Eurographics Symposium on Rendering (2006)
Tomas Akenine-Möller and Wolfgang Heidrich (Editors)

Segmentation-Based 3D Artistic Rendering

Alexander Kolliopoulos Jack M. Wang Aaron Hertzmann

University of Toronto†

Abstract

This paper introduces segmentation-based 3D non-photorealistic rendering, in which 3D scenes are rendered as a collection of 2D image segments. Segments abstract out unnecessary detail and provide a basis for defining new rendering styles. These segments are computed by a spectral clustering algorithm that incorporates 3D information, including depth, user-defined importance, and object grouping. Temporally coherent animation is created by biasing adjacent frames to have similar segmentations. We describe algorithms for rendering segments in styles inspired by a number of hand-painted images.

1. Introduction

Non-photorealistic rendering (NPR) algorithms allow us to create new forms of artistic 3D rendering and to explore the nature of art. An important element in painting and drawing is the partitioning of an image into *segments*, or distinct image regions. Although one does not normally interpret artistic images in this way, if we do view them looking for segments, the degree to which they can be found is striking. For example, Figures 2 and 3 show images with distinct artistic styles, but in each it is possible to identify some notion of segments, where distinct objects or texture have been grouped together. In general, each segment corresponds to grouping scene elements into a single 2D image region, and each segment is drawn as a single unit. Notice, for example, how in Figure 2, paint builds up near the segment boundaries that we have manually identified, as the artist carefully controls the strokes in those regions. Furthermore, these strokes near segment boundaries tend to follow the contours of those boundaries. Segments can be drawn with a single watercolor wash, a group of paint strokes, a solid color, or a variety of other techniques. Depending on the artistic style, segmentation performs several vital functions. First, segmentation helps abstract out unnecessary details, thereby clarifying the content of a scene. Second, segments can create a sense of 2D design, independent of the goal of expressing 3D content. Third, a good segmentation makes drawing easier, since each segment can be drawn with a single stroke or wash. It is not always straightforward to describe what makes a good segmentation, since it is highly dependent on many factors, including viewing position, scene geometry, the relative importance of different objects, and the artistic style. Segments are view-dependent, as illustrated by the change in detail of distant objects as they move closer to the viewer in the frames of hand-painted animation in Figure 3. We do not claim that most artists consciously segment images. Instead, we argue that, conscious or not, some form of segmentation is present in many artistic styles, and that segmentation provides an important tool for understanding and mimicking artistic styles.

For these reasons, a general approach to segmentation for automatic non-photorealistic rendering could be a valuable tool for designing new and interesting artistic styles. In this paper, we introduce segmentation-based 3D non-photorealistic rendering and animation, in which 3D scenes are rendered using segments as a fundamental primitive. To render a scene, our system computes a segmentation of the image plane into distinct segments; each segment is then rendered independently of the others. We show that using segments allows us to define a variety of NPR styles, with similar stylization and abstraction to those in hand-made images. We also describe a technique for producing temporally-coherent segmentations in order to create artistic animation.

Although previous authors have computed segmentations for image-based NPR, these systems produced rendering styles very closely tied to the choice of image-processing algorithms. In contrast, we consider rendering of 3D scenes, which frees us from the difficult problem of extracting scene

† E-mail: {alexk, jmwang, hertzman}@dgp.toronto.edu

Figure 1: *Left: A 3D scene. Center: A segmentation of the scene. Right: A painterly style applied to the segmented scene.*

properties from a photograph—our system makes use of geometric information and user annotations to produce better segmentations. This system uses a general approach to segmentation, taking into account the rich information available in a 3D scene and allowing a user to weight the effect of these features on the outcome of the segmentation process.

Our approach is as follows. For each frame, we first compute colors, depths, normals, and object IDs for each pixel by standard 3D rendering. We then segment the image based on these features, using a spectral clustering algorithm (Section 3). Temporal coherence in animation is encouraged by segmenting adjacent frames together. Given a segmentation, we can then render the scene in a variety of styles that explicitly make use of the segments (Section 4). Figure 1 shows such a segmentation of a 3D scene and a corresponding artistic style.

2. Related work

Some authors have used a notion of a user-specified segmentation in NPR. In some of the earliest NPR work, the rendering style is set manually for different objects [WS94] or image regions [CAS*97, SWHS97]. These systems require some form of user intervention to produce each image. In contrast, our system requires a user to author a 3D scene and rendering style, but can then automatically render arbitrary new views once segmentation parameters have been selected.

Fixed segmentations have been applied in object-space for 3D scenes. For example, 3D scenes in [CAS*97] are rendered with each object matted independently. The system described in [KHRO01] requires a user to group objects and specify stylistic parameters for the groups. Luft and Deussen [LD06] manually assign objects or groups of objects to segments to be rendered in a watercolor style. Using a fixed, object-based segmentation is computationally inexpensive, but it does not directly capture view-dependent effects. Performing segmentation in image-space, however, allows segments to dynamically abstract away or highlight detail in a view-dependent fashion.

Figure 2: *Left: Detail of Wayne Thiebaud's "Around the Cake" (Oil on canvas, 1962). Right: For purposes of illustration, we have manually identified segments within the painting, each of which roughly corresponds to a distinct set of brush stroke orientations, sizes, and/or colors.*

A number of authors have employed automatic image segmentation algorithms for NPR in 2D. DeCarlo and Santella [DS02] create abstracted image representations based on image segmentation and eye-tracking data; each segment is smoothed and rendered as a solid color with occasional edge strokes. Gooch et al. [GCS02] segment an image into very small regions and fit a single paint stroke to each region. Bangham et al. [BGH03] render each segment of a photograph with a solid color. Bousseau et al. [BKTS06] also apply their watercolor style to segmented photographs. Similar approaches have been applied to processing video sequences: in each case, the video sequence is broken into space-time segments, and each segment is rendered with a simple solid-color rendering style [WXSC04, CRH05]. All of these papers produced very high-quality, appealing results; however, most of the effort has focused on coupling the result of segmentation to a single rendering style. Our work is different from these previous works in that we present a general segmentation algorithm for rendering 3D scenes, incorporating 3D geometric information, and we show how a variety of rendering styles can be built from a segmentation of a scene.

Many rendering techniques have been developed for rendering 3D scenes, such as contour rendering [ST90], pen-and-ink illustration [WS94], and painting [Mei96]. These systems are largely complementary to ours: these papers

Figure 3: *Sequence of frames from Georges Schwizgebel's animation "L'Homme sans Ombre" (oil and acrylic on cells, 2004), showing a range of detail levels. Buildings in the distance are illustrated as simple boxes, but with much more detail in close-up.*

mainly describe various rendering styles for 3D, whereas we describe techniques for automatically assigning styles to different parts of a scene.

Our work is directly related to NPR methods that perform adaptive simplification of a 3D rendering, for example, varying detail as a function of object depth or image density [KLK*00, KMN*99, WS94]. A related approach is to remove strokes to match target tones or density [DS00, GDS04, WM04]. These systems are designed for simplifying specific styles of stroke renderings based on local criteria such as depth and tone. In contrast, our system performs global simplification of an image and provides a basis for a variety of rendering styles. These systems are complementary to ours and could possibly be combined with it. For example, adaptive stroke rendering could be applied to each image region based on the rendering style for that segment.

Our work bears superficial similarity to Level-of-Detail (LOD) techniques, but the goals are fundamentally different. LOD methods seek to improve performance by reducing scene complexity *without* changing the appearance of the scene. In contrast, our goal is to *modify* the visual appearance of the scene in an artistic manner, without regard to performance.

3. Segmenting 3D renderings

In this section, we describe a graph-based procedure for segmenting an image of a 3D rendering.

There are many existing image segmentation algorithms, but not all of them are suitable for our purposes. For our application, the method must be automatic (including determining the number of segments), must produce reasonable segmentations for complex images, must produce temporally-coherent segmentations, and must also be able to take into account object groupings, e.g., the fact that two pixels are or are not part of the same object. Additionally, we would like the method to be reasonably fast. One well-known segmentation algorithm is k-means, which is limited to producing roughly spherical segments, and cannot produce the curved or narrow segments observed in Figures 2 and 3. Furthermore, the number of segments must be specified in advance. The Mean Shift algorithm [CM02] can produce more complex segmentations and determine the number of segments, but it is not clear how to take grouping information into account, or how to enforce temporal coherence without performing segmentation as a batch process, as in [WXSC04]. Min-cut algorithms can produce very good segmentations on images [BK04] but require significant user guidance. We also experimented with local segmentation heuristics, but they yielded poor results for complex scenes.

Our segmentation is derived from the Normalized Cuts algorithm [SM00], a graph-based spectral method. Normalized Cuts optimizes a clustering metric that simultaneously minimizes a measure of similarity between different segments and maximizes similarity within them, while requiring the user to only set a single segmentation threshold parameter once the weighting function that specifies the similarity between pixels is fixed. Being a graph-based algorithm for segmentation, Normalized Cuts does not require an explicit feature space, so the weighting function can be specified directly to give one more intuitive control over the meaning of segmentation parameters. We introduce extensions to the Normalized Cuts algorithm that provide for temporal coherence and faster computation.

3.1. Graphs from 3D scenes

We begin by constructing an undirected graph $G = (V, E)$ from the image, with nodes V and edges E. 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. To produce the affinities, we first render several reference images: a color image of the scene, a depth map, a normal map, an object ID reference image, and an importance map. 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 object shaded in proportion to its importance, that is, an unimportant element is rendered with a darker shade than a more important element.

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

$$f_i = \left(w_c c_i, w_n n_i, \frac{w_z}{z_i + \beta} \right)^T \quad (1)$$

where $c_i = (r_i, g_i, b_i)^T$ is the color of the pixel, n_i is the camera-space normal at the pixel, z_i is the depth of the pixel, all w_c, w_n, and w_z are user-defined weights, and β is a user-defined bias on depth. 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 absolute 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.

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

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

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 σ_{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, then o_{ij} is set to $w_o > 1$; if the object IDs are the same, then $o_{ij} = 1$. This has the effect of weakening edges connecting nodes between two different objects in the scene. Group IDs are used with g_{ij} in a similar fashion, where objects may be tagged with group IDs by a user. If p_i and p_j belong to different groups, $g_{ij} = w_g > 1$, otherwise $g_{ij} = 1$. For example, this allows us to specify that a group of bushes should be segmented together before being segmented with a nearby object that happens to also be green. 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 $s_{ij} = \left(\max(s_i, s_j)\right)^{w_s}$, where $w_s \geq 1$ is the 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.

3.2. Normalized Cuts

We now review the Normalized Cuts algorithm [SM00]. Let A and B be any two disjoint sets of nodes in some graph G. The *cut* between A and B is defined as

$$\text{cut}(A,B) = \sum_{u \in A, v \in B} w(u,v). \quad (3)$$

Similarly, the *association* between A and V is

$$\text{assoc}(A,V) = \sum_{u \in A, t \in V} w(u,t). \quad (4)$$

The *normalized cut* between A and B is

$$\text{Ncut}(A,B) = \frac{\text{cut}(A,B)}{\text{assoc}(A,V)} + \frac{\text{cut}(A,B)}{\text{assoc}(B,V)} \quad (5)$$

The image segmentation problem is to partition the graph into segments A and B in a manner that minimizes $\text{Ncut}(A,B)$.

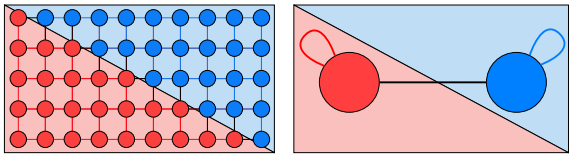

Figure 4: *Condensing a graph for Normalized Cuts. Left: A graph produced from a rendering of two colored triangles. Each node in the graph corresponds to a pixel, and edges connect adjacent pixels. Right: A corresponding condensed graph. Each condensed node has a self-edge with weight equal to the sum of the weights between its nodes in the original graph. The weight between the two condensed nodes is equal to the sum of the weights across the triangle boundary in the original graph.*

Finding the optimal normalized cut is NP-complete, but an approximate solution can be obtained as follows. Let $\mathbf{x}$ be a vector with N elements, each corresponding to one node in G, where $\mathbf{x}_i = 1$ if node i is in segment A, and $\mathbf{x}_i = -1$ otherwise. Let $\mathbf{D}$ be the *degree matrix*, a diagonal $N \times N$ matrix with $\mathbf{D}_{ii} = \sum_j w(i,j)$, and let $\mathbf{W}$ be the symmetric *weight matrix*, $\mathbf{W}_{ij} = w(i,j)$. With $\hat{\mathbf{W}} = \mathbf{D} - \mathbf{W}$, it can then be shown that

$$\text{Ncut}(\mathbf{x}) = \frac{(\mathbf{1}+\mathbf{x})^T \hat{\mathbf{W}} (\mathbf{1}+\mathbf{x})}{4k\mathbf{1}^T\mathbf{D}\mathbf{1}} + \frac{(\mathbf{1}-\mathbf{x})^T \hat{\mathbf{W}} (\mathbf{1}-\mathbf{x})}{4(1-k)\mathbf{1}^T\mathbf{D}\mathbf{1}} \quad (6)$$

where $k = \left(\sum_{\mathbf{x}_i > 0} \mathbf{D}_{ii}\right) / \left(\sum_i \mathbf{D}_{ii}\right)$ and $\mathbf{1}$ is an $N \times 1$ vector of ones. Moreover, it can be shown that, if we relax the problem to allow elements of $\mathbf{x}$ to take on any real value, $\text{Ncut}(\mathbf{x})$ is minimized by the eigenvector corresponding to the second smallest eigenvalue of the matrix $\mathbf{D}^{-1/2}\hat{\mathbf{W}}\mathbf{D}^{-1/2}$. This solution can be converted into a graph partition by thresholding $\mathbf{x}$, so that values above the threshold are set to 1 and the rest are set to -1. The threshold is chosen to minimize $\text{Ncut}(\mathbf{x})$ by exhaustively searching through all possible $N-1$ values.

To segment a graph into more than two partitions, the same Normalized Cut algorithm is recursively applied to each segment. The process stops when $\text{Ncut}(\mathbf{x})$ exceeds a user-defined threshold, τ_n, and no additional cuts are made.

3.3. Condensing graphs

The Normalized Cuts algorithm is very slow for even moderately-sized graphs, since it requires computing eigenvectors of a very large matrix. In order to get faster results, we apply Normalized Cuts to a reduced graph G' in which each node may correspond to many pixels. This graph is produced by combining all pairs of graph nodes sharing an edge with a weight larger than a user-defined threshold, τ_c; the remaining nodes comprise the nodes of G'. By decreasing this threshold, a user can obtain faster performance, although at the cost of a possible undersegmentation.

A naïve approach to computing the edges of this condensed graph would be to set the weight of the new edges of G' to be the sum of the weights of their corresponding edges of G. Unfortunately, this gives a poor approximation to clustering the original graph with Normalized Cuts. For example, consider an image made up of a red triangle and an adjacent blue triangle (Figure 4). In general, the optimal normalized cut for this image should separate the two triangles. However, suppose we condensed this graph to two nodes. Each node would correspond to a triangle, with a single condensed edge with weight w between them, where w is equal to the sum of the weights of the edges between the two triangles. Then, there is one possible cut, and it can easily be shown that the cost of this cut is 2, regardless of the original edge weights. Since this cost is the maximum possible cost of a cut, the edge will never be cut, as it exceeds any sensible user-specified value for the cut threshold, τ_n.

To address this, we augment G' by adding an edge from every condensed node u' to itself. The weight of this edge is equal to the sum of the edge weights collapsed on u':

$$w(u',u') = \sum_{u,v \in S(u')} w(u,v), \tag{7}$$

where $S(u')$ is the set of nodes in G that correspond to $u' \in G'$. We can then apply Normalized Cuts to G'. Since every node in G corresponds to one node in G', a cut of G' can be directly converted to a cut of G. Moreover, it is straightforward to show that the condensing operation has the following desirable property: every cut of G' has the same cost as the corresponding cut of G (Appendix A). That is, segmenting a condensed graph is equivalent to segmenting its original graph under the constraint that condensed edges cannot be cut.

In our implementation, we compute the weights for the edges of the full graph every time a new frame is rendered. The condensed graph is then constructed by visiting each node in the graph once and performing a breadth-first search along edges above the condensing threshold for nodes to add to the current condensed node. Finally, the weights on the condensed edges are summed from their corresponding edges in the full graph, and the condensed graph is ready to be segmented.

3.4. Temporal coherence

When rendering individual frames of an animation sequence independently, there is no guarantee that consecutive frames will yield consistent segmentations. We motivate a solution to this problem as follows. Suppose that we were segmenting an entire video sequence at once. In addition to constructing a graph for each frame, we would introduce graph edges between nodes in adjacent frames, possibly scaled by a user-specified weight w_k. In our algorithm, we only segment one frame at a time, so we can discard graph nodes for all future frames. Moreover, since the segmentation for the previous frame is already known, we can collapse the subgraph for the previous frame into one node per segment while discarding older frames.

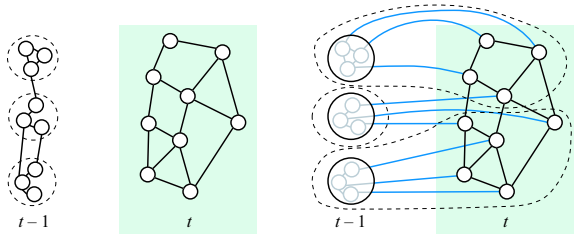

Figure 5: *Segmentation coherency. Left: The previous frame's graph nodes at $t-1$ are grouped into coherency nodes based on their segmentation (inside the dashed circles). Right: The coherency nodes in the previous frame are assigned edges (the blue lines) to nodes in the current frame t with weights scaled by w_k, and the combined graph is segmented.*

We call these condensed nodes corresponding to segments from the previous frame *coherency nodes* (illustrated in Figure 5). Introducing coherency nodes has the effect of segmenting adjacent frames together, under the constraint that the segmentation from the previous frame cannot be modified. Consequently, the segmentation for the current frame is biased to be similar to the previous frame.

In practice, we link together pixels at the same positions in image-space between frames. This works well for larger segments, but thin or fast moving segments might share few edges between frames. An approach to handling such details might be to associate areas of geometry with specific segments and track these areas between frames in object-space, however, the computational cost of associating pixels with points on geometry would be high.

4. Rendering styles

In this section, we demonstrate several artistic styles based on segmentation. There are a large number of segmentation parameters to tune, but they were selected to have an intuitive meaning in adjusting the composition of a scene. We found that setting parameters to achieve a reasonable segmentation for a single image is typically easy; sometimes using only a subset of the available parameters is sufficient for producing an acceptable segmentation. It can be more difficult to find a set of parameters that works well over an entire animation due to the need to tune the coherency, but once a set of parameters is found, the segmentation tends to work quite well. It is necessary to initialize a slightly different set of segmentation parameters for the first frame than for subsequent frames of an animation, since segment sizes should be smaller due to the lack of coherency nodes. Pa-

	c	w_c	w_n	w_z	β	w_o	w_g	w_s	w_k	τ_c	τ_n
Still life	0.25	8	1	2	1	2	2	-	-	0.15	0.005
Forest (initial frame)	0.25	3	-	50	0	2	5	6	-	0.75	0.015
Forest (subsequent frames)	0.25	3	-	50	0	3	5	6	0.2	0.75	0.08
Mug (fewer segments)	0.25	-	-	-	-	-	3	-	-	0.75	0.5
Mug (more segments)	0.25	1	0.5	-	-	-	2	-	-	0.75	0.5

Table 1: *Parameters used to generate segmentations for Figures 6-9. A dash indicates that a parameter's associated reference image was not rendered to compute the segmentation. The forest scene images were rendered as frames of an animation with coherency, and all other images were rendered independently.*

rameters used to generate all of the figures are given in Table 1.

Next, we discuss some of the specific rendering styles based on the result of a segmentation of a 3D scene.

Toon. A basic style is a cartoon rendering style, with toon shading and increased brightness and saturation applied to a scene. The segmentation is used to reduce contour density, rendering object contours only near the boundaries of the segments in image space. Contours within a segment are not drawn, eliminating unnecessary clutter. For example, the tree line in Figure 6 contains many contour edges that can be distracting. The set of boundary-contours is determined by sampling segment IDs in the image plane. This pixel sampling method does lead to a small number of contours being incorrectly labeled, but the errors are typically minor. Segments may be solidly shaded with the average color of the pixels inside of them, or a user may chose to only solidly shade segments with a large average depth. As segments get closer, they may be faded into their standard toon shading to reveal more detail.

Painted strokes. We also demonstrate a painterly rendering technique (Figure 7) capable of producing effects similar to those of Figures 2 and 3. In this style, each segment is filled with long, curved paint strokes based on the underlying reference color, similar to those of [Her98], until a target stroke density is met. Strokes are placed by selecting seed points randomly in the image plane, and rejecting those that fall in a region that is too dense. The length of a stroke is traced out from each seed, as in [JL97]. A distance field is computed for each segment using the algorithm described in [FH04], and points within a user-set distance from a segment boundary follow the isocontours of this distance field. Strokes further from a segment boundary follow a path based on the normals of the 3D geometry, as follows. Let n be the camera-space normal at a pixel. Then the direction of a stroke at that pixel is set to $(\max(|n_y|,|n_z|/2),n_x)^T$. Figure 7 shows the resulting stroke directions on one scene. This causes strokes to carefully follow segment boundaries, as in Figure 2. Strokes are terminated when they reach a maximum length, turn by an angle greater than a threshold, or enter an area where the difference in color is greater than a threshold. After a sufficient number of strokes are selected for rendering, addi-

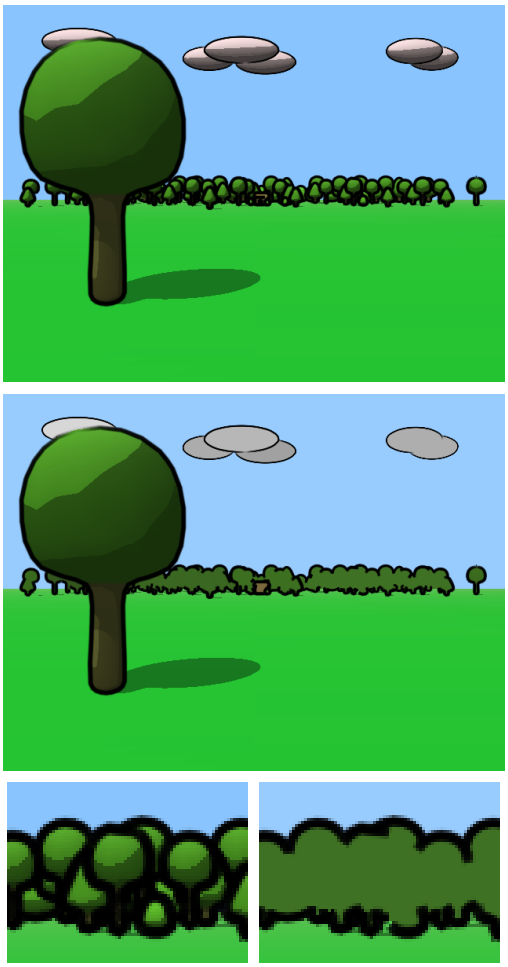

Figure 6: *A toon style. Top: A forest scene with no segmentation. Note how the contours are cluttered in the background where there are many trees. Center: Segmentation is applied to group together many of the background elements. Contours are only drawn near segment boundaries, resulting in a cleaner image. Bottom: Detail of the background without segmentation (left) and with segmentation (right).*

Figure 7: *The use of segmentation for a painterly style. Top: Only surface normals determine stroke direction, and length or differences in color terminate strokes. Bottom: Strokes near segment boundaries follow their path and strokes do not cross into adjacent segments.*

tional passes may be made to render thinner strokes in gaps that are difficult to fill with broad strokes. Strokes may optionally be rendered using Hertzmann's relief texturing algorithm [Her02], with greater changes in relief normals near segment boundaries, to simulate a build up of paint characterized by Figure 2, as in Figure 1. For coherency in animation we use the approach described in [Mei96]; seed points are projected onto the models to be tracked in 3D between frames. At each frame, seed points furthest from the viewer are removed in areas where the stroke density is too high, and new strokes are created in areas where the density is too low.

Stippling. A simple stippled style is demonstrated as well (Figure 8). This style is rendered by drawing more points in darker regions and near segment boundaries. A Perlin noise texture is first generated, with each value in $[0,1]$ corresponding to a point where a stippled point might be drawn. A function of the reference image darkness $d_r \in [0,1]$ and the average segment darkness $d_s \in [0,1]$ is computed at that pixel and scaled according to the distance δ to the nearest segment boundary. If the resulting value is greater than the value of the noise at the same pixel, a point is rendered. In our implementation, the function is defined as $f = (c + \alpha_r d_r + \alpha_s d_s) b_1 / (\delta^2 + b_2)^{b_3}$, where $c = 0.3$, $\alpha_r = 0.5$, $\alpha_s = 0.2$, $b_1 = 2.75$, $b_2 = 256$, and $b_3 = 0.25$. Frame to frame coherency is achieved by using the same noise texture throughout an animation. This technique is fast and avoids points "sticking" to the 3D geometry.

The rendering times for our algorithms depend highly on the scene complexity and the rendering style. Reference images and segmentations for the still life in Figure 7 with around 600 condensed nodes requires about 20 seconds per frame, on average, using a 3.4GHz Intel Xeon. The painting style takes an additional 40 seconds to compute, or 15 seconds for the toon style with contour reduction (most of this time is spent computing object space contours on the scene). However, the forest scene in Figure 6 can take as little as 5 seconds to render and segment over 100 condensed nodes, with another 15 seconds to draw paint strokes or 5 seconds to compute segment-boundary contours. Regardless of scene content, the stippled style requires less than one third of a second to render. Segmentation time is increased by complex scenes that result in a high number of condensed nodes, slowing down the eigenvalue decomposition. The painterly style can run slower with detailed models, since projecting the stroke seed points onto the 3D geometry for tracking in animation requires a large number of rays to be cast. Similarly, the toon style depends primarily on the time to compute view-dependent contours on the scene.

5. Discussion and future work

In this paper, we have introduced algorithms for creating segmented renderings of 3D scenes. This system produces image and video representations that abstract out distant and unimportant scene detail in a compelling and effective manner. Moreover, segmentation provides a basis for various artistic styles, as shown in Figure 9.

An important problem is to perform segment-based rendering in real-time. The time to compute the 613,280 edge weights alone takes about a quarter of a second for a 640x480 image, using only a single reference image. This does not take into account the overhead of computing the segmentation and rendering the final artistic style. An approach might be to compute the segmentation using reference images at a much lower resolution and propagating the segmentation to the final, larger rendered image. This would also reduce memory requirements, but the memory needed to store the reference images and graphs is not a significant problem, with buffer sizes being fixed other than the condensed graph, which is usually small. Furthermore, such an approximation will reduce the quality of the segmentation. One possibility is to design 3D culling and level-of-detail algorithms appropriate for segmented scenes. This would be desirable, since one of the main applications we envision for segmentation is in creating artistic virtual worlds. Modern

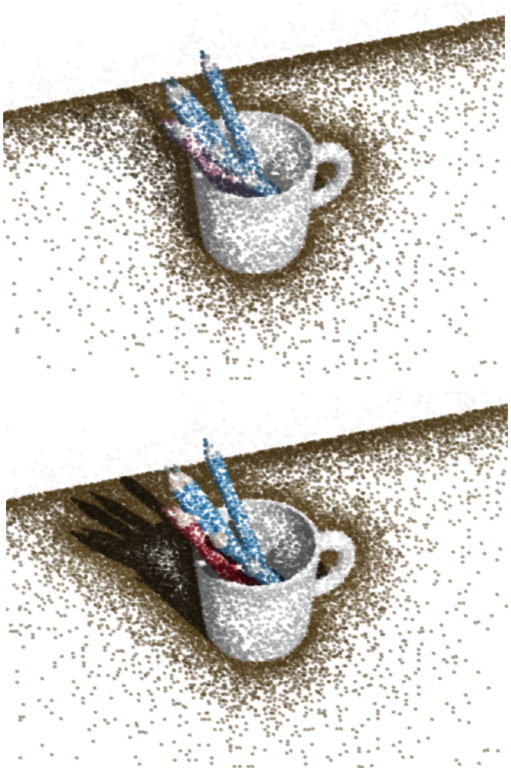

Figure 8: *The effect of different segmentations on the stippled style. Top: Only four segments are used–the pencils, mug, table, and background. Bottom: More segments are generated, such as the shadow on the table, resulting in much more focus on the outline of the shadow.*

programmable graphics hardware might also be used to construct the graph's matrix and perform the segmentation in much less time.

Our paper gives a taste of the rendering styles enabled by segmentation, but we expect that a wide variety of other styles can also be achieved. For example, it has been demonstrated that watercolor works well with segmentation [CAS*97, BKTS06, LD06], and styles such as batik and woodblock printing tend to have a segmented appearance that would fit well into our system.

Another open problem is to create a good interface for designing artistic styles that depend on a segmentation. Segments would fit very naturally into the procedural NPR shaders of Grabli et al. [GTDS04] as a fundamental primitive. A more challenging problem is to specify segmentation-based styles with a WYSIWYG interface [KMM*02].

Acknowledgments

We are grateful to Roey Flor for his contributions. This research was supported by grants from NSERC and CFI.

Appendix A: Condensed Normalized Cuts

Our method of constructing a condensed graph produces Normalized Cut values equal to the corresponding cuts on the full graph, as follows. Let $G' = (V', E')$ be a condensed graph that represents a graph $G = (V, E)$. Each condensed node u' in V' corresponds a set of nodes in V, $S(u')$. 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 in G' equal to the sum of the weights of their corresponding set of edges in G, we have $\mathrm{cut}(A', B') = \mathrm{cut}(A, B)$. This can be shown by the following:

$$\begin{aligned}
\mathrm{cut}(A', B') &= \sum_{u' \in A', v' \in B'} w(u', v') && (8)\\
&= \sum_{u' \in A', v' \in B'} \left(\sum_{u \in S(u'), v \in S(v')} w(u, v) \right) && (9)\\
&= \sum_{u \in S(A'), v \in S(B')} w(u, v) && (10)\\
&= \sum_{u \in A, v \in B} w(u, v) && (11)\\
&= \mathrm{cut}(A, B). && (12)
\end{aligned}$$

Note that $\mathrm{assoc}(A, V) = \mathrm{assoc}(A, A) + \mathrm{cut}(A, B)$, and $\mathrm{assoc}(A', V') = \mathrm{assoc}(A', A') + \mathrm{cut}(A', B')$. Thus, we need only show that $\mathrm{assoc}(A, A)$ and $\mathrm{assoc}(A', A')$ are equivalent:

$$\begin{aligned}
\mathrm{assoc}(A', A') &= \sum_{u' \neq v' \in A'} w(u', v') + \sum_{u' \in A'} w(u', u') && (13)\\
&= \sum_{u' \neq v' \in A'} \left(\sum_{u \in S(u'), v \in S(v')} w(u, v) \right) + \\
&\quad \sum_{u' \in A'} \left(\sum_{u, v \in S(u')} w(u, v) \right) && (14)\\
&= \sum_{u, v \in S(A')} w(u, v) && (15)\\
&= \sum_{u, v \in A} w(u, v) && (16)\\
&= \mathrm{assoc}(A, A). && (17)
\end{aligned}$$

As shown in [Kol05], the spectral optimization algorithm still applies to this condensed weight matrix with non-zeros on the diagonal.

References

[BGH03] BANGHAM J. A., GIBSON S. E., HARVEY R.: The art of scale-space. In *Proc. British Machine Vision Conference* (2003), pp. 569–578.

[BK04] BOYKOV Y., KOLMOGOROV V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE PAMI 26*, 9 (Sept. 2004), 1124–1137.

[BKTS06] BOUSSEAU A., KAPLAN M., THOLLOT J., SILLION F.: Interactive watercolor rendering with temporal coherence and abstraction. In *Proc. NPAR* (2006).

[CAS*97] CURTIS C. J., ANDERSON S. E., SEIMS J. E., FLEISCHER K. W., SALESIN D. H.: Computer-generated watercolor. In *Proc. SIGGRAPH* (1997), pp. 421–430.

[CM02] COMANICIU D., MEER P.: Mean shift: A robust approach toward feature space analysis. *IEEE PAMI 24*, 5 (2002), 603–619.

[CRH05] COLLOMOSSE J. P., ROWNTREE D., HALL P. M.: Stroke surfaces: Temporally coherent artistic animations from video. *IEEE Transactions on Visualization and Computer Graphics 11*, 5 (Sept. 2005), 540–549.

[DS00] DEUSSEN O., STROTHOTTE T.: Computer-generated pen-and-ink illustration of trees. In *Proc. SIGGRAPH* (2000), pp. 13–18.

[DS02] DECARLO D., SANTELLA A.: Stylization and abstraction of photographs. *ACM Transactions on Graphics 21*, 3 (July 2002), 769–776.

[FH04] FELZENSZWALB P. F., HUTTENLOCHER D. P.: *Distance Transforms for Sampled Functions*. Tech. Rep. TR2004-1963, Cornell CIS, 2004.

[GCS02] GOOCH B., COOMBE G., SHIRLEY P.: Artistic vision: Painterly rendering using computer vision techniques. In *Proc. NPAR* (2002), pp. 83–90.

[GDS04] GRABLI S., DURAND F., SILLION F.: Density measure for line-drawing simplification. In *Proc. Pacific Graphics* (2004), pp. 309–318.

[GTDS04] GRABLI S., TURQUIN E., DURAND F., SILLION F.: Programmable style for NPR line drawing. In *Proc. Eurographics Symposium on Rendering* (2004), pp. 33–44.

[Her98] HERTZMANN A.: Painterly rendering with curved brush strokes of multiple sizes. In *Proc. SIGGRAPH* (1998), pp. 453–460.

[Her02] HERTZMANN A.: Fast paint texture. In *Proc. NPAR* (2002), pp. 91–96.

[JL97] JOBARD B., LEFER W.: Creating evenly-spaced streamlines of arbitrary density. In *Proc. Eurographics Workshop on Visualization in Scientific Computing* (1997), pp. 45–55.

[KHRO01] KOWALSKI M. A., HUGHES J. F., RUBIN C. B., OHYA J.: User-guided composition effects for art-based rendering. In *Proc. Symposium on Interactive 3D Graphics* (2001), pp. 99–102.

[KLK*00] KLEIN A. W., LI W., KAZHDAN M. M., CORRÊA W. T., FINKELSTEIN A., FUNKHOUSER T. A.: Non-photorealistic virtual environments. In *Proc. SIGGRAPH* (2000), pp. 527–534.

[KMM*02] KALNINS R. D., MARKOSIAN L., MEIER B. J., KOWALSKI M. A., LEE J. C., DAVIDSON P. L., WEBB M., HUGHES J. F., FINKELSTEIN A.: WYSIWYG NPR: Drawing strokes directly on 3D models. *ACM Transactions on Graphics 21*, 3 (July 2002), 755–762.

[KMN*99] KOWALSKI M. A., MARKOSIAN L., NORTHRUP J. D., BOURDEV L., BARZEL R., HOLDEN L. S., HUGHES J. F.: Art-based rendering of fur, grass, and trees. In *Proc. SIGGRAPH* (1999), pp. 433–438.

[Kol05] KOLLIOPOULOS A.: *Image Segmentation for Stylized Non-Photorealistic Rendering and Animation*. Master's thesis, University of Toronto, 2005.

[LD06] LUFT T., DEUSSEN O.: Real-time watercolor illustrations of plants using a blurred depth test. In *Proc. NPAR* (2006).

[Mei96] MEIER B. J.: Painterly rendering for animation. In *Proc. SIGGRAPH* (1996), pp. 477–484.

[SM00] SHI J., MALIK J.: Normalized cuts and image segmentation. *IEEE PAMI 22*, 8 (Aug. 2000), 888–905.

[ST90] SAITO T., TAKAHASHI T.: Comprehensible rendering of 3-D shapes. In *Proc. SIGGRAPH* (1990), pp. 197–206.

[SWHS97] SALISBURY M. P., WONG M. T., HUGHES J. F., SALESIN D. H.: Orientable textures for image-based pen-and-ink illustration. In *Proc. SIGGRAPH* (1997), pp. 401–406.

[WM04] WILSON B., MA K.-L.: Rendering complexity in computer-generated pen-and-ink illustrations. In *Proc. NPAR* (2004), pp. 129–137.

[WS94] WINKENBACH G., SALESIN D. H.: Computer-generated pen-and-ink illustration. In *Proc. SIGGRAPH* (1994), pp. 91–100.

[WXSC04] WANG J., XU Y., SHUM H.-Y., COHEN M. F.: Video tooning. *ACM Transactions on Graphics 23*, 3 (Aug. 2004), 574–583.

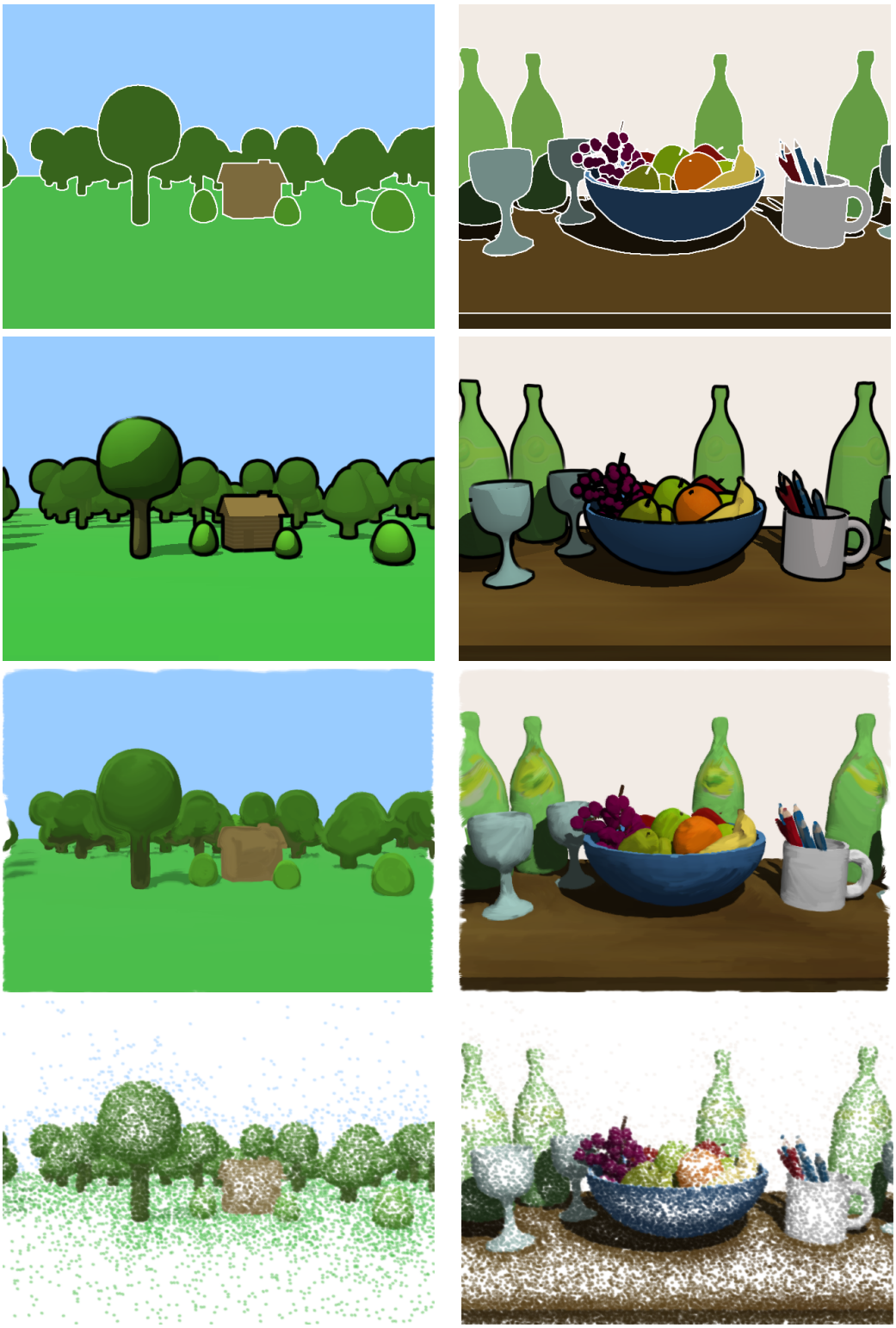

Figure 9: *Simple scenes rendered in various artistic styles based on 2D segments. From top to bottom: Scene segmentation, toon shading with segment-boundary contours, painterly strokes without relief texturing, and stippling.*