.

In Figure 3.8(a) points are distributed the same way, but at a larger scale, with 200 nodes rather than 50. Normalized cuts again gives the intuitive partitioning of points, in Figure 3.8(b). Using the same condensing threshold value of 0.01, the graph is reduced to 11 condensed nodes, a nearly 95% reduction in the number of nodes (Figure 3.8(c)). Again, the condensed normalized cuts approach gives the same segmentation as the algorithm on the full graph (Figure 3.8(d)), with a normalized cut value 0.000047 in



(a) Unsegmented. (b) Normalized cuts. (c) Condensed graph.

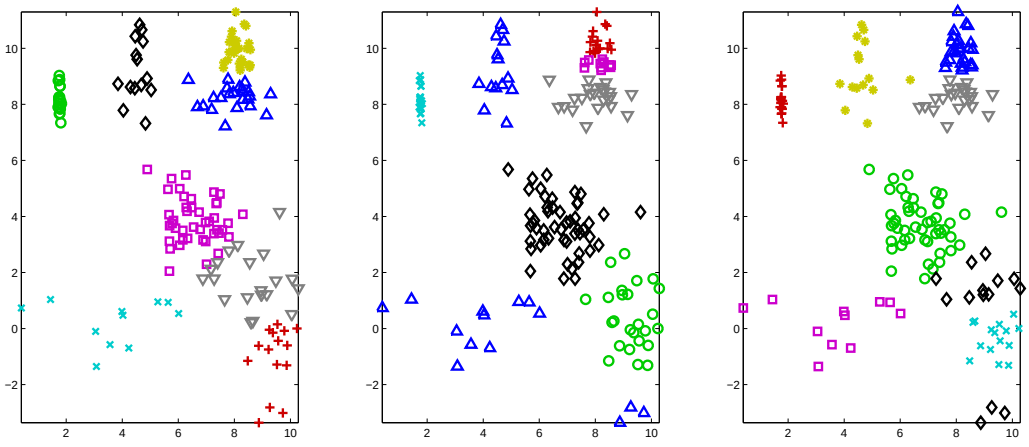


(d) Condensed normalized cuts. (e) Naïve condensed normalized cuts.

Figure 3.8: Segmentation of a large set of point data.

both cases. Without keeping track of internal association within condensed nodes, the segmentation result can be quite poor, as in Figure 3.8(e), where the optimal normalized cut value is 0.001121.

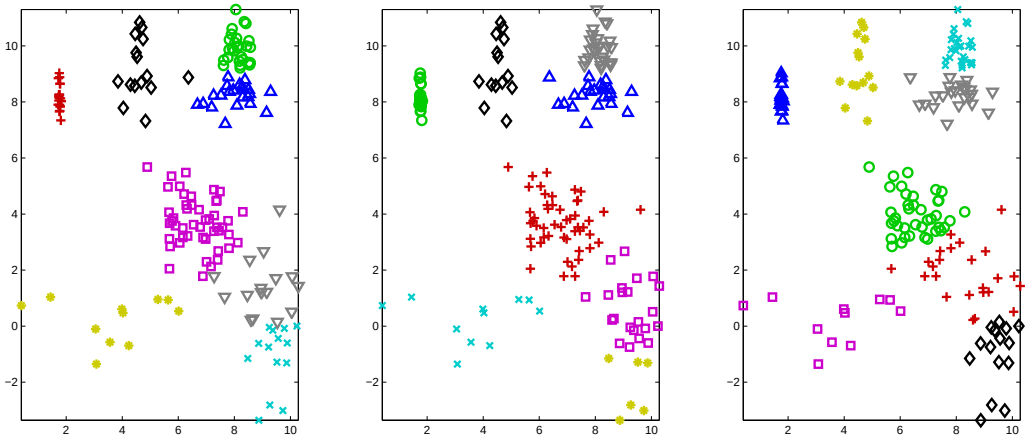
Finally, we consider a set of data points clustered by various multiclass approaches. Figure 3.9 shows a point data set that was generated by randomly selecting eight means and variances in $x$ and $y$ for normal distributions, each with a random number of points. The points for each distribution were allowed to overlap so that a single segmentation is not clear, as the different multiclass methods tend to perform very similarly on obvious clusterings. In fact, the "correct" segmentation in this case has a low $K$-way normalized association value, due to the overlapping distributions. Six different methods for clustering the points on the hypersphere are run on this data set, in all cases $K$ is explicitly set to 8. Since this case does not lead to a degenerate result for the Yu-Shi non-maximum



(a) Non-maximum suppression/rotation.

(b) $K$-means.

(c) Gradient descent.



(d) Hochbaum-Shmoys.

(e) Orthogonal Hochbaum-Shmoys.

(f) Rotated orthogonal Hochbaum-Shmoys.

Figure 3.9: Multiclass spectral segmentation of ambiguously clustered data.

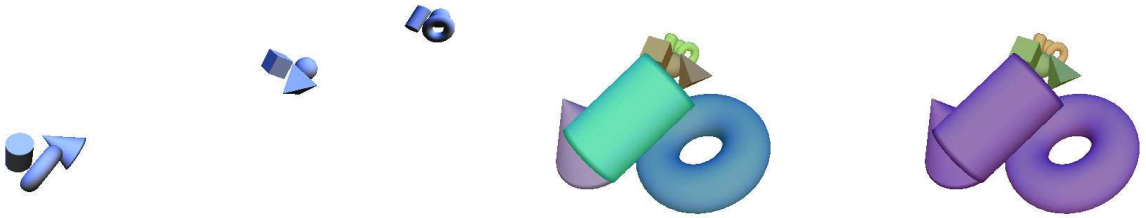
suppression/rotation algorithm, it performs well, with $\mathsf{knassoc} = 0.6737$ (Figure 3.9(a)). The low objective value indicates the poor separation of the data; in better cases it is much closer to 1. $K$-means in Figure 3.9(b) performs quite poorly, with three groups of distant points being segmented together. Perhaps worse, it only assigns points to seven of the eight segments, so its objective value is meaningless. The gradient descent algorithm almost succeeds, but one segment is cut in two by another segment (Figure 3.9(c)). This failure leads the objective value to drop to 0.6707. While this value is not far from that of the Yu-Shi algorithm, perceptually it is clearly much worse. Furthermore, the running

time for gradient descent in this case is around 3 seconds, whereas the other methods tend to take from 0.2 to 0.5 seconds, with $K$-means taking the longest of them. Such performance would be problematic for interactive segmentation. The second row of plots are all variations on Hochbaum-Shmoys in spectral clustering. Figure 3.9(d) uses the Euclidean measure of distance to select cluster centers with the first center randomly selected, resulting in knassoc $= 0.6703$. It is surprising that this is a numerically worse score than that of gradient descent, when the results appear much more reasonable. While the misclassifications are not as obvious in this case, there are a small number of points that have a large effect on the $K$-way normalized association value. The use of negative dot product as the distance measure for selecting centers greatly improves the numerical performance of Hochbaum-Shmoys for this data (Figure 3.9(e)). Even though we use the same initial center, the objective value for orthogonal Hochbaum-Shmoys is 0.6721, since a few of the outliers are reclassified into more appropriate segments. Finally, rotating the space by the Yu-Shi algorithm to select the first center as the point closest to any axis give the results in Figure 3.9(f). This results in a slightly more uniform distribution of points to segments in ambiguous regions, which results in knassoc $= 0.6740$, a value even better than that of the Yu-Shi algorithm. While this will not always be the case, the two algorithms perform very similarly in most cases, with the rotated orthogonal Hochbaum-Shmoys having the advantage of guaranteeing that at least some points are assigned to all segments.

### 3.5.2 Experiments on 3D Scenes

Now we examine results of condensed normalized cuts clustering on a number of 3D scenes.

Figure 3.10 shows a simple geometric scene, made up of three groups of three objects clustered closely together, with the groups lying roughly along a line in space. In Figure 3.10(a), a view showing the separation between the groups of objects is given. This



(a) The scene viewed from the side. (b) One plausible segmentation. (c) Another segmentation with larger depth bias.

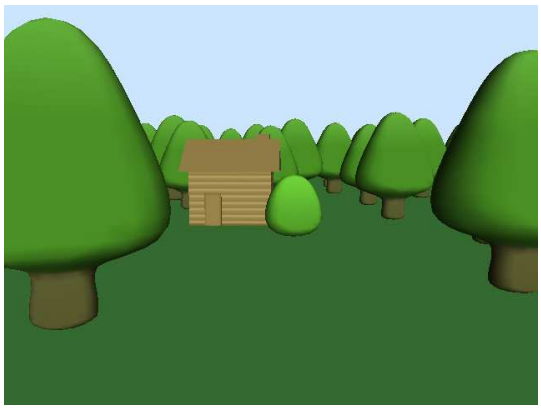
Figure 3.10: A segmentation of a simple geometric scene, emphasizing segmentation by depth.

scene is rendered viewing in the direction of the line that the groups of objects lie on to demonstrate segmentation using only depth information. In the first segmentation (Figure 3.10(b), segments indicated by color), the depth weight ($w_z$) is 40, and all other user-assigned weights are zero. The condensing threshold is set to approximately 2/3, at 0.67, and the normalized cut threshold is set to 0.05. Objects in the foreground are each given their own segment, while the two groups of objects in the background are become exactly two clusters. This is a result of representing depth as $1/(z + \beta)$, which tends to separate elements closer to the viewer more than those further away. Another possible segmentation in Figure 3.10(c), where the foreground group of objects are clustered together, as the two groups in the background are. This is due to increasing the value of $\beta$ to 8, which treats all objects like they are further away, so the depths of the foreground elements are made to appear more similar to each other. In both cases, the segmentation is reasonable, partitioning objects as one would expect, even without using the available object ID information. It was a trivial matter to find weights to give the desired segmentations, and they are fairly robust to perturbations. This scene takes about 0.36 seconds to segment on a 1.6 GHz Intel Centrino processor, including generating all reference images, building the graph, and segmenting it. This is a tremendous speed up

over the minutes normalized cuts can take running on a full graph corresponding to an image. The code that performs the segmentation has not been significantly optimized, so it is likely that this performance may be improved with a more careful implementation.

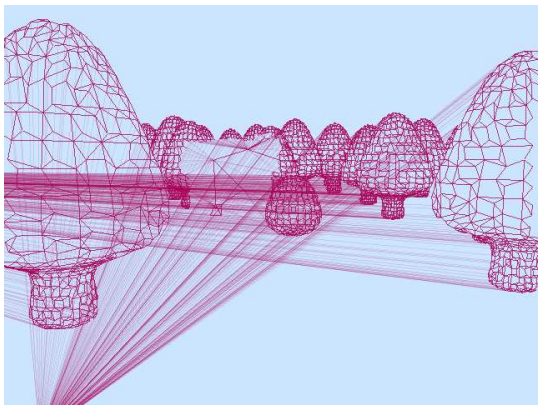
In Figure 3.11, we examine a more natural scene, consisting of a cottage surrounded by trees in a forest and a large bush. In this case, explicit group IDs are assigned to objects—the cottage and door are two groups, the trees and bush are another two groups, and the sky and ground are also groups. The sky is not modeled as part of the mesh, rather, any background pixels automatically take the sky color specified by the user and are treated as being part of a single polygon face of a unique object, with depth set to a constant value. Here, segmentation parameters are set as follows: color weight ($w_c$) is 10, depth weight ($w_z$) is 40, depth bias ($\beta$) is 4, and group weight ($w_g$) is 2. The condensing threshold is set to 0.4, reducing this scene's condensed graph to 30 nodes, and with a normalized cut threshold of 0.02, this is clustered into 15 segments (Figure 3.11(b)). A visualization of the face adjacency graph and condensed graph are given respectively in Figures 3.11(c) and 3.11(d), with lighter edges indicating those that are cut and darker edges indicating those that connect nodes assigned to the same segment. The edges that connect to a node far out of view to the left belong to the ground plane, and those that connect to a node below the field of view belong to the sky. This scene is more complex than the first example, but it still takes only about 0.59 seconds to render and segment, well within the reach of interactive rates.

Figure 3.11(e) demonstrates the importance of group ID information by setting $w_g = 0$. The ground segment then swallows up nearly all of the trees, due to the similar shades of green and little difference in depths. A determined user can find a segmentation close to what is easy to produce with group IDs by tweaking weights and thresholds, but it is an entirely unrewarding experience. Figure 3.11(f) shows results after doubling $w_c$, lowering the condensing threshold to 0.35, and raising the normalized cut threshold to 0.05. Reducing the condensing threshold is necessary because the condensed graph

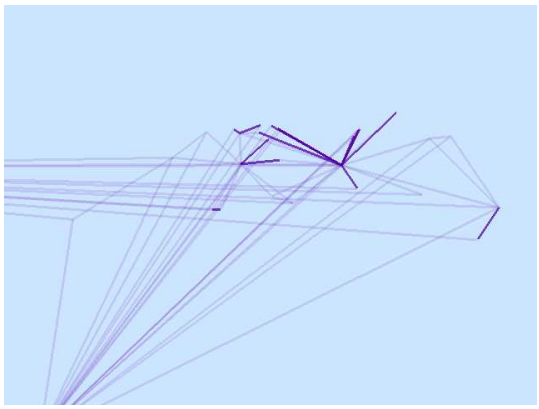


(a) The rendered image to be segmented.

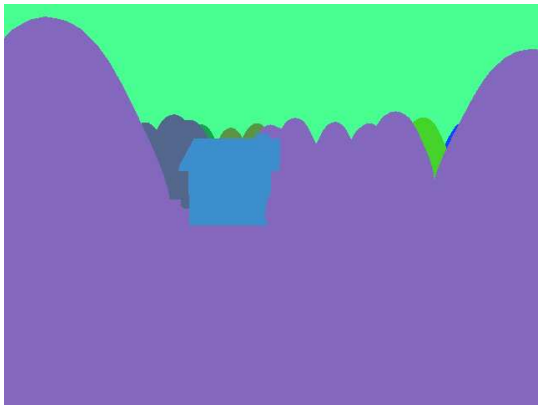
(b) A segmentation of the scene.



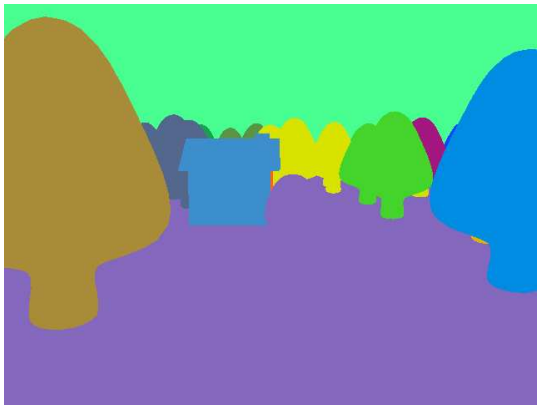
(c) The face adjacency graph.



(d) The condensed graph.



(e) A segmentation without group information.



(f) A segmentation without group information, but with other changes to parameters (see text).

Figure 3.11: A segmentation of a scene made up of a cottage and trees surrounding it. A particular segmentation is achieved by modifying edge weights by assigning objects to groups.

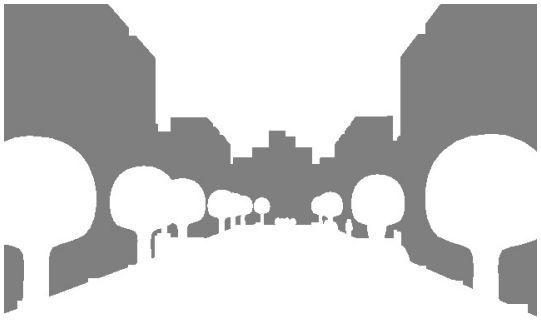
becomes quite large with the particular setting of weights used, however reducing it much more than 0.35 results in some inappropriate collapsing of edges. Even so, the condensed graph is still much larger than it is with $w_g = 2$, with 85 nodes rather than 30. The normalized cut threshold must also be raised to allow more cuts to be made which would otherwise be over the threshold. Even so, we lose the door in the cottage's segment, and the bush is combined with the ground—both due to similarity in color and depth. Without giving users an explicit way to tag objects into logical groups, such cases can be difficult to avoid.

Next, we consider an example utilizing importance assignments in Figure 3.12. The scene is modeled as a dense urban setting, with pedestrians, trees, and tall buildings with several floors of windows (Figure 3.12(a)). Here, the plant life all belong to a single group, while buildings, windows, pedestrians' bodies, and pedestrians' spherical heads are four separate groups, in addition to two groups for the sky and ground. With $w_c = w_o = w_g = 1$, $w_z = 60$, condensing threshold 0.4, and normalized cut threshold 0.033, we find the segmentation in Figure 3.12(b). This is an oversegmentation of the scene, with far too many windows and buildings partitioned out in the background, cluttering the segmentation image. This may be fixed by assigning lower importance to the buildings and windows. Figure 3.12(c) shows the result of using a constant importance map of 0.5 for the buildings and windows, with importance equal to 1.0 everywhere else, and setting $w_s$, the weight for importance, to 1.0. This produces a much cleaner segmentation of elements further from the viewer, without disrupting the segmentation of other objects in the scene—the trees and pedestrian are unaffected by the changes. It could take significantly more time to achieve a similar result using only the other available parameters. Lowering importance of objects is an easy, direct way to get objects to be segmented together. This makes it a good complement to the group parameter, since raising $w_g$ increases separation between groups and raising $w_s$ increases cohesion between unimportant objects.

(a) The rendered image to be segmented.

(b) A segmentation of the scene.

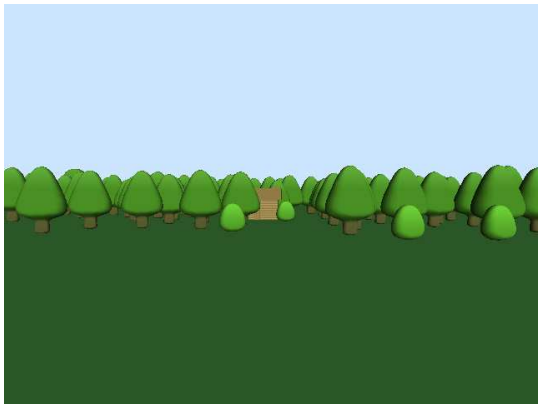
(c) A segmentation of the scene using importance to remove unnecessary detail.



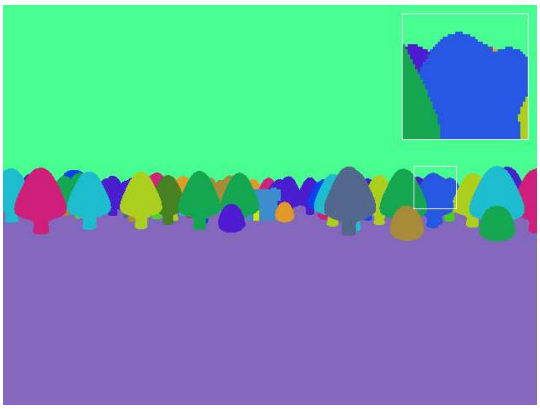
(d) The importance image.

(e) The depth image.

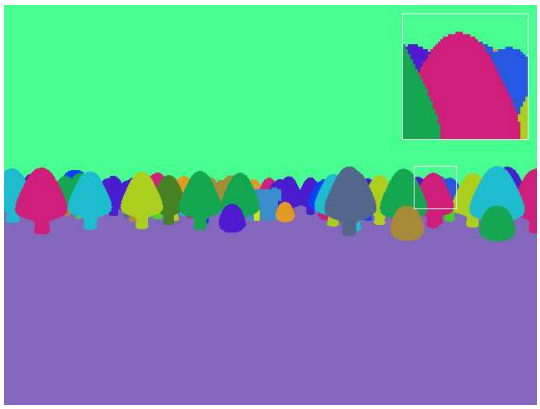
Figure 3.12: A segmentation of an urban scene. Excessive detail can be removed by assigning objects to be unimportant so they will be segmented together.



(a) The rendered image to be segmented.



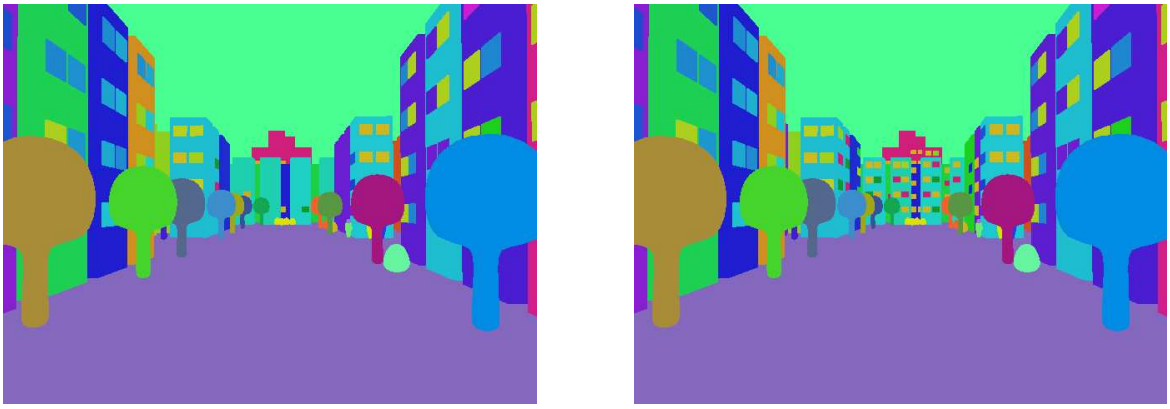
(b) Greedy normalized cuts.



(c) Multiclass normalized cuts.

Figure 3.13: Greedy and multiclass segmentations of a forest scene using the same parameters.

Finally, we compare greedy normalized cuts, using the recursive two-way algorithm, and multiclass normalized cuts, using rotated orthogonal Hochbaum-Shmoys. Figure 3.13 shows the cottage scene with the view pulled back to show more of the forest. The same segmentation parameters are used except the depth bias is set to zero to allow more segmentation to occur. Otherwise, there is no difference at all in the segmentation produced by the two approaches. Even with this parameter set to encourage more segmentation, we find very little difference in the two partitionings. Condensing results in 108 nodes, and both algorithms produce exactly 53 segments. There is one significant difference, however, which is highlighted in the insets of the figure. Greedy normalized cuts segments out one single pixel from a tree that is distant and much darker than the

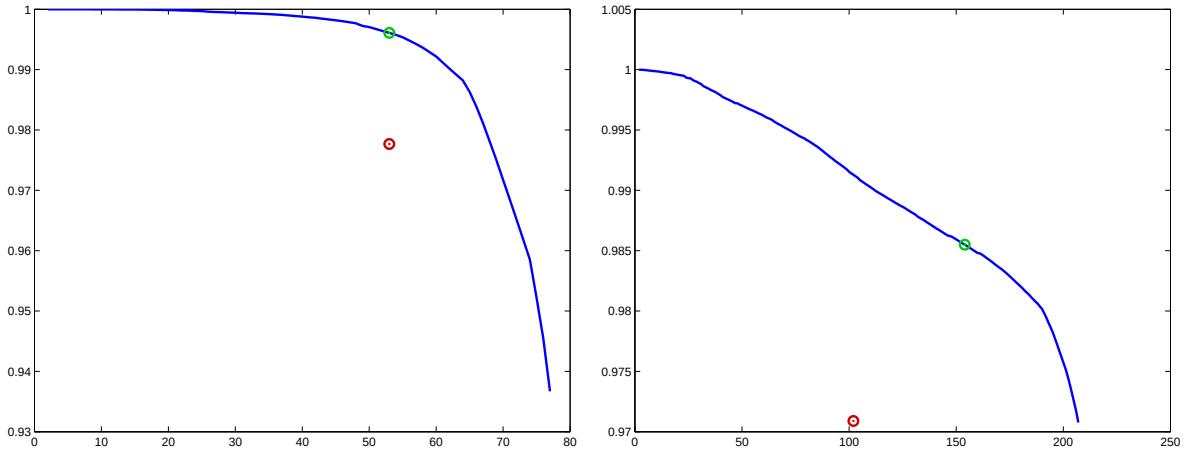


(a) Greedy normalized cuts. (b) Multiclass normalized cuts.

Figure 3.14: Greedy and multiclass segmentations of an urban scene using the same parameters.

surrounding trees, while the multiclass algorithm takes the more reasonable approach of ignoring this outlying pixel and instead cutting a more significant tree from the background trees. This indicates a problem in the greedy approach, where the algorithm can produce graphs to segment in strange ways, whereas the global multiclass method tends to ignore such outliers. The problem is also evident when we examine the $K$-way normalized association values for the two different approaches. For greedy two-way normalized cuts, $\mathsf{knassoc} = 0.978$, but for the multiclass case, $\mathsf{knassoc} = 0.996$. On our test computer, the multiclass approach takes about 0.75 seconds and the greedy approach takes about 0.8 seconds to produce this segmentation.

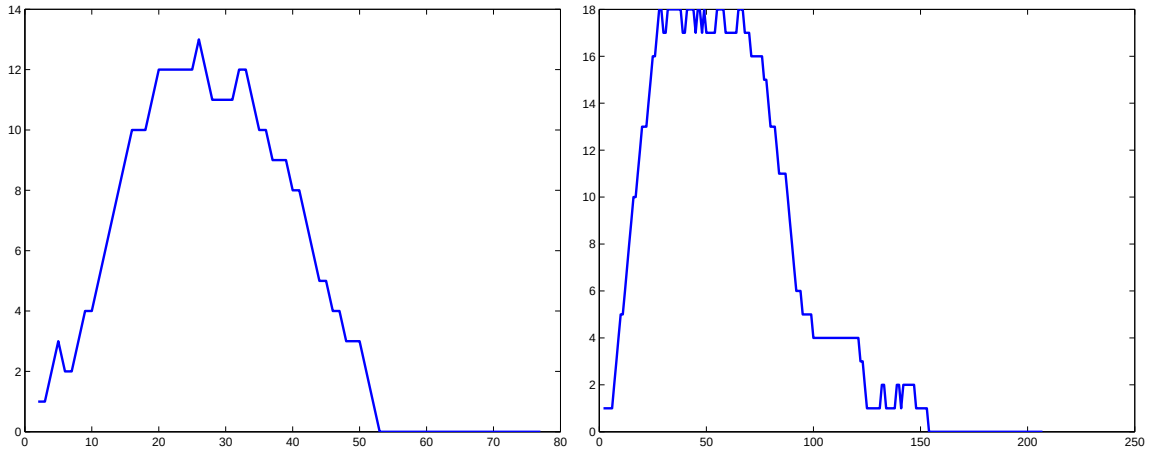
In another example (Figure 3.14), we revisit the urban scene. The parameters are set to the same values as before, except importance is not used. The normalized cut threshold is set lower to 0.02, to reduce the amount of segmentation slightly; the results of this change in threshold can be compared with Figure 3.12(b). It is interesting that the multiclass result for this setting of parameters is closer to the greedy result with the higher normalized cut threshold than it is to the greedy result with the same threshold. The number of segments given by the multiclass approach tends to be the same as or higher than that of the greedy approach. In this case, 209 condensed nodes are reduced




(a) Forest scene. (b) Urban scene.

Figure 3.15: Graphs of knassoc for different values of $K$. The blue line is the objective value for the solution given by multiclass normalized cuts for a fixed $K$, the red circle is the result for greedy two-way recursive normalized cuts, and the green circle is the solution selected for multiclass normalized cuts.

to 102 segments for greedy normalized cuts or 154 segments for multiclass normalized cuts. In this case, $K$ is quite different for the two algorithms. This may seem a bit counterintuitive, but $K$ for the multiclass algorithm tends to be greater than or equal to that for the greedy algorithm. This is because greedy normalized cuts will try to decrease the quality cuts within segments, since it attempts to maximize association within them at each step. Multiclass normalized cuts, on the other hand, will find a near globally optimal set of cuts, and it will raise $K$ as long as any one segment can be divided again. We cannot compare the $K$-way normalized association value for different values of $K$, since is decreases as $K$ increases, but even so, this metric is higher for the multiclass algorithm than the greedy version, with knassoc $= 0.985$ rather than the recursive algorithm's knassoc $= 0.971$. If we graph the $K$-way normalized association value for different values of $K$, it is clear that there is a better segmentation for $K = 102$ that the greedy algorithm could not find (Figure 3.15(b)). Despite the higher number of segments, the multiclass algorithm runs significantly faster on this scene, taking 1.4




(a) Forest scene. (b) Urban scene.

Figure 3.16: The number of segments that can be subdivided with the normalized cut threshold (vertical axis) for a given number of segments, $K$ (horizontal axis), in multiclass normalized cuts.

seconds as opposed to 2.1 seconds for the greedy algorithm. For a single frame, we simply set the initial number of segments to half the number of condensed nodes and search from there.

Even without a good estimate of the initial number of segments, the multiclass method typically performs faster than the greedy recursive algorithm. This is because eigenvectors for the graph need only be computed once for the multiclass algorithm. Then the appropriate number of precomputed eigenvectors can be selected for each value of $K$. With greedy normalized cuts, eigenvectors need to be computed for every cut, since the eigenvectors cannot be reused for the subgraphs separated by a cut. However, in the multiclass case, we still must compute eigenvectors for at least some of the partitions generated to determine their suitability. This test to determine segment suitability limits the speed of this approach somewhat. Even so, when animating several frames, the knowledge of the previous frame's number of segments will usually boost the speed of the multiclass algorithm. The greedy algorithm, on the other hand, cannot benefit from such information from the previous frame. As there is cost associated with clustering the

rows of the $K$-column eigenvector matrix and checking the quality of the segmentation, an improved guess for $K$ can only improve the performance of multiclass clustering.

## 3.6 Summary

This chapter has presented a fast approach to normalized cuts based on the assumption that very strong edges in a graph often should not be cut by a segmentation algorithm. Condensing graphs by orders of magnitude allows the normalized cuts algorithm as well as its multiclass counterpart to be run without changing the original objective function, only cut constraints are added to the original graph. A method of setting edge weights that gives a user influence over the resulting partitioning for scenes rendered from 3D meshes allows a user to guide the segmentation if it is not performing exactly as desired. In experiments, the condensed segmentation algorithm is shown to work reasonably well, segmenting out objects even without explicit information about which pixels belong to which object and producing partitionings of the image plane that reflect something meaningful about the scene. With a solid segmentation algorithm developed, we may now consider how artistic rendering styles can be enhanced with segmentation information.

Chapter 4

# Animation

*"Animation can explain whatever the mind of man can conceive. This facility makes it the most versatile and explicit means of communication yet devised for quick mass appreciation."*

– Walt Disney

---

While the segmentation techniques presented in the previous chapter can be applied to individual frames independently to produce animation, the results will typically be rather poor. Segmentation across frames in all but the most simple cases will exhibit unstable behavior that results in flickering of segments. In this chapter, we consider an approach to reducing instabilities in segmentation across frames for interactive animation.

## 4.1 Temporal Coherence in Segmentation

When rendering an animation sequence, there is no guarantee that consecutive frames will yield consistent segmentations. This is particularly apparent for images in which there are multiple ways to segment a region that have nearly optimal normalized cut values, since slight changes in the image may lead to very different optimal segmentations. This often occurs when the number of pixels on the boundaries between two possible segments increases or decreases slightly, as this can have a significant impact on the weight of outgoing edges relative to the internal edge weights for a small region.

Other works have tackled this problem by segmenting a volume of video all at once [20, 91], so similarity within a region is just as important between frames as it is within a frame. This is not an option here because we would like to produce scenes that can be

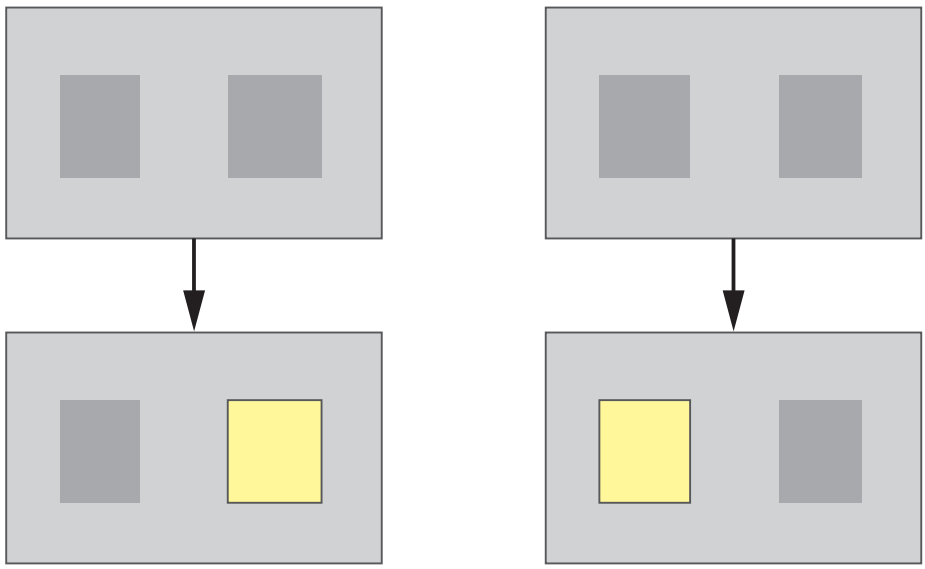


Figure 4.1: An example of an unstable segmentation. A small change in camera orientation can cause the number of pixels in the two darker boxes to change, affecting which one has a larger association. Even though the visual difference in these two images is very small, the segmentation is quite different.

rendered at nearly interactive rates, and in the case of interactive environments, there is no way to account for what changes in the scene might occur in future frames. We instead approach this problem by attempting to enforce coherency between two adjacent frames only.

## 4.2 Coherency for Real-Time Graph Segmentation

### 4.2.1 Coherency Nodes

In order to get coherence in segmentation between frames, we introduce *coherency nodes* to the condensed graph. Each coherency node corresponds to a segment from the previous frame (Figure 4.2). Coherency nodes can be motivated as follows. Suppose we were segmenting an entire video sequence at once. In addition to constructing graphs within

each frame, graph edges would be introduced between nodes in adjacent frames that correspond to the same polygon, with a user-specified weight $w_k$. Since we only segment one frame at a time and have no knowledge of the future, graph nodes for all future frames must be discarded. Moreover, since the segmentation for the previous frame is already known, we can collapse the graph for the previous frame into a single node per segment—called a "coherency node"—while discarding older frames. We can then apply the principles of condensed graph segmentation by keeping track of internal association for the coherency nodes to get a segmentation problem equivalent to segmenting two frames together under the constraint that the previously selected partitionings cannot be modified.

In practice, this graph is implemented as follows. Suppose we have a segment in the previous frame, $t-1$, with an associated coherency node labeled $Q$. Then this coherency node corresponds to the set of nodes within its segment in frame $t-1$, and $Q$ is connected by an edge to each condensed node in frame $t$ that corresponds to a node $Q$ represents. The weight of this edge is $w_k n_i$, where $n_i$ is the number of pixels covered by the face in the current frame. Additionally, $Q$ has a self-edge with weight equal to the association within its segment from frame $t-1$, except edges to coherency node of frame $t-2$ are ignored, if there are any.

Note that, often, some faces of a mesh will not be rendered to the reference images because they occupy less than a pixel, and so they will not be assigned to a segment. This can lead to many faces being considered unsegmented, even when they are surrounded by polygons that were all assigned to the same segment. Hence, in the following frame, if such a polygon becomes visible, it will not get an edge to any coherency node, although it clearly should belong to the same segment as the polygons around it. To address this, assignments of segments are propagated to nearby unsegmented polygons, so that unassigned polygons will be grouped with their neighbors and thus contribute to the coherency edges.

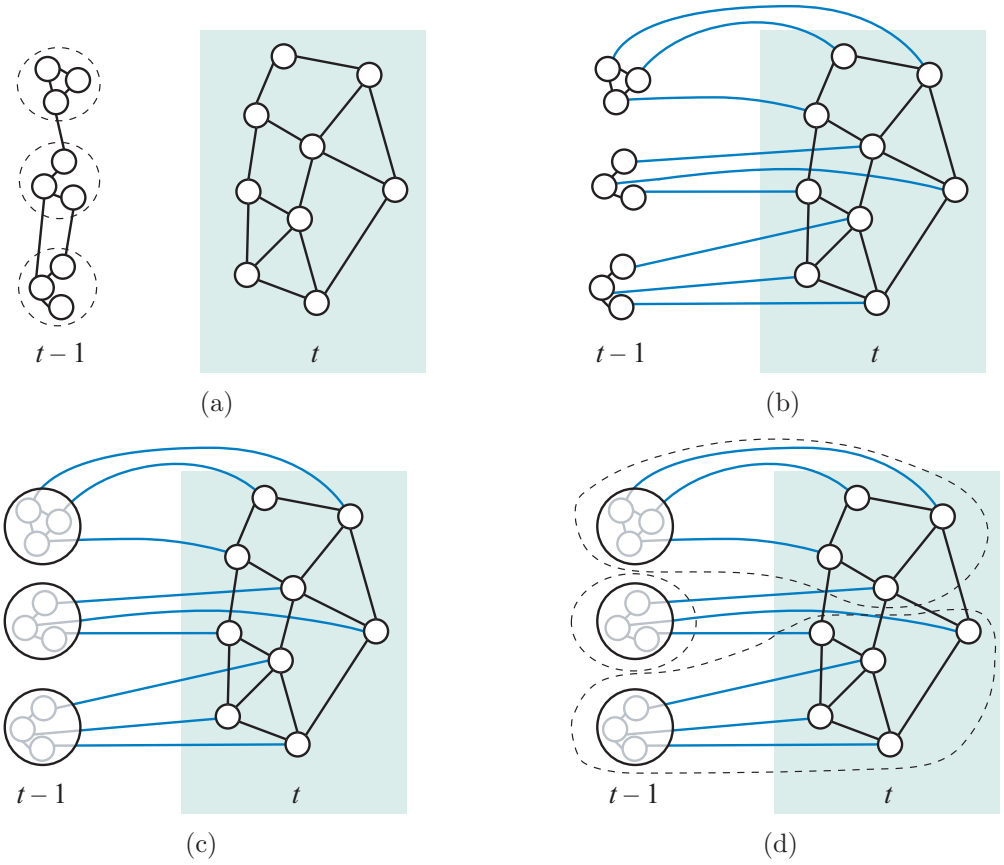



Figure 4.2: Using coherency nodes for temporal coherence. (a) Frame $t-1$ is segmented, and the current frame $t$ has its graph constructed. (b) Nodes in frame $t$ have edges connected to their corresponding nodes in the previous frame. (c) Segments in frame $t-1$ are condensed to coherency nodes. (d) The frames are segmented together.

### 4.2.2 Coherency Bias

Since introducing coherency nodes has the effect of segmenting two adjacent frames together under the constraint that the segmentation from the previous frame cannot be modified, the segmentation for the current frame is biased to be similar to that of the previous frame. A consequence of this is that rendering a single frame alone may appear very different from rendering that frame in an animation with coherency enabled. In cases where in an individual frame there are subtle details that are segmented out by cuts with values close to the normalized cut threshold, these features can be lost in animation due

to the coherency biasing the frame to be like the previous frame, which may not have segmented out the details. Hence, the user might have to find the right balance for $w_k$. Setting it too high can result in very little variation in detail between frames—one might zoom in on objects that were far away, only to find smaller segments never appear or appear only briefly before disappearing off screen. On the other hand, setting $w_k$ too low can result in instability in segmentation between frames, with segments flickering on and off due to slight changes in camera orientation.

This can be viewed as being due to segments not being allowed to appear until they are strong enough to be considered quite stable. One way to address this problem, then, is to lower the normalized cut threshold when selecting segmentation parameters so that only very strong segments appear when viewing a single frame. This threshold can be raised to some acceptable level for animation to get more predictable behavior in frame-to-frame segmentation. More tools for defining a segmentation could also help artists avoid the problem of coherency bias if they can create more separation between graph nodes that belong in different segments. However, the danger in this is that the additional input could just result in making it less clear exactly what is having an effect on the segmentation.

## 4.3 Results

Figure 4.3 shows adjacent frames from an example of an animation rendered twice, once with coherency on, and again with coherency off. The animation consists only of the camera zooming in on a small village, with windows on buildings appearing as the camera gets closer and they take up enough space to warrant their own segment. Without coherency, windows flicker on and off unpredictably; but with coherency, all windows in a segment tend to appear together. The frames for the coherent version of the animation are shown starting 10 frames later than the incoherent version, due to the coherency bias

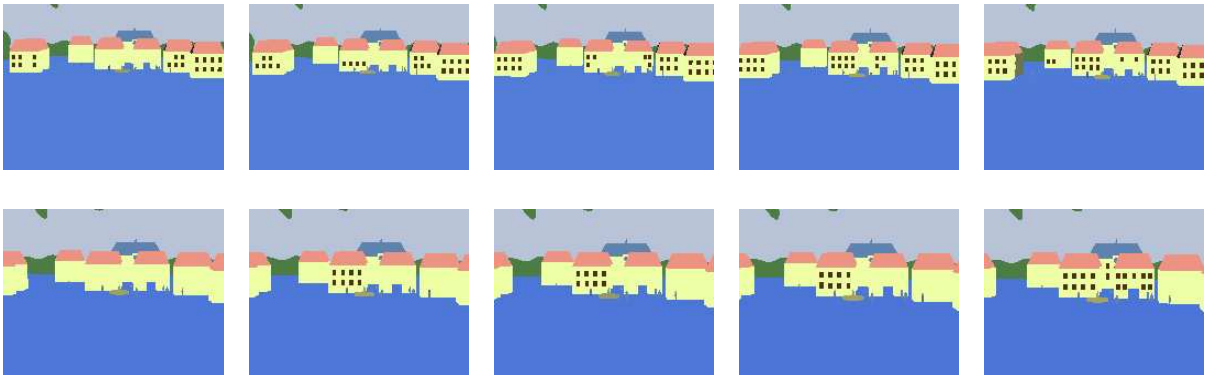


Figure 4.3: A simple scene with average color segment shading as a camera zooms in. Top row: Segmentation without coherency. Bottom row: Segmentation with coherency.

delaying the appearance of the windows.

## 4.4 Summary

We have introduced a strategy of segmenting the current frame with the previous frame using coherency nodes to prevent instabilities in segmentation due to small scene perturbations. This method biases the current frame to be like the previous frame, so an artist must keep this in mind when choosing segmentation parameters, since an image with coherency enabled will often not be segmented exactly the same way in the middle of an animation as it would have been as a single stand-alone image. Next we propose artistic styles for segmentation as well as the considerations for animating artistic styles.

Chapter 5

# Artistic Styles

*"I am enough of an artist to draw freely upon my imagination. Imagination is more important than knowledge. Knowledge is limited. Imagination encircles the world."*

– Albert Einstein

In this chapter, we demonstrate several novel artistic styles based on segmentation. Although some of these rendering techniques are quite simple, together they show that segmentation can be used as a basis for a variety of visual effects that would otherwise be difficult to reproduce. We also consider how artistic effects may need to be modified to work well in animation.

## 5.1 Solid Shading

A basic rendering style is a cartoon rendering style, which shades each segment in a constant color, as employed in many previous works in NPR that have applied constant shading to segments. Figure 5.1 demonstrates examples of this segment coloring, where each segment is rendered with the average color of the pixels inside of it. We also experimented with using median color, but this can cause the color to change unexpectedly with small perturbations of the camera. When a segment contains distinct colors with approximately equal distributions in a segment, the median color often will not be stable throughout an animation. As long as segments are sufficiently uniform in shade and different segments do not make too much use of the same colors, the effect can be suitable for simplifying a picture. Otherwise, if many segments are made up of very similar colors,

Figure 5.1: Solid, average color segment shading.

segment boundaries can be difficult to see, and more complicated styles might be more suitable.

In the next section, we add contours to emphasize shape and segmentation, both of which can be lost with only segment color as a visual guide to the image composition.

## 5.2 Contours

### 5.2.1 Contour Detection

As discussed in Chapter 2, there are several available techniques for rendering contours on polygonal meshes. While there are methods that allow hardware acceleration for very fast contour rendering, these techniques tend to be inflexible [28, 77]. For example, they often only give a single style of contour, with a constant shade and thickness, because we cannot get an explicit chain of contour edges to paint strokes on. More recent hardware contour rendering work has shown more potential [67], but it still is limited to local contour edge detection without chaining. Therefore, we use a software data structure to store contour chains.

We use object space detection of contours, specifically using Hertzmann and Zorin's approach [48], which avoids many of the problems associated with finding contours on

mesh edges between front and back facing polygons. This finds contours through faces rather than on edges by interpolating $\mathbf{v} \cdot \mathbf{n}$ across mesh edges to find its zeros, where $\mathbf{v}$ is the view vector and $\mathbf{n}$ is the normal. The curve along $\mathbf{v} \cdot \mathbf{n} = 0$ gives us the contours of some smooth shape that the triangle mesh approximates. This approach is only applicable for meshes that approximate smooth surfaces, however, so the user is allowed to tag polyhedral objects in a scene which then have contour edges detected in the usual object space fashion of those edges between front and back facing polygons. We assume that such polyhedral meshes will have long enough edges so that chaining is unnecessary. To have face adjacency data quickly available, we use a halfedge data structure for storing all mesh data [7]. This is especially useful for chaining, since contours cross into adjacent faces.

Once the contour chains are found, they must be broken into completely visible or completely hidden chains. One potential change in visibility is where one chain passes under another. These image space intersections can be easily detected in object space. If one segment starts at $\mathbf{a}_1$ and ends at $\mathbf{a}_2$ and the other goes from $\mathbf{b}_1$ to $\mathbf{b}_2$, assuming the viewpoint is at $\mathbf{c}$, we intersect the segment from $\mathbf{a}_1$ to $\mathbf{a}_2$ with the plane formed by $\mathbf{b}_1$, $\mathbf{b}_2$, and $\mathbf{c}$ and check if the intersection point is between $\mathbf{a}_1$ and $\mathbf{a}_2$. If so, we do the same for $\mathbf{b}_1$ and $\mathbf{b}_2$ with the plane formed by $\mathbf{a}_1$, $\mathbf{a}_2$, and $\mathbf{c}$. If that intersection is between $\mathbf{b}_1$ and $\mathbf{b}_2$, we can break the chain at the intersection point furthest from $\mathbf{c}$. Another place where chains must be broken is at cusps, that is, points on the contour chain where the curve's tangent is in the viewing direction. Detecting these points on our piecewise linear contour chains is possible by expressing the surface tangent in terms of its curvature and finding intersections of our contour chains with another set of curves on the mesh. These curves are the zero set of the radial curvature, given by the cusp function $C = \kappa_1(\mathbf{v} \cdot \mathbf{w}_1)^2 + \kappa_2(\mathbf{v} \cdot \mathbf{w}_2)^2$, where $\kappa_1$ and $\kappa_2$ are principle curvatures and $\mathbf{w}_1$ and $\mathbf{w}_2$ are corresponding principle curvature directions. To find these cusps, we require some estimation of curvature at the mesh vertices. For this purpose, we use Rusinkiewicz's

algorithm which estimates curvature at each face and then computes it at the vertices as a weighted average of the adjacent face's curvatures [79].

In practice, we first detect smooth contour chains on objects tagged as smooth. These chains are broken at cusps, and then polyhedral tagged objects have their contours detected and added to the list of chains. All of the chains are clipped against the viewing volume, and the remaining chains have occlusions detected by finding and breaking intersections of contour chains as described above. The image plane is broken up into a grid to prevent excessive checking of contour segments against segments that they cannot possibly intersect. Points along the chains are sampled to determine their visibility, and finally quad strips are mapped to the visible chains, displaying a texture assigned by the user.

### 5.2.2 Segment Boundary Contours

This gives us all the contours in a scene, but it does not address the segmentation in any way. We combine the segmentation information with the contour chains by optionally rendering only the boundary of each segment. This is in contrast to the usual practice of rendering silhouettes at the level of objects in a scene. We accomplish this by making two small changes to the contour rendering pipeline.

First, when searching for contour intersections, if two intersecting contours belong to the same segment, then the contour closer to the viewpoint is also broken. A contour is considered to belong to the same segment as the faces it passes through. Contours can be broken if the segment ID changes between adjacent chain segments, although this is rare in practice. This gives us an additional property on the contour chains—not only are they all entirely visible or hidden, but they also each belong to a unique segment.

Second, one additional step after the chain visibility test is added. A chain is checked to determine whether it is interior to a segment by sampling the segment ID of pixels on either side of the chain. If the segment ID matches at all sampled pixels, the chain

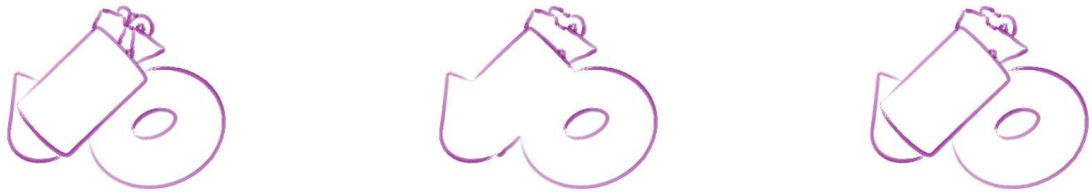


Figure 5.2: Left: All contours rendered in a purple watercolors style on a scene with some simple geometric objects. Center and right: Segment boundary contours with two different segmentations.

is considered interior to that segment; and it can be hidden or tagged to be rendered in a different style. This is one reason for constructing a face adjacency graph, which constrains each face to belong to a unique segment; it greatly simplifies finding interior chains in this fashion.

This pixel sampling method for determining whether a contour is interior to a segment or not does lead to a small number of contours being incorrectly labeled, but the errors are typically not visible, and the effects can be reduced by increasing the image resolution. The planar map algorithm of Winkenbach and Salesin [94] could also be used to compute these contours exactly, although at significant computational cost.

Rendering contours on the boundaries of segments greatly increases the perceptual effect of the segmentation, clearly marking where segment edges are found. Figures 5.2 and 5.3 compare examples of rendering all contours and only contours at segment boundaries. Clutter is reduced in these pictures by only rendering contours at segment boundaries, and the overall segmentation is brought out more clearly than when using color alone. This also produces something of a flattening effect, making the pictures appear to be constructed in a 2D plane rather than rendered from 3D models, which is a pleasing effect for artistic rendering.

Figure 5.3: Left: All contours rendered on segmented scenes. Right: Segment boundary contours rendered on the same scenes.

## 5.3 Watercolor

A simple watercolor style can be created simply by adding Perlin noise [72] to the image, desaturating the average color each segment is rendered in, and darkening edges between segments. Figure 5.4 shows examples on a number of scenes. As simple as the algorithm is to generate these watercolor emulating images, the effect is striking. The noise models the variation paint absorption that is caused by the texture of the paper. The desaturating of the colors makes them appear more muted, as dried watercolors are often not made up of extremely vibrant colors. Edge darkening models an effect in watercolor that occurs due to surface tension in the water that prevents it from spreading at the boundaries when painted over a dried wash [23]. While more complicated algorithms are possible,

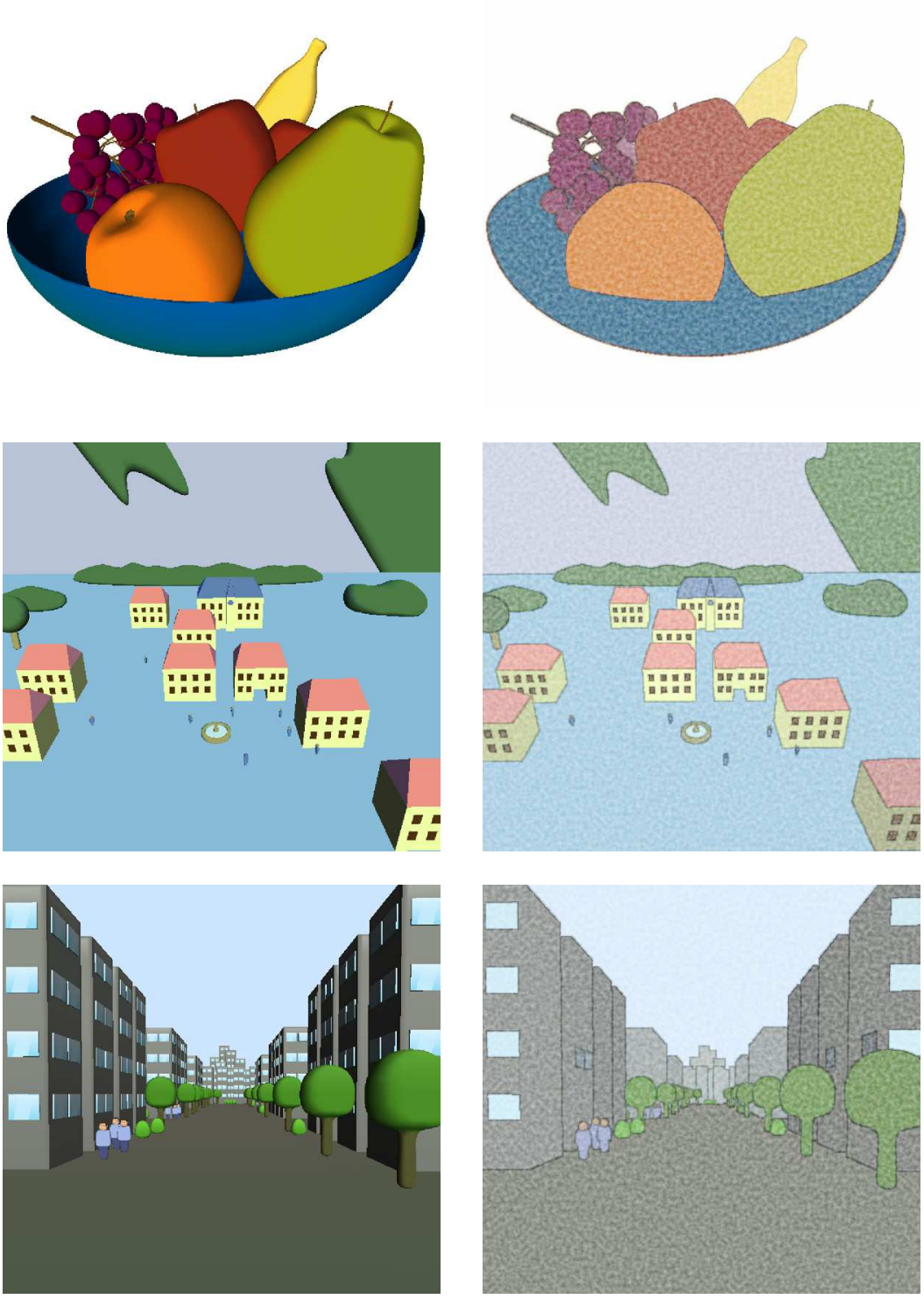


Figure 5.4: Left: Color reference images. Right: Segmented images with a watercolor style.

this approach is a simple extension to solid-shading segment coloring, so it benefits from any enhancements we may make to that algorithm, such as for animation, and it is not computationally demanding. The current implementation takes too long for interactive applications, but with some tuning, it could probably be optimized to run at speeds that are not too prohibitive to live user interaction.

## 5.4 Oil Painting

Painterly rendering styles are also possible within the framework of segmentation. In fact, oil painting simulation is one of the more compelling applications of segmentation in NPR. In this style, each segment is filled with paint strokes in two passes. The first pass attempts to fill each segment with strokes as follows. Very large segments are filled with large horizontal strokes, similar to the background in Figure 1.2. Smaller segments are filled by drawing smaller strokes in horizontal, vertical, or diagonal directions with random variations in direction. These strokes may terminate by reaching the edge of their segment or by reaching a maximum stroke length. The second pass emphasizes segment boundaries, similar to the way strokes in Figure 1.2 carefully follow the segment boundaries that we manually highlighted.

To compute these boundary strokes, a distance field is computed for each segment using Felzenszwalb and Huttenlocher's linear-time algorithm [35]. Let $R$ be the radius of the brush stroke size. A pixel with distance $R$ from the segment boundary is located, and, starting from this pixel, a path is traced in the distance map that maintains this distance $R$ to the boundary, within a small threshold. The stroke ends when it cannot be continued without doubling back on itself or when a maximum length is reached. All strokes are then subsampled by a factor of $R$ and rendered using Hertzmann's relief texturing algorithm, which gives a bump-mapped effect by using a stroke texture to generate a height field which is lit in software [46]. The strokes are rendered against a

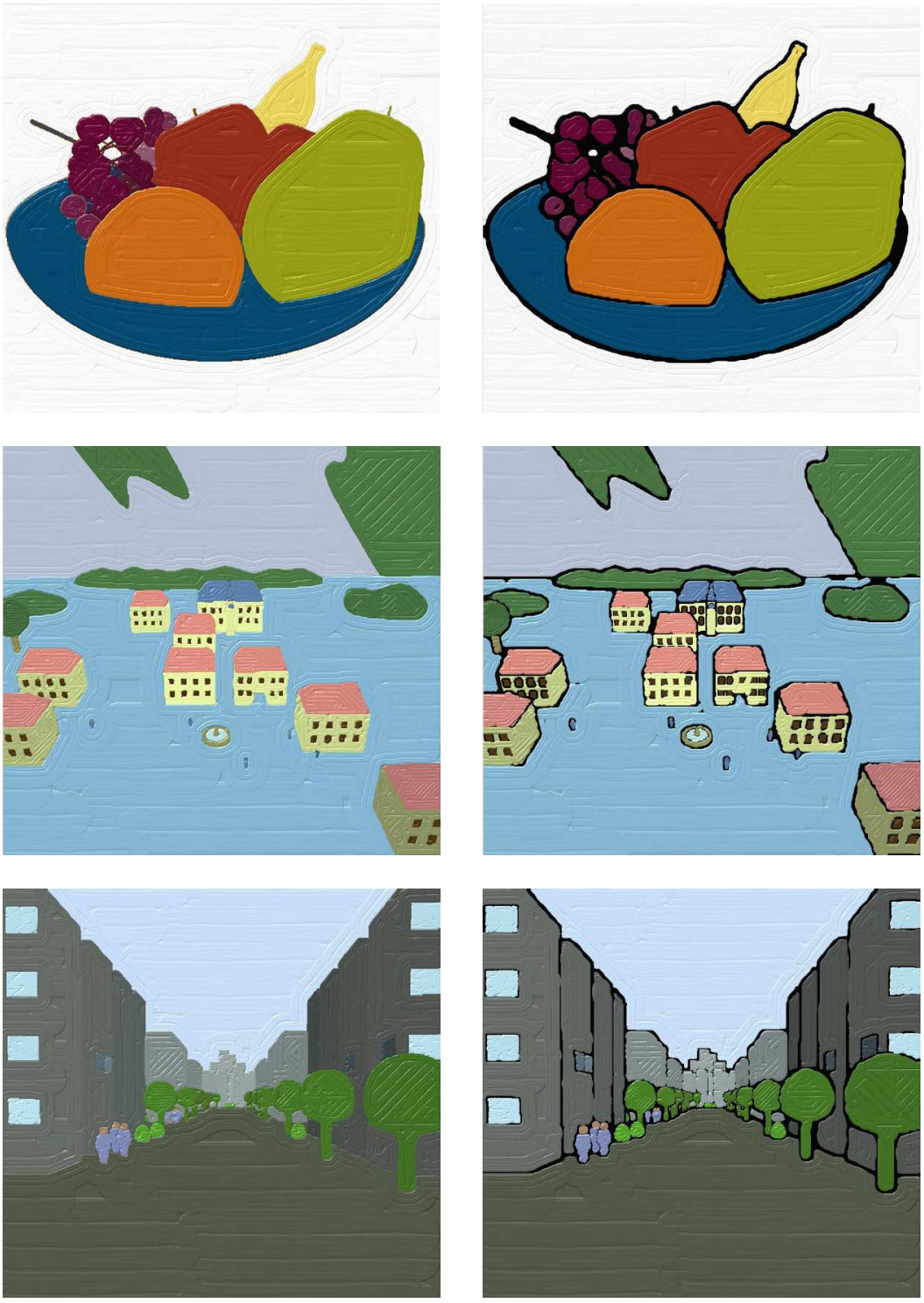


Figure 5.5: Left: Segmented scenes in an oil paint style. Right: The same style with a black background.

colored background, which is determined by the user-selected style. If the color reference image is used as a background, then holes in the strokes are filled with reasonable colors, so that gaps between strokes are not obvious. If a black image is used for a background, then irregular black gaps appear between segments in a manner similar to those in Figure 1.6. These gaps do not simply correspond to thick segment boundary contour renderings, since paint covers the gaps irregularly, giving them the character of being under the paint, rather than over it as contours would be.

## 5.5 Temporal Coherence in Artistic Styles

For a style as simple as constant color shading for each segment with coherent frame-to-frame segmentations, difficulties can still arise in animation. We have selected the average color to shade segments, even though intuitively the median might seem to give a better representation of the color of a segment. After all, a few outliers that are very dark or very bright can throw off the average color, making the entire segment lighter or darker even though only a few pixels might be affected. While this is not a problem with the median color, in experiments, it was found that in many cases there can be a two shades in a segment with nearly the same number of pixels. For example, a log cabin might be a light shade of brown in the direction of a light source above, with about an equal number of dark brown pixels in shadow. Small changes in the camera can result in the median changing from light to dark and back very rapidly. In other words, the median color can be unstable in some cases, but average color tends to change more gradually with perturbations of the camera.

Even using average color, there are situations in which the segment colors can change rapidly in distracting ways. An example of this can be constructed with a segment that is a light color in one area, with the rest a darker shade. Then if the lighter area becomes occluded, say by rotating the camera such that part of the segment goes out of view, this

will cause the segment to suddenly become dark, when before it was a blend of the light and dark parts of the segment. Also, consider the case where a segment bifurcates into two by being split by some new occluding segment. As soon as the bifurcation occurs, unless the two new segments happen to have very similar shading, their color will change immediately. So even in this simplest case of constant segment shading, we find many possible causes of flickering and other undesirable effects.

Rather than sampling segment colors for using the average or median, one could assign each object a single color. Then, if objects are sorted in a list to indicate their priority by the user, the most dominant object in each segment could determine the segment color. This would likely prevent a lot of the problems apparent in shading based on sampled colors, but this comes at the cost of only allowing a small list of colors to be assigned to segments. This would make it difficult to tell where segment boundaries are, since many segments would be exactly the same color. Furthermore, the user must be burdened with sorting objects by priority, when at times no relationship is clear between two objects. Segments would still "pop" as objects change segments, causing segment colors to suddenly change between frames.

One possible modification for reducing these distracting effects is to track which coherency nodes are segmented with each segment in the current frame. Then the colors used in each of these previous segments can be averaged with the color for the current frame, to produce more subtle blending effects in the time domain, to reduce the sometimes harsh all-or-nothing nature of segmentation in animation. This should give a smoothing effect similar to that of [20], which uses video volumes to prevent such jumps in shading.

For our watercolor simulation, since segments are based on a single color, they can benefit from such a color blending system. Otherwise, we create watercolored animation by using the same noise-generating function across frames, to prevent what could appear to be a distracting static. It might also be interesting to attempt to translate and zoom

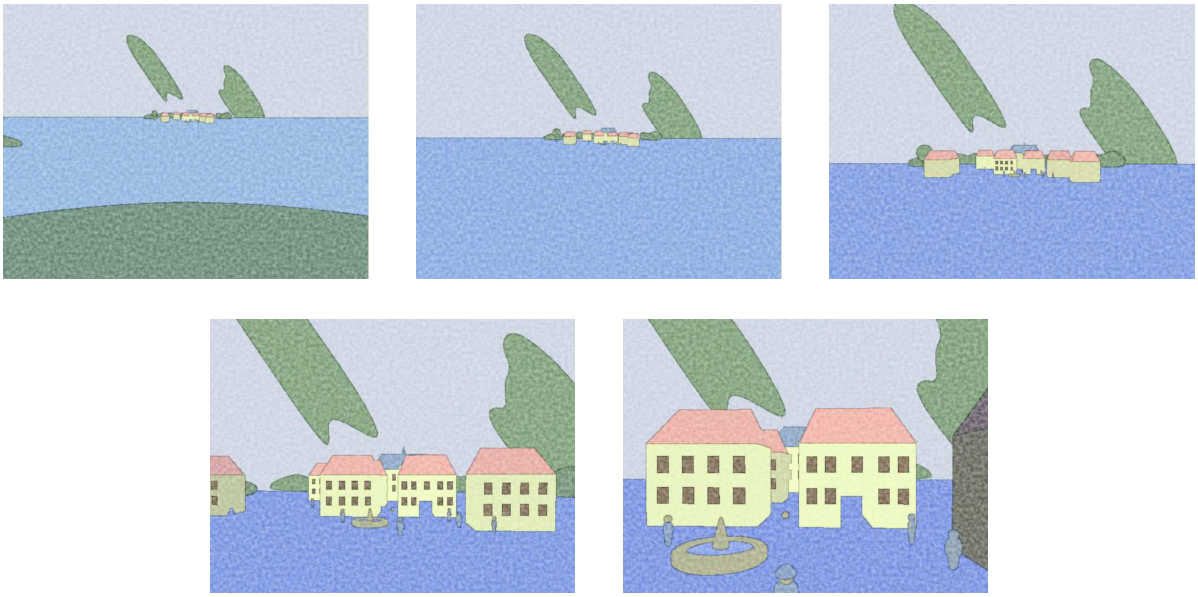


Figure 5.6: Frames of a scene recreated in 3D to model that of Figure 1.6 rendered in a watercolor style with coherency.

the noise function to match the camera movements, but such motions might appear unnatural. It is difficult to know what the "right" way to portray animated watercolors is when this is a medium that is so difficult to animate in reality. Figure 5.6 reimagines the hand-painted animation of Figure 1.6 as a watercolor animation, using segmentation of frames similar to the way the artist did in the original work.

For artistic coherency in stroke-based styles, such as our oil painting style, a more complicated algorithm would be required. There are a few possible approaches beyond simply painting each frame independent of decisions made in previous frames. The work of Hertzmann and Perlin [47], which paints strokes to create painterly representations of video, provides some insights into what works in their experiments. One possibility is to paint over the previous frame, only updating regions that have changed. While this would seem the most natural approach, it is not consistent with our strategy of carefully painting around segment boundaries, and in Hertzmann and Perlin's work, it is noted that this produces an "effect of a continually smudged image plane," which is

not always pleasing. They instead move brush strokes according to optical flow, similar to the approach taken by Litwinowicz on smaller strokes [62]. An analogous approach in the context of our oil painting system would be to attempt to attach boundary strokes to segment boundaries, while moving and resizing interior fill strokes to match the changes in segment shape. Then new strokes would only be added where they are needed.

## 5.6 Summary

This chapter has introduced a number of styles that may be applied to segmented scenes. We have also identified some sources of instabilities in artistic styles that can occur even with an ideally segmented animation, and we have discussed some possible solutions to these problems. While these styles are all fairly simple, together they demonstrate that segmenting the image plane is conducive to a broad range of styles, rather than being tied specifically to one or two techniques. Many of the styles presented in other works, discussed in Chapter 2, may also be applied to segments by treating each segment as a separate image with irregular borders. For example, the physically simulated watercolor model presented by Curtis et al. [23] could be applied to each segment, allowing overlap by sorting segments for rendering based on some average depth criteria to obtain realistic looking layering between segments. This could give much more realistic appearing watercolor renderings, at a cost of significant computational time. Another possibility is using segment boundaries as edges to initialize a rendering algorithm based on the placement of edges between regions, such as Hausner's decorative mosaic algorithm [44].

The research prototype used here only allows one style per scene. An interesting extension might be to allow an artist to assign segments different rendering styles, either based on simple criteria such as depth or segment size, or hand assigned by the user. However, such a system that gives users detailed control over segment properties might be difficult to extend reliably to automatic animation.

Chapter 6

# Conclusion

*"Art, like morality, consists of drawing the line somewhere."*
– G.K. Chesterton

---

This thesis has presented algorithms for creating segmented artistic renderings of 3D scenes. This system produces image and animated representations of scenes that abstract out distant, unimportant, and cluttered scene detail while simultaneously providing a segmentation that can act as a basis for various artistic styles. A graph-based approach to segmentation is used for its flexibility in defining affinity between points over. Either greedy or multiclass normalized cuts may be applied on compact representations of graphs to accelerate the segmentation, approaching interactive rendering rates. These accelerated segmentations often result in no loss of quality in segmentation while reducing the graph size by orders of magnitude. Segmentation takes place in the image plane rather than object space because it corresponds to the types of 2D segmentations that are often seen in art. An unexplored advantage of the conceptual simplicity of this approach is that interesting projection effects can be applied before the rendering and segmentation step with no need for any other modification to the system. Nonlinear projections, such as those described in [18], could be used to produce the reference images to get unique artistic styles.

A variety of simple styles are proposed, some of which would be very difficult to model without an explicit model of segmented image regions. An area for future work is to produce more convincing, high-quality artistic styles that are particularly suited to

this segmented framework. This system has been demonstrated to be suitable for the purposes proposed by testing it on simple 3D scenes. Rendering large scenes without abstraction leads to substantial clutter and detail that we would not expect from artistic imagery. With further work, the approach presented here should enable live interaction with scenes that automatically and adaptively vary the abstraction level to match the scene, as determined in advance by an artist.

An important problem is to perform segment-based rendering in real-time. Some speedup can certainly be gained by a more finely-tuned implementation. For example, the current system attempts to render each object and compute contours on every object in the scene; there is no scene-level culling before the rendering step. This can result in a non-trivial amount of computation being wasted finding contours that will only be thrown away. One possibility is to design 3D culling and level-of-detail algorithms appropriate for segmented scenes. This would be extremely desirable, since one of the main applications this work is targeting for segmentation is in creating artistic virtual worlds. Further, the artistic styles presented here in some cases take quite a while to render, speedups here would be beneficial to frame rates. Since applying an artistic style to a given segmentation can take place in parallel with the segmentation of the next frame, the current trend in parallelizing processing hardware could be extremely beneficial to animation speed. Indeed, a carefully designed rendering style might be able to process each segment independently, enabling the use of several processors to simultaneously work on a single frame.

Another critical problem that is not fully addressed by this work is to create a good interface for designing styles. In the current implementation, a user has numerous sliders to manipulate, corresponding to weights that affect how affinity between points is defined as well as condensing and normalized cut thresholds. It is difficult to imagine such an interface being used in a production environment where precise control over a scene is necessary or in games where no user intervention can be expected. Segments would fit

very naturally into the procedural NPR shaders of Grabli et al. [41] as a fundamental primitive. Rather than using NPR shaders only on contour data structures, it would be interesting to create a shader system that presents segment regions in such a natural way to the designer. A more challenging problem is to design segmentation styles by examples with a WYSIWYG interface [53]. One can imagine an artist being able to guide a segmentation at various keyframes by indicating what points in a scene must belong to different or similar segments without ever seeing the actual parameters used by the segmentation algorithm, similar to current systems that allow a user to quickly indicate a few points that belong to certain regions, while the precise region boundaries are computed automatically [11, 59, 78].

While there are many directions this work could go in the future, the system presented here represents a promising first step in segmentation-based real-time artistic rendering systems. Many styles remain to be explored under this framework, but this is an advantage rather than an incompleteness, as this segmentation approach gives the NPR designer and user such flexibility beyond a single artistic appearance or type of scene. A future challenge that goes far beyond this work is to design higher-level NPR algorithms, capable of making decisions about style based on the semantic content of a scene rather than the low-level relationship between pixels and objects. Such a system would be a big step in producing automatic artistic systems that make decisions about stylistic content similar to those an artist might make.

# Bibliography

[1] Aseem Agarwala. SnakeToonz: A Semi-Automatic Approach to Creating Cel Animation from Video. In *NPAR '02: Proceedings of the 2nd International Symposium on Non-Photorealistic Animation and Rendering*, pages 139–146. ACM Press, 2002.

[2] Aseem Agarwala, Aaron Hertzmann, David H. Salesin, and Steven M. Seitz. Keyframe-Based Tracking for Rotoscoping and Animation. *ACM Transactions on Graphics*, 23(3):584–591, 2004.

[3] Arthur Appel. The Notion of Quantitative Invisibility and the Machine Rendering of Solids. In *Proceedings of the 1967 22nd National Conference*, pages 387–393. ACM Press, 1967.

[4] Arthur Appel, F. James Rohlf, and Arthur J. Stein. The Haloed Line Effect for Hidden Line Elimination. In *Proceedings of the 6th Annual Conference on Computer Graphics and Interactive Techniques*, pages 151–157. ACM Press, 1979.

[5] J. Andrew Bangham, Stuart E. Gibson, and Richard Harvey. The Art of Scale-Space. In *Proceedings of British Machine Vision Conference*, 2003.

[6] Asa Ben-Hur, David Horn, Hava T. Siegelmann, and Vladimir Vapnik. Support Vector Clustering. *Journal of Machine Learning Research*, 2:125–137, 2002.

[7] Mario Botsch, Stephan Steinberg, Stephan Bischoff, and Leif Kobbelt. OpenMesh – A Generic and Efficient Polygon Mesh Data Structure. In *OpenSG Symposium*, 2002.

[8] David Bourguignon, Marie-Paul Cani, and George Drettakis. Drawing for Illustration and Annotation in 3D. In *Computer Graphics Forum (Proceedings of Eurographics 2001)*, volume 20, pages 114–122, September 2001.

[9] Yuri Boykov and Vladimir Kolmogorov. An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 359–374, 2001.

[10] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast Approximate Energy Minimization via Graph Cuts. In *Proceedings of the International Conference on Computer Vision*, volume 1, pages 377–384, 1999.

[11] Yuri Y. Boykov and Marie-Pierre Jolly. Interactive Graph Cuts for Optimal Boundary and Region Segmentation of Objects in N-D Images. In *Proceedings of the International Conference on Computer Vision*, volume 1, pages 105–112. ACM Press, 2001.

[12] John W. Buchanan and Mario C. Sousa. The Edge Buffer: A Data Structure for Easy Silhouette Rendering. In *NPAR '00: Proceedings of the 1st International Symposium on Non-Photorealistic Animation and Rendering*, pages 39–42. ACM Press, 2000.

[13] Chris Buck and Kevin Lima. *Tarzan*, 1999. Disney.

[14] Capcom. Auto Modellista, 2003.

[15] Chakra Chennubhotla and Allan Jepson. Hierarchical Eigensolver for Transition Matrices in Spectral Methods. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 273–280. MIT Press, Cambridge, MA, 2005.

[16] Fan R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

[17] Jonathan M. Cohen, John F. Hughes, and Robert C. Zeleznik. Harold: A World Made of Drawings. In *NPAR '00: Proceedings of the 1st International Symposium on Non-Photorealistic Animation and Rendering*, pages 83–90. ACM Press, 2000.

[18] Patrick Coleman. Interactive control of nonlinear projection for complex animated scenes. Master's thesis, University of Toronto, 2004.

[19] J. P. Collomosse, D. Rowntree, and P. M. Hall. Cartoon-Style Rendering of Motion from Video. In *Proceedings of Video, Vision and Graphics (VVG)*, pages 117–124, July 2003.

[20] J. P. Collomosse, D. Rowntree, and P. M. Hall. Stroke Surfaces: A Spatio-temporal Framework for Temporally Coherent Non-photorealistic Animations. Technical Report 2003–01, University of Bath, U.K., June 2003.

[21] Dorin Comaniciu and Peter Meer. Mean Shift: A Robust Approach Toward Feature Space Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.

[22] I. J. Cox, S. B. Rao, and Y. Zhong. Ratio Regions: A Technique for Image Segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 557–564. IEEE Computer Society, 1996.

[23] Cassidy J. Curtis, Sean E. Anderson, Joshua E. Seims, Kurt W. Fleischer, and David H. Salesin. Computer-Generated Watercolor. In *SIGGRAPH '97: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, pages 421–430, August 1997.

[24] Eric Daniels. Deep Canvas in Disney's Tarzan. In *SIGGRAPH '99: ACM SIGGRAPH 99 Electronic Art and Animation Catalog*, page 124. ACM Press, 1999.

[25] Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. Suggestive Contours for Conveying Shape. *ACM Transactions on Graphics*, 22(3):848–855, 2003.

[26] Doug DeCarlo and Anthony Santella. Stylization and Abstraction of Photographs. In *SIGGRAPH '02: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, pages 769–776, New York, NY, USA, 2002. ACM Press.

[27] Oliver Deussen and Thomas Strothotte. Computer-Generated Pen-and-Ink Illustration of Trees. In *SIGGRAPH '00: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 13–18. ACM Press/Addison-Wesley Publishing Co., 2000.

[28] D. Sim Dietrich Jr. GPU Toon Shading. NVIDIA Corporation. `http://developer.nvidia.com/object/toon_shading_geforce256.html`.

[29] Pat Duke. Personal communication, 2004.

[30] Gershon Elber. Line Art Rendering via a Coverage of Isoparametric Curves. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):231–239, September 1995.

[31] Gershon Elber. Line Art Illustrations of Parametric and Implicit Forms. *IEEE Transactions on Visualization and Computer Graphics*, 4(1), January–March 1998.

[32] Gershon Elber. Interactive Line Art Rendering of Freeform Surfaces. *Computer Graphics Forum*, 18(3):1–12, September 1999.

[33] Gershon Elber and Elaine Cohen. Hidden Curve Removal for Free Form Surfaces. In *SIGGRAPH '90: Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, volume 24, pages 95–104, August 1990.

[34] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Image Segmentation Using Local Variation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, page 98. IEEE Computer Society, 1998.

[35] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Distance Transforms for Sampled Functions. Technical Report TR2004-1963, Cornell Computing and Information Science, 2004.

[36] Max Fleischer. Method of Producing Moving Picture Cartoons, 1917. US Patent no. 1,242,674.

[37] L. R. Ford and D. R. Fulkerson. Maximal Flow Through a Network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[38] R. E. Gomory and T. C. Hu. Multi-Terminal Network Flows. *Journal of SIAM*, 9(4):551–570, December 1961.

[39] Bruce Gooch, Greg Coombe, and Peter Shirley. Artistic Vision: Painterly Rendering Using Computer Vision Techniques. In *NPAR '02: Proceedings of the 2nd International Symposium on Non-Photorealistic Animation and Rendering*, pages 83–90. ACM Press, 2002.

[40] Stéphane Grabli, Frédo Durand, and François Sillion. Density Measure for Line-Drawing Simplification. In *Proceedings of Pacific Graphics*, 2004.

[41] Stéphane Grabli, Emmanuel Turquin, Frédo Durand, and François Sillion. Programmable Style for NPR Line Drawing. In *Rendering Techniques 2004 (Eurographics Symposium on Rendering)*. ACM Press, June 2004.

[42] Matt Groening. Futurama, 1999–2003. 20th Century Fox.

[43] Paul Haeberli. Paint by Numbers: Abstract Image Representations. In *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, pages 207–214. ACM Press, 1990.

[44] Alejo Hausner. Simulating Decorative Mosaics. In *SIGGRAPH '01: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 573–580. ACM Press, 2001.

[45] Aaron Hertzmann. Painterly Rendering with Curved Brush Strokes of Multiple Sizes. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pages 453–460. ACM Press, 1998.

[46] Aaron Hertzmann. Fast Paint Texture. In *NPAR '02: Proceedings of the 2nd International Symposium on Non-Photorealistic Animation and Rendering*, pages 91–96. ACM Press, 2002.

[47] Aaron Hertzmann and Ken Perlin. Painterly Rendering for Video and Interaction. In *NPAR '00: Proceedings of the 1st International Symposium on Non-Photorealistic Animation and Rendering*, pages 7–12. ACM Press, 2000.

[48] Aaron Hertzmann and Denis Zorin. Illustrating Smooth Surfaces. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 517–526. ACM Press/Addison-Wesley Publishing Co., 2000.

[49] D. Hochbaum and D. Shmoys. A Best Possible Heuristic for the K-Center Problem. *Mathematics of Operations Research*, 1985.

[50] Tobias Isenberg, Bert Freudenberg, Nick Halper, Stefan Schlechtweg, and Thomas Strothotte. A Developer's Guide to Silhouette Algorithms for Polygonal Models. *IEEE Computer Graphics and Applications*, 23(4):28–37, 2003.

[51] Tobias Isenberg, Nick Halper, and Thomas Strothotte. Stylizing Silhouettes at Interactive Rates: From Silhouette Edges to Silhouette Strokes. In *Computer Graphics Forum (Proceedings of EUROGRAPHICS)*, volume 21, pages 249–258. Blackwell Publishing, September 2002.

[52] Robert D. Kalnins, Philip L. Davidson, Lee Markosian, and Adam Finkelstein. Coherent Stylized Silhouettes. *ACM Transactions on Graphics*, 22(3):856–861, July 2003.

[53] Robert D. Kalnins, Lee Markosian, Barbara J. Meier, Michael A. Kowalski, Joseph C. Lee, Philip L. Davidson, Matthew Webb, John F. Hughes, and Adam Finkelstein. WYSIWYG NPR: Drawing Strokes Directly on 3D Models. In *SIGGRAPH '02: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, pages 755–762. ACM Press, 2002.

[54] R. Kannan, S. Vempala, and A. Veta. On Clusterings: Good, Bad and Spectral. In *FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, page 367. IEEE Computer Society, 2000.

[55] Allison W. Klein, Wilmot Li, Michael M. Kazhdan, Wagner T. Corrêa, Adam Finkelstein, and Thomas A. Funkhouser. Non-Photorealistic Virtual Environments. In *SIGGRAPH '00: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 527–534. ACM Press/Addison-Wesley Publishing Co., 2000.

[56] Michael A. Kowalski, Lee Markosian, J. D. Northrup, Lubomir Bourdev, Ronen Barzel, Loring S. Holden, and John F. Hughes. Art-Based Rendering of Fur, Grass, and Trees. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 433–438. ACM Press/Addison-Wesley Publishing Co., 1999.

[57] John Lansdown and Simon Schofield. Expressive Rendering: A Review of Nonphotorealistic Techniques. *IEEE Computer Graphics and Applications*, 15(3):29–37, May 1995.

[58] Anat Levin, Dani Lischinski, and Yair Weiss. Colorization Using Optimization. *ACM Transactions on Graphics*, 23(3):689–694, 2004.

[59] Yin Li, Jian Sun, Chi-Keung Tang, and Heung-Yeung Shum. Lazy Snapping. *ACM Transactions on Graphics*, 23(3):303–308, 2004.

[60] Richard Linklater. *A Scanner Darkly*. To be released.

[61] Richard Linklater. *Waking Life*, 2001. 20th Century Fox.

[62] Peter Litwinowicz. Processing Images and Video for an Impressionist Effect. In *SIGGRAPH '97: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, pages 407–414, New York, NY, USA, August 1997. ACM Press/Addison-Wesley Publishing Co.

[63] J. MacQueen. Some Methods for Classification and Analysis of Multivariate Observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297. University of California Press, 1967.

[64] Lee Markosian, Michael A. Kowalski, Daniel Goldstein, Samuel J. Trychin, John F. Hughes, and Lubomir D. Bourdev. Real-Time Nonphotorealistic Rendering. In

*Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, pages 415–420. ACM Press/Addison-Wesley Publishing Co., 1997.

[65] D. Martín, J. D. Fekete, and J. C. Torres. Flattening 3D Objects Using Silhouettes. In *Computer Graphics Forum (Proceedings of EUROGRAPHICS)*, volume 21, pages 239–248. Blackwell Publishing, September 2002.

[66] Scott McCloud. *Understanding Comics: The Invisible Art*. Kichen Sink Press, 1993.

[67] Morgan McGuire and John F. Hughes. Hardware-determined feature edges. In *NPAR '04: Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, pages 35–147. ACM Press, 2004.

[68] TDK Mediactive. RoboTech: Battlecry, 2002.

[69] Barbara J. Meier. Painterly Rendering for Animation. In *SIGGRAPH '96: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, Computer Graphics Proceedings, Annual Conference Series, pages 477–484, August 1996.

[70] A. Ng, M. Jordan, and Y. Weiss. On Spectral Clustering: Analysis and an Algorithm. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, 2002.

[71] J. D. Northrup and Lee Markosian. Artistic Silhouettes: A Hybrid Approach. In *NPAR '00: Proceedings of the 1st International Symposium on Non-Photorealistic Animation and Rendering*, pages 31–38. ACM Press, 2000.

[72] Ken Perlin. An Image Synthesizer. In *SIGGRAPH '85: Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, pages 287–296. ACM Press, 1985.

[73] Pietro Perona and William T. Freeman. A Factorization Approach to Grouping. In *ECCV '98: Proceedings of the 5th European Conference on Computer Vision*, pages 655–670. Springer-Verlag, 1998.

[74] Ferdinand Petrie and John Shaw. *The Big Book of Painting Nature in Watercolor*. Watson-Guptill Publications, 1990.

[75] Alexander Petrov. *The Old Man and the Sea*, 1999. Panorama Film Studio of Yaroslav.

[76] Abhishek Ranjan. Motion Segmentation Using Spanning Trees and Graph Cuts. Bachelor's thesis, IIT Bombay, 2003.

[77] Ramesh Raskar. Hardware Support for Non-Photorealistic Rendering. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, pages 41–47. ACM Press, 2001.

[78] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. "Grabcut": Interactive Foreground Extraction Using Iterated Graph Cuts. *ACM Transactions on Graphics*, 23(3):309–314, 2004.

[79] Szymon Rusinkiewicz. Estimating Curvatures and Their Derivatives on Triangle Meshes. In *Symposium on 3D Data Processing, Visualization, and Transmission*, September 2004.

[80] Takafumi Saito and Tokiichiro Takahashi. Comprehensible Rendering of 3-D Shapes. In *SIGGRAPH '90: Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, pages 197–206. ACM Press, 1990.

[81] Michael P. Salisbury, Sean E. Anderson, Ronen Barzel, and David H. Salesin. Interactive Pen-and-Ink Illustration. In *SIGGRAPH '94: Proceedings of the 21st*

*Annual Conference on Computer Graphics and Interactive Techniques*, pages 101–108. ACM Press, 1994.

[82] Michael P. Salisbury, Michael T. Wong, John F. Hughes, and David H. Salesin. Orientable Textures for Image-Based Pen-and-Ink Illustration. In *SIGGRAPH '97: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, pages 401–406. ACM Press/Addison-Wesley Publishing Co., 1997.

[83] Anthony Santella and Doug DeCarlo. Visual Interest and NPR: An Evaluation and Manifesto. In *NPAR '04: Proceedings of the 3rd International Symposium on Non-Photorealistic Animation and Rendering*, pages 71–150. ACM Press, 2004.

[84] Georges Schwizgebel. *L'homme sans ombre*, 2004. Studio GDS, the National Film Board of Canada, Télévision Suisse Romande.

[85] E. Sharon, A. Brandt, and R. Basri. Fast Multiscale Image Segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 70–77. IEEE Computer Society, 2000.

[86] Jianbo Shi and Jitendra Malik. Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, August 2000.

[87] Smilebit and Sega. Jet Set Radio Future, 2002.

[88] Daniel A. Spielman and Shang-Hua Teng. Spectral Partitioning Works: Planar Graphs and Finite Element Meshes. In *Proceedings of the 37th Annual IEEE Conference on Foundations of Computer Science*, pages 96–105, 1996.

[89] Jhonen Vasquez. Invader Zim, 2001–2003. Viacom.

[90] Olga Veksler. Image Segmentation by Nested Cuts. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, page 1339. IEEE Computer Society, June 2000.

[91] Jue Wang, Yingqing Xu, Heung-Yeung Shum, and Michael F. Cohen. Video Tooning. *ACM Transactions on Graphics*, 23(3):574–583, August 2004.

[92] Yair Weiss. Segmentation Using Eigenvectors: A Unifying View. In *ICCV '99: Proceedings of the International Conference on Computer Vision*, volume 2, pages 975–982. IEEE Computer Society, 1999.

[93] Brett Wilson and Kwan-Liu Ma. Rendering Complexity in Computer-Generated Pen-and-Ink Illustrations. In *Proceedings of the 3rd International Symposium on Non-Photorealistic Animation and Rendering*, pages 129–137. ACM Press, 2004.

[94] Georges Winkenbach and David H. Salesin. Computer-Generated Pen-And-Ink Illustration. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, pages 91–100. ACM Press, 1994.

[95] Georges Winkenbach and David H. Salesin. Rendering Parametric Surfaces in Pen and Ink. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 469–476. ACM Press, 1996.

[96] Z. Wu and R. Leahy. An Optimal Graph Theoretic Approach to Data Clustering: Theory and Its Application to Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1101–1113, 1993.

[97] Brian Wyvill, Kees van Overveld, and Sheelagh Carpendale. Rendering Cracks in Batik. In *NPAR '04: Proceedings of the 3rd International Symposium on Non-Photorealistic Animation and Rendering*, pages 61–70. ACM Press, 2004.

[98] T. Yamazaki. Introduction of EM Algorithm into Color Image Segmentation. In *IEEE International Conference on Intelligent Processing Systems*, pages 368–371, August 1998.

[99] Stella X. Yu and Jianbo Shi. Multiclass Spectral Clustering. In *Ninth IEEE International Conference on Computer Vision*, pages 313–319, October 2003.

[100] Lihi Zelnik-Manor and Pietro Perona. Self-Tuning Spectral Clustering. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*. MIT Press, 2005.