Stylized and Abstract Painterly Rendering System Using a Multiscale Segmented Sphere Hierarchy

Ming-Te Chi and Tong-Yee Lee, *Member*, *IEEE*

Abstract—This paper presents a novel system framework for interactive, three-dimensional, stylized, abstract painterly rendering. In this framework, the input models are first represented using 3D point sets and then this point-based representation is used to build a multiresolution bounding sphere hierarchy. From the leaf to root nodes, spheres of various sizes are rendered into multiple-size strokes on the canvas. The proposed sphere hierarchy is developed using multiscale region segmentation. This segmentation task assembles spheres with similar attribute regularities into a meaningful region hierarchy. These attributes include colors, positions, and curvatures. This hierarchy is very useful in the following respects: 1) it ensures the screen-space stroke density, 2) controls different input model abstractions, 3) maintains region structures such as the edges/boundaries at different scales, and 4) renders models interactively. By choosing suitable abstractions, brush stroke, and lighting parameters, we can interactively generate various painterly styles. We also propose a novel scheme that reduces the popping effect in animation sequences. Many different stylized images can be generated using the proposed framework.

Index Terms—Painterly rendering, bounding sphere hierarchy, multiscale region segmentation, stylization and abstraction, stroke-based rendering.

1 Introduction

Non-photorealistic rendering (NPR) is a powerful technique for generating images in the manner of artistic styles such as painting [1], [2], [3], [4], [5], [6], [7], pen and ink [8], [9], stippling [10], [11], and sketching [12], [13], [14]. In recent years, a large number of approaches on NPR have been proposed. Many of these earlier-stage approaches concentrated on generating still images. Several interesting systems and methods [1], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19] are later designed for rendering 3D scenes in a variety of artistic styles. Cornish et al. [15] addressed several common themes in generating NPR stylized image/animation from 3D models, including: 1) placing stylized strokes with some randomness to imitate drawing styles created by artists, 2) orientating the stroke directions to express varied artistic effects, and 3) ensuring the screen-space stroke density for frame-to-frame coherence.

The basic idea behind our framework is described in Fig. 1. First, the input model is converted into a set of 3D points, shown in Fig. 1a. We propose a novel QSplat-like [20] multiresolution hierarchy developed from bounding spheres to organize these points. Each nonleaf sphere has aggregate attributes associated with it such as its average color and accumulated size. Unlike QSplat's partitioning method which used only the sphere's spatial information, our scheme takes other attributes such as color and curvature into account. We perform a multiscale region-based segmentation on the 3D points to build a hierarchy. Our multiscale method is extended from a color image segmentation method called JSEG [21]. Our system is inspired by an image-based NPR work [22]. The aim of the proposed hierarchy is to organize the points into a meaningful structure of parts and boundaries. Fig. 1b shows a multiscale segmented hierarchy from the lower (right) to the higher (left) levels. From the right to left sides, regions with similar regularities, such as color attributes, are clustered into a larger region in the higher levels of the proposed hierarchy. In this example, each region is represented by a constant mean color. The boundaries are well maintained at different scales. By choosing suitable NPR abstractions, brush stroke for sphere nodes, and lighting/viewing parameters, we can interactively generate various painterly styles, as shown in Fig. 1c. The main contributions of this paper are as follows:

- We present a novel 3D painterly rendering framework. This framework can offer flexibility in NPR stylization and abstraction, coherent NPR quality, and interactive rendering performance on PC platforms. The flexibility can be achieved by introducing abstract operations, rules, and the dissimilarity parameters on the traversal algorithm of the proposed hierarchy. We also propose a novel smooth level transition method that reduces the popping effect in NPR animation.
- To the best of our knowledge, little work on 3D NPR has involved the color patterns of regions on 3D model surfaces. A novel QSplat-like region-based sphere hierarchy is proposed. With this hierarchy, the region boundary/edge structure at different

• *The authors are with the Computer Graphics Group/Visual System Laboratory, Department of Computer Science and Engineering, National Cheng-Kung University, No. 1, Ta-Hsueh Road, Tainan 701, Taiwan, R.O.C. E-mail: dodowell@csie.ncku.edu.tw, tonylee@mail.ncku.edu.tw.*

Manuscript received 25 Aug. 2004; revised 3 Mar. 2005; accepted 16 Mar. 2005; published online 9 Nov. 2005.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-0095-0804.

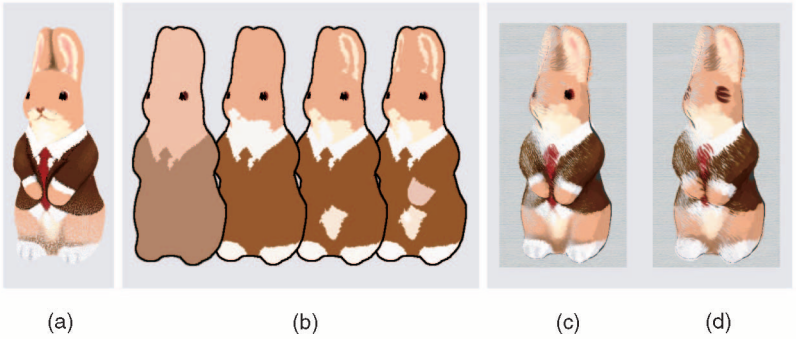


Fig. 1. (a) Input: 3D models are converted into a set of points, (b) multiscale region segmentation, and (c) painterly rendering with different stylization and abstraction.

scales can be well maintained and produce many specific NPR stylizations.

The rest of this paper is organized as follows: Section 2 reviews the related work. The preprocessing and hierarchy-building methods are presented in Section 3. The rendering algorithm and NPR abstraction are described in Section 4. The proposed system is evaluated and the experimental results are demonstrated in Section 5. The conclusion and future work are given in Section 6.

2 Related Work

In this section, the related works are surveyed. For other interesting works, see two excellent survey books [24], [25] or recent SIGGRAPH or NPAR conference proceedings. Haeberli proposes a pioneering stroke-based rendering scheme [26] to create painterly images using an ordered collection of strokes described by shape, size, color, and orientation. Hertzmann [4] presents a greedy algorithm that creates multiple sized brush strokes. This method imitates the procedure in which a painter draws a picture, forming the paintings from multiple layers using rough to detailed stroke sizes. The brush strokes in these two image-based works were placed in the screen space, which is sufficient for still images but may create a shower door effect or popping effect in animation. To solve this problem, Meier [1] associates the strokes with particles defined on the surface of the objects. Because the strokes are associated with the actual locations in space, they move smoothly across the screen in a consistent manner as the viewing parameters change. However, this technique cannot regulate the screen space stroke density. For example, as objects move away from the viewpoints, the particle density in the screen space increases. In this situation, too many strokes will create an obviously much darker image and may produce a completely different look for the image.

To address the frame-to-frame coherence issue, Cornish et al. [15] proposes a view-dependent particle, multiresolution hierarchy to adaptively change the particle density according to the distance between objects and the viewpoint. Because the particles generated and vertices in the original model are closely correlated, a less densely or irregularly sampled polygonal model does not guarantee having enough particle density for various viewing conditions. In addition, when the hierarchy levels change, a subset of particles are added or removed. This may still cause noticeable popping. To reduce this popping effect, Xu et al. [19] propose a continuous level of detailed structure, in which only one particle is added or removed when changing between two adjacent levels. Praun et al. [14] introduce the tonal art map (TAM) technique to maintain coherence across tones and scales for various lighting/viewing conditions. This approach combines hardware acceleration technology to achieve a real-time hatching rendering. The QSplat [20] rendering is originally designed for rendering point-based models that are organized as a bounding sphere hierarchy. The QSplat includes a culling and LOD metric to efficiently render point splats.

3 Multiscale Segmented Sphere Hierarchy

Previous NPR approaches address the importance of region structures [7], [23] such as edges and borders. In this section, we will present our data structure and algorithms to establish a multiscale segmented sphere hierarchy that can organize the input models into a meaningful region hierarchy according to model attribute similarities such as color and curvature. For this purpose, we extend a well-known image segmentation method called JSEG [21] to manipulate the 3D models. The JSEG method is excellent at segmenting images with fine-grained texture patterns.

3.1 Sampling Input Models

Our preprocessing algorithm begins with either a triangular mesh or a cloud of points. Instead of explicitly sampling the input mesh, QSplat takes the mesh vertices as the initial input points and generates a hierarchical bounding sphere structure from these points. When the mesh has triangles of various sizes, the overestimated bounding sphere may cause the traversal algorithm to descend the hierarchy unnecessarily. To avoid this problem, we adopted the resampling and relaxation approach of Turk et al. [27] to explicitly sample the input mesh. This approach allows us to explicitly determine the number of sampling points. We do not want too many or too few points/spheres in the hierarchy leaf nodes. To express various artistic effects, an artist must have control over how the painterly strokes are oriented. In this paper, the user-specified vector field, normal and curvature information [1], [15] from a 3D model can be used as orientation fields to guide stroke drawing. Sometimes the principal curvature direction is too noisy or

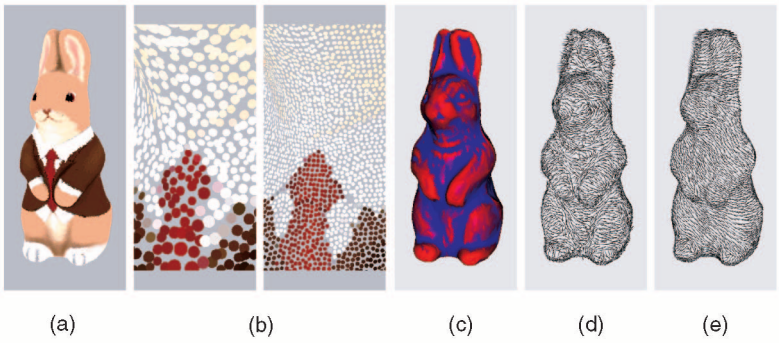


Fig. 2. (a) Input mesh model, (b) sphere distributions before and after resampling and relaxation, (c) curvature distribution coded in color from blue to red, (d) vector field using principal curvature direction, and (e) vector field after RBF smoothing. Note that in (b): Left, QSplat creates leaf spheres of various sizes, but in (b): right, the leaf sphere sizes are uniform after resampling and relaxation.

too detailed. We follow the method in [28] to select several feature points with larger curvatures and use their principal curvature directions to interpolate other nonfeature point directions using the radial basis function (RBF). In this manner, we can smooth the orientation of the vector fields on the input models. The input data resource can also be 2D images. We treat a 2D image as an array of 3D points on a plane, and then apply the proposed algorithm to it. In the current implementation, each sphere node contains several pieces of information, including the position, radius, normal, normal cone, color, curvature, TAM, and principal curvature direction. Fig. 2 shows an example sampling a bunny model using the proposed method. The methods used to build a tree to organize these newly sampled points (i.e., or called spheres) are presented next.

3.2 Building the Hierarchy

The leaf sphere sizes in the QSplat tree must be large enough to guarantee that no holes are left during rendering. Once we have assigned the leaf sphere sizes, the QSplat system exploits a *BuildTree* algorithm to develop the rest of the hierarchy. As the tree is built, attributes such as the normal and color at the interior nodes are set to the average of the attributes in the subtrees. In the *BuildTree* algorithm, different partitioning methods can generate different trees. The QSplat adopts a K-D-like tree-partition method. There are several other popular partitioning schemes available such as the Octree and covariance splitting [29]. These methods consider only the node spatial information. In our application, the leaf nodes have various colors. As we employ these partition methods directly to build QSplat trees, the model region boundaries could be ill-maintained. Later in this section, we will experimentally verify this claim. To maintain the color region boundaries, we propose a new QSplat-like bounding sphere tree that combines the advantages of covariance splitting method and a mature image segmentation method called JSEG. Compared to trees built by the previous methods mentioned above, the proposed tree produces a better hierarchy for abstract painterly rendering.

The original JSEG is a fully automatic color-based image segmentation method that identifies regions using color-texture patterns rather than only the color information. In practice, color image segmentation is not an easy task because of the nature of textured pattern. In the JSEG algorithm, image colors are first quantized into several representative colors to reduce the amount of color information. The quantized colors then become several class indexes for building a class map. The class map represents the property of the color-texture patterns. In our 3D extension, the local J is defined by following the JSEG algorithm formulation [21]. For each 3D node p, the local K-nearest neighboring node set K centered at p and the size of K is N including p. Let P_k be the position for a K-nearest neighboring node k and $k \in K$, and m be the mean position for the node set K,[1]

$$m = \frac{1}{N}\sum_{k\in K} P_k, S_T = \sum_{k\in K}\|P_k - m\|^2. \quad (1)$$

Suppose that all nodes are classified into C classes according to perceptual color quantization [33]. For each node p and its neighboring node set K, K is classified into the same C classes, K_i, where $i = 1, \cdots, C$. Let m_i be the mean of the N_i data points of class K_i,

$$m_i = \frac{1}{N_i}\sum_{k\in K_i} P_k \text{ and } S_W = \sum_{i=1}^{C}\sum_{k\in K_i}\|P_k - m_i\|^2. \quad (2)$$

The measured local J for p is then defined as

$$J = S_B/S_W = (S_T - S_W)/S_W. \quad (3)$$

In this manner, we can compute a local J for each node and obtain a 3D J-map for a given model. The higher the J value, the more likely the node is near the region boundaries [21]. The JSEG method ensures that the J value stored on a nonboundary node is close to 0. Once we obtain a J-map, we start to grow regions in this J-map from some seed nodes with lower J values. In our experiments, we always choose seed nodes whose J values are below the mean J multiplied by a small constant. Similarly, we terminate the growing region at an adjacent node when its J value reaches a user-selected bound. After this region has been grown, the oversegmented regions whose size or color similarity is below a threshold to the neighboring clusters are then merged, thus accomplishing segmentation. Fig. 3 shows an example of 3D segmentation. In a comparison study, we applied the JSEG software [30] to the rendered image in Fig. 3a and found that our 3D segmented result was very similar to the image's

1. Since K is not a large node set in our implementation, we use the mean position and Euclidean distance instead of surface center and geodesic distance in (1) and (2).

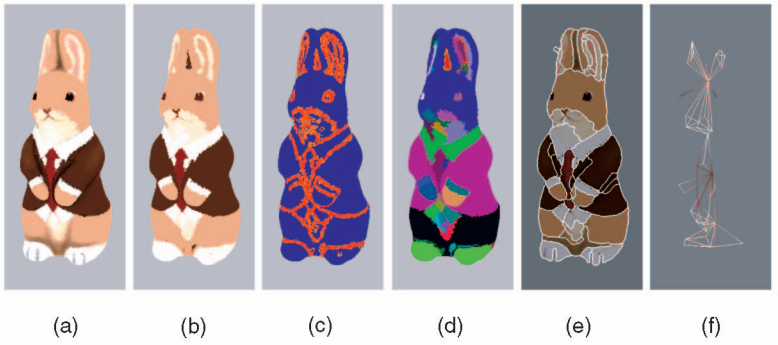


Fig. 3. (a) the original 3D color points, (b) results of color quantization, (c) a 3D J-image encoded by color from blue to red, (d) results after 3D segmentation, (e) using JSEG software [30], the segmented result on a 2D image for comparison study, and (f) a corresponding region adjacency graph for (d).

2D counterpart. Note that in 2D, we can see some over-segmented regions due to the 2D background color. In addition, in Fig. 3c, we can see the J value in the nonboundary region is close to 0, encoded as the color blue.

Once we obtain the segmented regions, the next step is using them to build a hierarchy. We can treat all regions as vertices of a graph. A region adjacency graph (RAG) is built to organize the regions. Fig. 3f shows an example of a region adjacency graph for a segmented input model. In our hierarchy, two neighboring regions are merged to form a new region according to the distance between their mean colors and their region sizes. For each connected region pair r_i and r_j, let the merge evaluation function be defined as:

$$e_{ij} = \begin{cases} color\_dis(r_i, r_j), & if \quad Size_{r_i} < \alpha \times Size_{mean} \\ & and \quad Size_{r_j} < \alpha \times Size_{mean}, \\ color\_dis(r_i, r_j) \times \varepsilon, & else. \end{cases} \tag{4}$$

In the above function, ε is a positive number near zero. The *color_dis* function will return the distance in the HSI color space between two mean colors in regions r_i and r_j, $\alpha = 0.3$ and $\varepsilon = 0.05$ in this paper. We evaluate the merge evaluation function and find a connected region pair that has the smallest merge evaluation value. This connected region pair is then merged into a bounding sphere node. Recursively, we proceed with this merge process and build a hierarchy from the regions to the root. In this manner, we make sure that each node's property is more homogenous than that of its parent node; any merge region pair is more homogenous in the color space than other regions. The above merging process is treated as a hierarchical region-based merging. Each region may cover a large area or contain several subregions of various curvature distributions; therefore, we will perform further partitioning within each region using a covariance splitting method [29]. This splitting method splits the inputs along the direction of greatest variations and sets user-specified thresholds for the curvature variations to hierarchically cluster the input data. The advantage of this method over other methods, like the K-D tree, is that it takes the model surface variations (i.e., curvature) into account. The pseudocode for building the proposed hierarchy is shown in Fig. 4.

Fig. 5 shows a comparison study of hierarchies built using different splitting methods. In this study, different levels for each hierarchy are shown and each node is drawn using its descendant leaf nodes to compare how well each tree can maintain the region boundaries at different levels. In Figs. 5a and 5b, both the K-D tree and covariance splitting methods produce a uniform blur effect with different Gaussian kernel sizes from the leaf nodes to the root. In this example, we can see that Fig. 5b is slightly better than Fig. 5a by taking curvature information into account. For example, look at the hands of the bunny; Fig. 5b's result is better than Fig. 5a's. In contrast, the proposed method maintains the region boundaries (see Fig. 5c) across different levels of the tree and obviously performs much better than both Fig. 5a and Fig. 5b.

In rendering NPR style images, the difference between each node and its child nodes is recorded in our hierarchy. The dissimilarity of a sphere node is defined as:

$$Node.S = \sum_{i=1}^{n} [|node.value - node.child[i].value| + (node.child[i].S)]. \tag{5}$$

This dissimilarity is an accumulated value from a nonleaf sphere node to its descendent leaf nodes. Several node values and dissimilarities are recorded in each node including the variance in color, curvature, bounding sphere coverage, the normal cone width, and J. In Section 4, we will show how the dissimilarity information is used to select

```
BuildRegionBasedTree(sphereNode[begin, end])
{
  regions = segmentation(sphereNode[begin, end]);
  for each region r_i in regions
    BuildTree( [begin , end ]); //* covariance splitting method
  Graph(V, E)=BuildRAG(regions);
  While(Graph(V).size != 1) {
    Calculate merge evaluation value e_ij for each connected
    region pair;
    Find the region pair r_i and r_j which has minimum e_ij ;
    BoundingSphereAndMergeSphere( r_i , r_j , Graph(V, E));
  }
  return Graph.v;
}
```

Fig. 4. The pseudocode for hierarchy building.

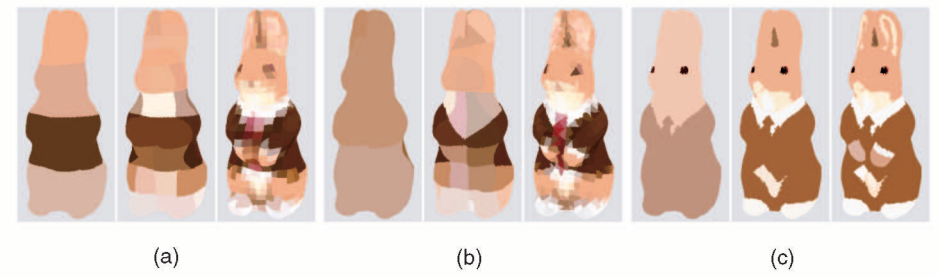


Fig. 5. A comparison study of hierarchies to show region boundaries of different levels. (a) K-D tree, (b) covariance splitting tree, and (c) the proposed tree.

appropriate nodes from the hierarchy to render NPR style images.

4 Rendering Algorithm

The proposed rendering is a two-pass method, including 1) the first pass renders objects into depth buffer for obtaining visibility information and 2) the second pass executes the actual rendering using visibility information. For the second pass, subject to several user-specified **abstraction rules** and **rendering options**, the proposed rendering algorithm selects suitable nodes from the proposed tree and stroke textures for display. The pseudocode in Fig. 6 is used for the second pass. This pseudocode consists of two procedures: 1) selecting candidate regions and 2) drawing suitable stroke nodes for display under the selected regions. These two procedures are subject to user-specified testing rules. Once we select some region nodes, we can draw its descendant nodes as stroke nodes using the region node's color, region borders, and its orientation vector field. Before we describe the details, a clear distinction has to be made between the region and stroke nodes, as illustrated in Fig. 7. In the following sections, we will describe how to control the region and stroke nodes selection in more detail.

```
TraverseHierarchyRegion(node)
{
  if (culling test)
    Skip this branch;
  else if(test region node abstraction rule or node is a leaf node) {
      TraverseHierarchyStroke (node);
    } else {
      for each child in children(node) {
        DrawRegionBorder(node);
        TraverseHierarchyRegion(child);
      }
   }
}
TraverseHierarchyStroke(node)
{
  if (test stroke node rule or node is a leafnode) {
    drawnode =SmoothDrawNode(node, parent_node);
    set NPR options;
    if silhouette (drawnode)
      DrawSilhouette(drawnode);
    else
      DrawSylizedSplat(drawnode);
  else {
    for each child in children(node) {
      TraverseHierarchyStroke(child);
    }
  }
}
```

Fig. 6. The rendering pseudocode in the second pass.

4.1 Selecting Candidate Region Nodes

Many 2D NPR methods such as [4], [22] propose some abstraction mechanisms for expressing original objects with minimal elements under a given threshold. Hertzmann [4] attempts to paint images using the minimal number of strokes under a color difference threshold. DeCarlo et al. [22] use the minimal regions with eye tracking information and color segmentation to paint images. In Section 3.2, we introduce a dissimilarity value S on each node in (5). This information indicates the accumulated difference in attributes from a node to its descendants. In *TraverseHierarchyRegion( )*, we can simply set our testing rule as S is less than a user-specified dissimilarity threshold. Under this selected attribute (i.e., color) dissimilarity threshold, the proposed method attempts to use minimal regions to express input models. With a larger S, the upper-level nodes (i.e., large regions) will be selected and the rough details will be used to paint images. This simple metric can be seen as a **region-based color abstraction** of models. For example, the

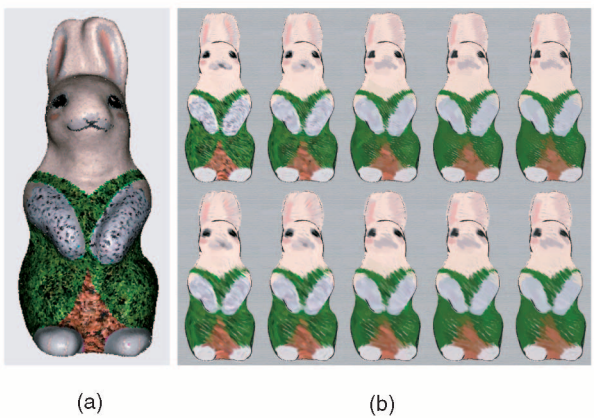


Fig. 7. (a) Another bunny model with more varied, fine-grained textures. (b) Each row of images from the left to the right represents different region nodes from the finer to coarse details. Each column of images from the top to the bottom represents different sizes of brush strokes drawn from small to large scale.

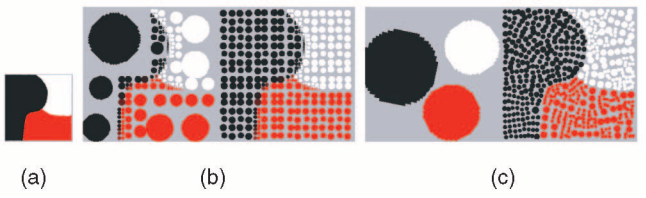


Fig. 8. The region boundary problem in hierarchy. (a) Input model, (b) K-D tree, and (c) the proposed tree. In (b) and (c), the left side shows the selected region nodes and the right side shows the stroke nodes. The (b) method requires deeper traversal of tree due to poor node clustering in color space.

selected S value is increasing for images from the left to the right in Fig. 7. Thus, the region details decrease from the left to the right, respectively. Once region nodes are determined, we draw their descendants as strokes in *TraverseHierarchyStroke( )*. Fig. 8 shows the advantage of the proposed tree. In this example, the K-D tree requires deeper tree traversal than the proposed method to accurately fit the boundaries. In other words, in terms of the number of minimal regions or computation efficiency, the proposed tree performs better than the K-D tree.

4.2 Selecting Candidate Stroke Nodes

Once we find the candidate region nodes in *TraverseHierarchyRegion( )*, we need to select their suitable descendant nodes, called stroke nodes, for display in *TraverseHierarchyStroke( )*. The stroke nodes are used to paint each region. The top of each column in Fig. 7 uses smaller strokes to paint each region than those used by the bottom column. In Sections 4.2.1 and 4.2.2, we propose two methods for choosing stroke nodes: 1) view-independent and 2) view-dependent stroke abstractions, respectively. These two methods represent two different views of NPR painterly rendering: One uses a fixed number of strokes to paint each region and the other uses strokes with a fixed size to paint on the canvas.

We use QSplat [20]'s approach to render a stroke node as a single texture-mapped quad sprite. In our implementation, the image texture is selected from one image from a set of TAMs [14] that consist of two-dimensional texture images. The user must provide a set of TAMs for the proposed system. In this set of TAMs, each texture image in the bottom row represents different tones for various lighting conditions. Each column image represents different scales for a tone image at the bottom. Each column image can be termed as a sequence of mipmaps of the same tone. Pruan et al. [14] do not apply TAMs to LOD geometry. The TAMs are only applied to the same geometry. Once the viewing conditions change (zoom-in or zoom-out), this technique automatically selects one image from a column of images using the MIPMAP scheme.

4.2.1 View Independent Stroke Abstractions

For view-independent method, we simply choose stroke nodes (i.e., bounding spheres) using the rule:

$$\boldsymbol{Radius}(node) \leq \boldsymbol{StrokeRadiusThreshold}. \quad (6)$$

In rendering, once a stroke node is selected, it will not be changed (i.e., the geometry is fixed at some sphere hierarchy levels) for various viewing conditions. Therefore, we term this the "view-independent" approach. To ensure coherent tones and screen density in the strokes for various lighting/viewing conditions, we assign a set of two-dimensional TAMs to each stroke node in our implementation. Depending on the current lighting/viewing conditions, we choose an image from the assigned two-dimensional TAMs to draw a stroke node. This method does not have a popping problem because the strokes are adhered to the models. The disadvantage of this approach is that it cannot dynamically switch sphere hierarchy levels to gain computation efficiency for various viewing conditions. We propose another method called the "view-dependent" approach in Section 4.2.2 to fix this drawback. The "view-independent" approach is more flexible than the approach by Pruan et al. [14], although the geometry is fixed for both methods. Our "view-independent" approach allows users to flexibly select some fixed sphere hierarchy levels before rendering instead of always selecting the same nodes.

4.2.2 View Dependent Stroke Abstractions

The second approach involves simply choosing the stroke nodes using the rule:

$$\boldsymbol{SplatScreenSize}(node) \leq \boldsymbol{StrokeSizeThreshold}, \quad (7)$$

where $\boldsymbol{SplatScreenSize}(node)$ is the projected size of the splat (i.e., sphere) on the screen. In this manner, the size of the strokes painted in each region is fixed and stroke selection is view-dependent. As the view changes, the hierarchy is traversed to select a set of stroke nodes to ensure the stroke density. In the view-dependent approach, we assign only the bottom row of TAMs instead of two-dimensional TAMs to each stroke node for various lighting conditions. Note that TAMs were not applied to the geometry LOD in [14]. We cannot simply assign each TAM column to the corresponding sphere hierarchy level. As in Cornish et al.'s work [15], for various viewing conditions, a subset of nodes are added (zoom-in) or removed (zoom-out) to maintain the stroke density in the proposed sphere hierarchy. Both [15] and our method will cause popping. We propose a smooth level transition method in Section 4.2.3 to solve the popping problem. This smooth transition efficiently renders the hierarchy, maintains the stroke density, and avoids popping in various lighting/viewing conditions. The view-dependent approach requires only a row of TAMs rather than the entire set of TAMs required by the view-independent method. Therefore, the view-dependent method is more attractive.

4.2.3 Smooth-Level Transitions

The QSplat [20] does not incorporate an algorithm for smooth transitions as sections of the model changes from one level of detail to another. Figs. 9a and 9b show the tree-level transitions. In Fig. 9a, once the level changes, child nodes varying from 2 to 4 are added or removed. As seen in the supplementary video clip, this may cause noticeable popping like [15]. This is because the QSplat tree adopts a discrete set of levels rather than a continuous LOD structure. As suggested by [20], geomorphs in Hoppe's progressive mesh system [31] can be used to potentially solve or reduce this problem. For

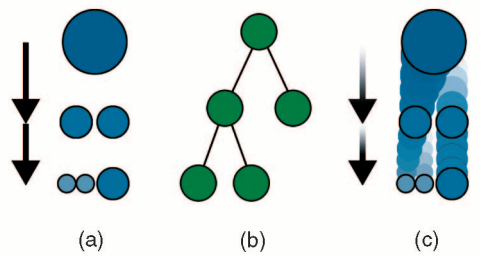


Fig. 9. (a) The discrete-level transitions like [15], (b) the corresponding hierarchy, and (c) the proposed continuous-level transition.

example, to generate continuous LOD, the continuous nodes are generated by interpolation, including color, position, and size, between the parent and child nodes. This naive geomorph implementation slightly improves this problem. Furthermore, from our practice, this approach still has obvious artifacts because of the overlapped strokes from these interpolated nodes. In Section 3.1, our models are uniformly sampled and most sibling nodes are well spaced (i.e., no overestimated bounding spheres) in the tree. When these interpolated nodes are progressively transiting toward their parent node, overlapping among these interpolated nodes increases. Therefore, in particular for semitransparent stroke patterns, the tone from these nodes becomes much darker. Popping is still noticeable in the animation sequence. We adopt a novel variant of the above geomorph implementation to compromise between the popping and tone problems. First, the intermediate nodes are interpolated using child and parent nodes. Next, we select a child node, denoted G, whose position is nearest to the parent and its radius size is the largest among the child nodes. Excluding this G node, we linearly interpolate an **alpha** blending value using *ChildSplatStrokeSize* and *ParentSplatStrokeSize* for each intermediate stroke node whose splat size is *StrokeSizeThreshold* that is between *ChildSplatStrokeSize* and *ParentSplatStrokeSize*. Therefore, except for the G node, the other interpolated nodes with an alpha value become more transparent as they approach their parent node. This allows avoiding sudden changes between two LOD levels. The idea is illustrated in Fig. 9c. In contrast to other methods, this new approach performs surprisingly well and can successfully eliminate most popping occurring in other methods. See the supplementary video clip for a comparison among different methods.

4.3 Silhouettes, Region Borders, and Directional Strokes

The silhouette is an important feature in NPR. In *TraverseHierarchyStroke( )*, if a node meets silhouette condition: $\boldsymbol{normal}(node) \cdot \boldsymbol{view\_vector} = 0 = \boldsymbol{cos}(90°)$, it is drawn as a silhouette rather than a stroke on the canvas. Because the model is a discrete set of nodes rather than a continuous node, the silhouette condition is slightly modified as: $\boldsymbol{normal}(node) \cdot \boldsymbol{view\_vector} < \boldsymbol{cos}(90° \pm \boldsymbol{ThreasholdDegree})$, where the *ThreasholdDegree* is used to control the silhouette width (see an example in Figs. 13c and 13d). The region borders are also important features that express objects. In Section 3.2, we have a region adjacency graph (RAG) to organize segmented regions from the JSEG algorithm. In *TraverseHierarchyStroke( )*, we use leaf nodes to draw region borders instead of region candidate nodes. Subject to a user-specified threshold, our system can draw different levels of region borders from RAG to express objects. Fig. 10 shows an example illustration of various levels of region borders. The strokes can be oriented along some vector fields. Figs. 10c and 10d show an example of stroke orientations controlled using the stroke's node vector fields and region node vector fields. The choice can be subject to the user's specification. In practice, to enhance the region effect, we suggest using the region node orientation. In actual paintings, brush strokes have the same direction in one segmented area, but the stroke direction may change in the image border edge areas. Similarly, we can use either the stroke node color or region color (i.e., mean color in the region) to draw stroke nodes. In addition, Perlin's Noise [32] value can be added to slightly modify either the stroke direction or color to create stylized strokes with some randomness to imitate the drawing styles created by artists These NPR options can be specified in *TraverseHierarchyStroke( )*.

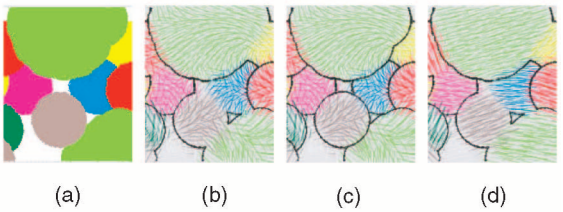


Fig. 10. (a) Original input with color. In (b) and (d), we show region borders with coarse to fine details. This example is inspired by [22]. (c) Stroke orientation controlled by stroke node vector fields. (d) The orientation is controlled by region node vector field. To emphasize the regularity in each region, case (d) seems to be better than (c) as suggested by many previous works [7], [23].

5 Nonphotorealistic Rendering Styles

5.1 Results

This section shows images rendered by several very different NPR styles. We also describe how each effect is produced. A video of all examples in this paper can be found at http://couger.csie.ncku.edu.tw/~vr/NPR/Demo.htm. For a given bunny model, Figs. 11a, 11b, and 11c show different image styles including watercolor, hatching, and half toning. These three image styles are rendered in a single layer using the proposed method with different TAM stroke textures. The input data are originally a cloud of points rather than a

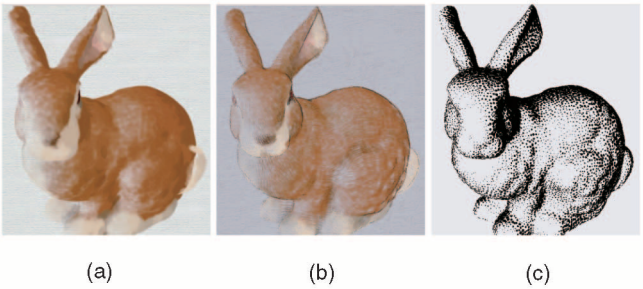


Fig. 11. Illustrations of the same model with different NPR styles: (a) watercolor, (b) hatching, and (c) half toning.

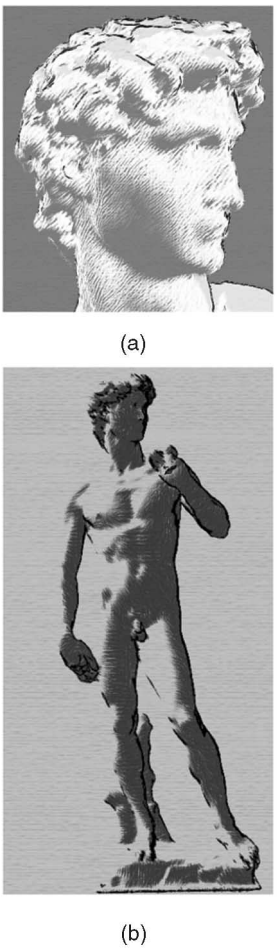


Fig. 12. NPR rendered results of David statue using the proposed system.

polygon model. In this example, we use [29] to simplify this point model first before building the tree. To calculate the curvature of the point surface models, we measure the estimated surface variation [29] using PCA on the neighborhood covariance matrix. Fig. 12 shows another NPR result for rendering a well-known point-based model, the David statue. We render a simplified resolution of the original David model here due to the current system memory limit.

The proposed system can imitate paintings with multiple brush strokes of multiple sizes in [4]. At runtime, we can make a simple classification to identify if a leaf node represents a finer or coarser detail. In Figs. 13a and 13b, curvature and color gradient information are used to classify the leaf nodes into two classes: coarse (Fig. 13a) and finer (Fig. 13b) details. A large brushstroke size is then chosen to draw (Fig. 13a) to obtain an image. A small brushstroke size is then used to draw (Fig. 13b) to obtain another image. These two images are then blended to create the final image (Fig. 13e). For this example, the proposed rendering method is run twice to obtain these two layers. Note that we also show the TAMs used in Fig. 13a and Fig. 13b for various lighting conditions. Fig. 14 shows another example using the same process.

Fig. 15 shows another two-layer painting example inspired by [4] and [22]. The image is painted using constant color regions. Fig. 15a shows the source image. Fig. 15b is an image painted from the first layer using region borders. Fig. 15b is produced using only borders where the adjacent color gradients are larger than a user-specified threshold. The image in Fig. 15c is blended with the second layer painted with different oriented strokes and borders using color gradients. Fig. 15d shows another two-layer painted image with more abstraction (i.e., the finer details are removed). This example demonstrates the flexibility of the proposed method. The user can freely adjust different options and parameters to vary the style of painting in a very intuitive manner.

Fig. 16 shows an example of 3D Chinese landscape painting created by the proposed method. In Chinese landscape painting, TSUN stroke techniques are used to express the rock textures [6].

It will be interesting to give the system a hint to create an abstracted image that has details in only some the places as shown in Fig. 17c. For example, the user can paint a gray-level image called a local importance map, as shown in Fig. 17b, to guide the local abstraction. The proposed traversal method is then changed to traverse down to find the stroke nodes first and then traverse up to find a suitable region node that fits the local abstract value specified by the

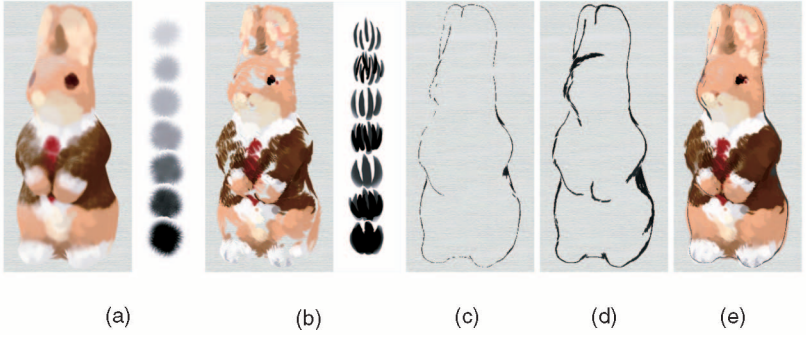


Fig. 13. Painterly image with multiple brush strokes of multiple sizes. Width varying silhouettes: (c) 10 degree and (d) 20 degree for ThresholdDegree.

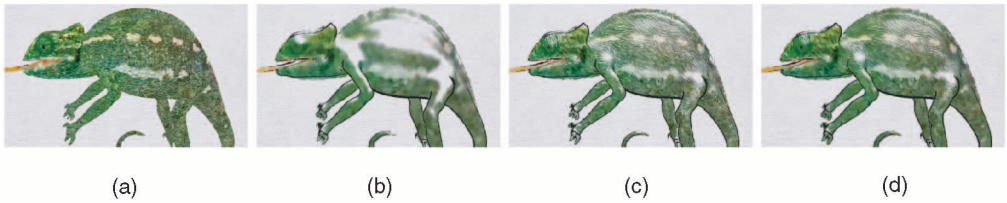


Fig. 14. Another example with multiple brush strokes of multiple sizes. (a) Original chameleon model, (b) the coarse details with large brush strokes, (c) the finer details with small brush strokes, and (d) the blended results using (b) and (c).

local importance map. Fig. 18 shows another example of a point-based model called the octopus.

In Table 1, we demonstrate the rendering efficiency of our proposed rendering system for the illustrations in this section. We also show stroke size ($N \times N$) in pixel units for each example. We implemented our system using OpenGL on the GeForceFX 5900Ultra graphics card. Our test platform is a Pentium 4 2.4G with 512 MB of main memory, running the MS windows 2000 system. All images were rendered at 640×480 resolution. The obtained frame rate was the average interaction performance with the models. There was not much performance difference between rendering different image styles in a single layer of the proposed method. If multilayer rendering was required to generate a series of layers, the rendering performance was slightly degraded due to overhead in each layer. In addition, some options such as region borders and silhouette drawings increased the rendering time. At present, the major bottleneck in the rendering performance is the speed required to access the depth buffer for splat visibility testing. This access time ranges from 0.007 to 0.009 seconds. In addition, in all examples in this section, the time for required to obtain the visibility information ranged from 0.005 (Fig. 10d) to 0.021 (Fig. 18) seconds. This cost depends on the number of nodes rendered into the Z-buffer. For example, in Fig. 18, 19,353 nodes were rendered into the Z-buffer.

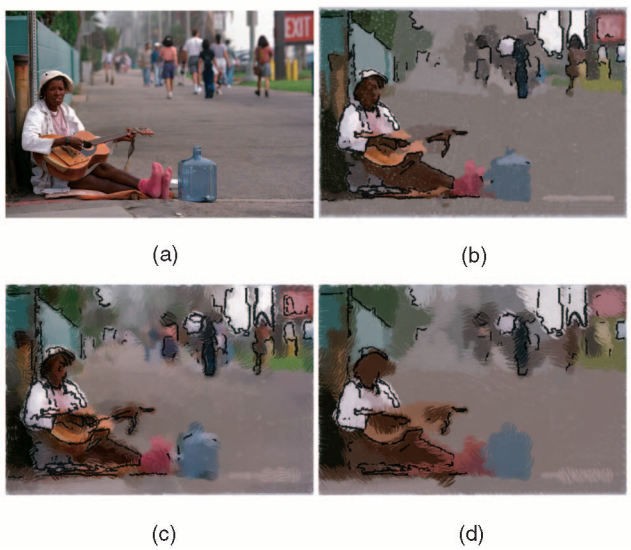


Fig. 15. Another example of a two-layer painting subject to different NPR options and user-specified parameters including region borders, stroke sizes, and patterns. (Photo courtesy http://philip.greenspun.com).

5.2 User Controls and System Limitations

The main parameters provided to users for generating NPR results and the limitations of the proposed system are addressed next. In preprocessing, the proposed system provides users with three main parameters to control the JSEG segmentation: 1) the threshold for color quantization, 2) the neighborhood size for local J value, and 3) the color similarity threshold for region merge to reduce over-segmented regions. In the tree traversal, the system requires that the users provide *DissimilartyThreshold* (i.e., selecting region nodes) and *StrokeRadiusThreshold* or *StrokeSizeThreshold* (i.e., selecting brush stroke nodes) parameters to control the NPR abstraction degree. Users also need to provide a set of prerendered TAMs to draw stroke nodes and other optional NPR parameters to control the silhouette condition, region borders, stroke directions, and so on.

No single segmentation method can perform well in all cases. We used the JSEG method for segmentation in the proposed system. The JSEG method has the advantage of correctly segmenting images with fine-grained texture patterns in which the texture colors do not vary significantly. From our experience, if colors or texture patterns vary significantly in a region, this region will be potentially oversegmented. In Fig. 19, both the input source images, Fig. 19a and Fig. 19b, consist of three distinct regions from visual observation. Using JSEG, we can successfully segment Fig. 19a into three regions, but the method fails for Fig. 19b because the colors or textured patterns vary too much. In the future, we will investigate other segmentation schemes to enhance the proposed system.

6 Conclusions and Future Work

A novel system framework for three-dimensional NPR stylized and abstract painterly rendering is proposed. The proposed framework successfully generates many interesting stylized images and additional artistic styles can be developed in the future. In contrast to other previous 3D NPR works, the proposed tree is built using color and geometric information. Therefore, the region borders can be well abstracted from coarse to fine scales.

Several future works can be performed and described as follows. For example, we will investigate better segmentation methods to enhance our system. We may consider geometric feature boundaries like [34] when building the hierarchy. The proposed system requires user-specified parameters to adjust the rendering styles. Each style can be

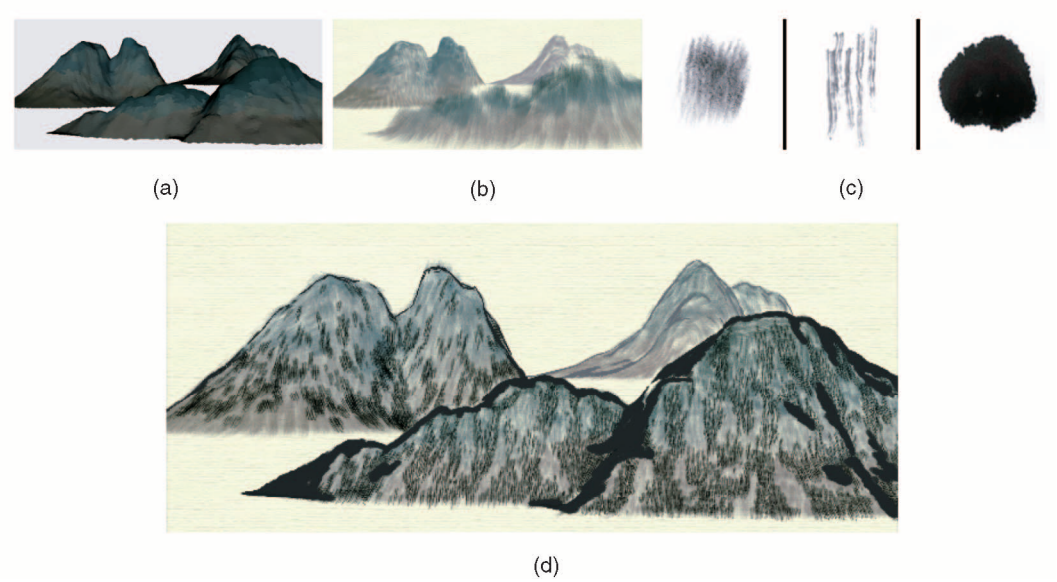


Fig. 16. Simulated TSUN techniques in Chinese landscape painting. (a) Original models, (b) the first layer, (c) a set of TAM strokes, and (d) the final results (the first + second layers + silhouette).

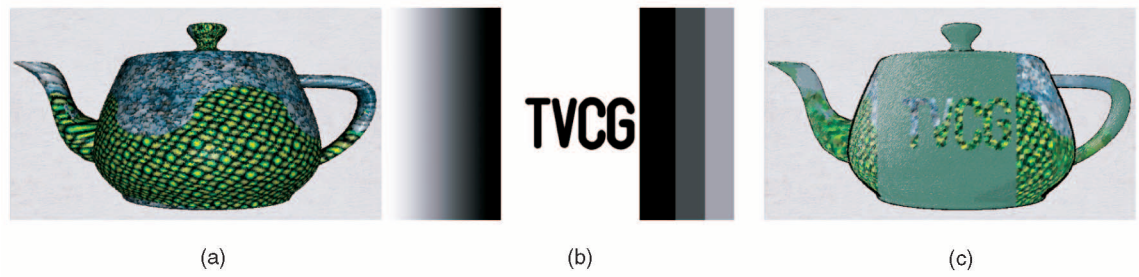


Fig. 17. (a) Original model. In (c), the details are controlled by a gray-level image (b).

set using an intuitive set of parameters determined by users. In the future, we hope to include some automatic learning and style transfer techniques such as "Image analogies" [5] to reduce the manual effort or to guide users in setting parameters. To express models with deformation such as [35], [36] in stylized and abstract manners is another challenging topic. Some extra work in the proposed system will be done in the near future such as painting long strokes along the principle curvature direction over surfaces. Borders rendered with better line-drawing styles will also be included.

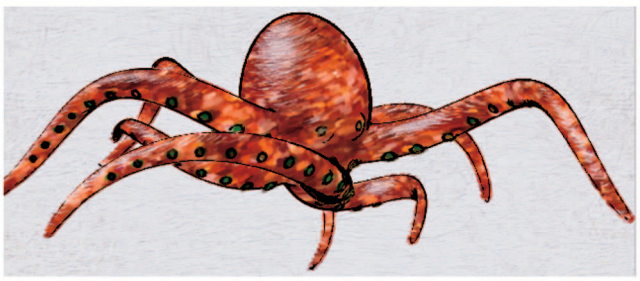


Fig. 18. Octopus point model rendered in a NPR style.

ACKNOWLEDGMENTS

The authors give their sincere thanks to anonymous reviewers' helpful comments to improve their paper. They thank the Stanford Computer Graphics Laboratory for the bunny, David model, and QSplat point rendering software source code, and the ETH Zurich Computer Graphics Laboratory for the Octopus and Chameleon models. They appreciate the painting and creation for models from Chinng-Fu Chen and Hsun-Jen Chen at thier graphics group/visual system laboratory. This project is supported by the National Science Council under contract No. NSC93-2213-E-006-060, NSC-93-2213-E-006-026, and NSC-92-2213-E-006-067.

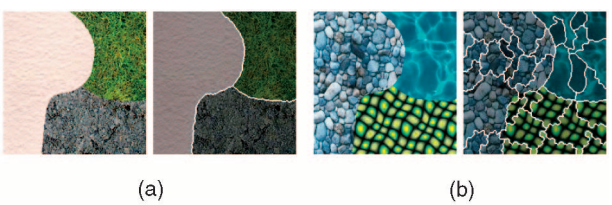


Fig. 19. Segmented results using JSEG. In contrast to (a), (b) seems to be oversegmented.

TABLE 1
Rendering Performance

Figure	Model leaf nodes	Layer 1 (nodes / stroke size)	Layer 2	Layer 3 silhouette /Region *b*order	Total nodes	Fps
Fig 7b: top	64,184	10,365(10x10)	/	3,246/*s*	13,611	19
Fig 7b: bottom	64,184	6,684(20x20)	/	3,246/*s*	6,684	31
Fig. 10d	57,600	385(35x35)	/	2,337/*b*	2,722	69.6
Fig. 11	51,808	2,343(26x26)	/	/	2,343	54
Fig. 12a	273,097	14,557(10x10)	/	1,717/*s*	16,274	19.7
Fig. 12b	273,097	44,760(5x5)	/	3,238/*s*	47,998	12.8
Fig. 13	64,184	1,065(29x29)	1,145(20x20)	2,944/*s*	5,154	32
Fig. 14d	101,685	12,290(14x14)	18,231(9x9)	4,019/*s*	34,540	6.9
Fig. 15b	85,920	11,838(14x14)	/	4,283/*b*	16,121	8
Fig. 15c	85,920	2,923(31x31)	2,150(25x25)	5,802/*b*	10,875	9
Fig. 15d	85,920	2,923(31x31)	2,150(25x25)	3,007/*b*	8,080	10
Fig. 16	164,745/86,745/144,365	1,359	2,691	13,860/*s*	17,910	10.1
Fig. 17	111,394	21,027(7x7)	/	7,653/*s*	28,680	30.3
Fig. 18	465,878	23,045 (8x8)	/	19,458/*s*	42,503	6

Note we separately show leaf nodes for the three mountain models used in Fig. 16.

REFERENCES

[1] B.J. Meier, "Painterly Rendering for Animation," *Proc. SIGGRAPH '96,* pp. 477-484, 1996.

[2] C.J. Curtis, S.E. Anderson, J.E. Seims, K.W. Fleischer, and D.H. Salesin, "Computer-Generated Watercolor," *Proc. SIGGRAPH '97,* pp. 421-430, 1997.

[3] E. Akelman, "Implicit Surface Painting," *Proc. Implicit Surfaces Conf.,* pp. 63-68, 1998.

[4] A. Hertzmann, "Painterly Rendering with Curved Brush Strokes of Multiple Sizes," *Proc. SIGGRAPH '98,* pp. 453-460, 1998.

[5] A. Hertzmann, C.E. Jacobs, N. Oliver, B. Curless, and D.H. Salesin, "Image Analogies," *Proc. SIGGRAPH '01,* pp. 327-340, 2001.

[6] D.-L. Way and Z.-C. Shih, "The Synthesis of Rock Textures in Chinese Landscape Painting," *Computer Graphics Forum,* vol. 20, no. 3, 2001.

[7] B. Wang, W. Wang, H. Yang, and J. Sun, "Efficient Example-Based Painting and Synthesis of 2D Directional Texture," *IEEE Trans. Visualization and Computer Graphics,* pp. 266-277, 2004.

[8] M. Salisbury, C. Anderson, D. Lischinski, and D.H. Salesin, "Scale-Dependent Reproduction of Pen-and-Ink Illustrations," *Proc. SIGGRAPH '96,* pp. 461-468, 1996.

[9] O. Deussen and T. Strothotte, "Computer-Generated Pen-and-Ink Illustration of Trees," *Proc. SIGGRAPH '00,* pp. 13-18, 2003.

[10] A. Lu, C.J. Morris, J. Taylor, D.S. Ebert, C. Hansen, P. Rheingans, and M. Hartner, "Illustrative Interactive Stipple Rendering," *IEEE Trans. Visualization and Computer Graphics,* vol. 9, no. 2, pp. 127-138, Apr.-June 2003.

[11] O.M. Pastor, B. Freudenberg, and T. Strothotte, "Real-Time Animated Stippling," *IEEE Computer Graphics and Applications,* vol. 23, no. 4, pp. 62-68, July/Aug. 2003.

[12] L. Markosian, M.A. Kowalski, D. Goldstein, S.J. Trychin, J.F. Hughes, and L.D. Bourdev, "Real-Time Nonphotorealistic Rendering," *Proc. SIGGRAPH '97,* pp. 415-420, 1997.

[13] A. Lake, C. Marshall, M. Harris, and M. Blackstein, "Stylized Rendering Techniques for Scalable Real-Time 3D Animation," *Proc. Int'l Symp. Nonphotorealistic Animation and Rendering '00,* pp. 13-20, 2000.

[14] E. Praun, H. Hoppe, M. Webb, and A. Finkelstein, "Real-Time Hatching," *Proc. SIGGRAPH '01,* p. 581, 2001.

[15] D. Cornish, A. Rowan, and D. Luebke, "View-Dependent Particles for Interactive Non-Photorealistic Rendering," *Proc. Graphics Interface Conf. (GRIN '01),* pp. 151-158, 2001.

[16] R.D. Kalnins, L. Markosian, B.J. Meier, M.A. Kowalski, J.C. Lee, P.L. Davidson, M. Webb, J.F. Hughes, and A. Finkelstein, "WYSIWYG NPR: Drawing Strokes Directly on 3D Models," *Proc. SIGGRAPH '02,* pp. 755-762, 2002.

[17] M.A. Kowalski, L. Markosian, J.D. Northrup, L. Bourdev, R. Barzel, L.S. Holden, and J.F. Hughes, "Art-Based Rendering of Fur, Grass, and Trees," *Proc. SIGGRAPH '99,* pp. 433-438, 1999.

[18] M.X. Nguyen, H. Xu, X. Yuan, and B. Chen, "INSPIRE: An Interactive Image Assisted Non-Photorealistic Rendering System," *Proc. 11th Pacific Conf. Computer Graphics and Applications (PG '03),* pp. 472-476, Oct. 2003.

[19] H. Xu and B. Chen, "Stylized Rendering of 3D Scanned Real World Environments," *Proc. Int'l Symp. Nonphotorealistic Animation and Rendering '04,* pp. 25-34, 2004.

[20] S. Rusinkiewicz and M. Levoy, "Qsplat: A Multiresolution Point Rendering System for Large Meshes," *Proc. SIGGRAPH '00,* pp. 343-352, 2000.

[21] Y. Deng, B. Manjunath, and H. Shin, "Color Image Segmentation," *Proc. IEEE Conf. Computer Vision and Pattern Recognition '99,* pp. 446-451, June 1999.

[22] D. DeCarlo and A. Santella, "Stylization and Abstraction of Photographs," *Proc. SIGGRAPH '02,* pp. 769-776, 2002.

[23] B. Gooch, G. Coombe, and P. Shirley, "Artistic Vision: Painterly Rendering Using Computer Vision Technologies," *Proc. Int'l Symp. Nonphotorealistic Animation and Rendering,* 2002.

[24] B. Gooch and A. Ashurst Gooch, *Non-Photorealistic Rendering.* A.K. Peters Ltd., 2001.

[25] T. Strothotte and S. Schlechtweg, *Non-Photorealistic Computer Graphics. Modelling, Rendering, and Animation.* Morgan Kaufmann Publishers, 2002.

[26] P. Haeberli, "Paint by Numbers: Abstract Image Representations," *Proc. SIGGRAPH '90,* pp. 207-214, 1990.

[27] G. Turk, "Re-Tiling Polygonal Surfaces," *Proc. SIGGRAPH '92,* pp. 55-64, 1992.

[28] J. Hays and I. Essa, "Image and Video Based Painterly Animation," *Proc. Int'l Symp. Nonphotorealistic Animation and Rendering,* 2004.

[29] M. Pauly, M. Gross, and L.P. Kobbelt, "Efficient Simplification of Point-Sampled Surfaces," *Proc. IEEE Visualization Conf. '02,* pp. 163-170, 2002.

[30] JSEG_Software: http://vision.ece.ucsb.edu/segmentation/jseg/software/, 2005.

[31] H. Hoppe, "Progressive Meshes," *Proc. SIGGRAPH '96,* pp. 99-108, 1996.

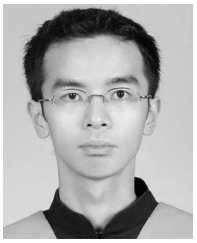
[32] K. Perlin, "An Image Synthesizer," *Proc. SIGGRAPH '85,* pp. 287-296, 1985.

[33] Y. Deng, C. Kenney, M.S. Moore, and B.S. Manjunath, "Peer Group Filtering and Perceptual Color Image Quantization," *Proc. Int'l Symp. Circuits and Systems,* 1999.

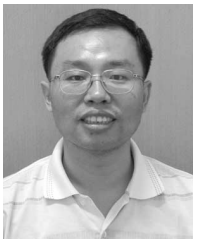
[34] D. Cohen-Steiner, P. Alliez, and M. Desbrun, "Variational Shape Approximation," *ACM Trans. Graphics,* vol. 23, no. 3, pp. 905-914, 2004.

[35] C.-H. Lin and T.-Y. Lee, "Metamorphosis of 3D Polyhedral Models Using Progressive Connectivity Transformations," *IEEE Trans. Visualization and Computer Graphics,* vol. 11, no. 1, pp. 2-12, Jan./Feb. 2005.

[36] T.-Y. Lee and P.-H. Hung, "Fast and Intuitive Metamorphosis of 3D Polyhedral Models Using SMCC Mesh Merging Scheme," *IEEE Trans. Visualization and Computer Graphics,* vol. 9, no. 1, pp. 85-98, Jan.-Mar. 2003.



Ming-Te Chi received the BS degree in geography from National Taiwan University, Taipei, Taiwan, in 2000 and the MS degree from the Department of Computer Science and Information Engineering, National Cheng Kuang University, Tainan, Taiwan, in 2003. He is currently working toward the PhD degree in the Department of Computer Science and Information Engineering, National Cheng-Kung University. His research interests include computer graphics and nonphotorealistic rendering.



Tong-Yee Lee received the BS degree in computer engineering from Tatung Institute of Technology in Taipei, Taiwan, in 1988, the MS degree in computer engineering from National Taiwan University in 1990, and the PhD degree in computer engineering from Washington State University, Pullman, in May 1995. Now, he is a professor in the Department of Computer Science and Information Engineering at National Cheng-Kung University in Tainan, Taiwan, Republic of China. He has served as a guest associate editor for *IEEE Transactions on Information Technology in Biomedicine* from 2000 to 2005. His current research interests include computer graphics, nonphotorealistic rendering, image-based rendering, visualization, virtual reality, surgical simulation, and distributed and collaborative virtual environment. He leads the Computer Graphics Group/Visual System Lab at National Cheng-Kung University (http://couger.csie.ncku.edu.tw/~vr). He is a member of the IEEE.