Real-time Painterly Rendering of Terrains

Shiben Bhattacharjee
shiben@research.iiit.ac.in

P. J. Narayanan
pjn@iiit.ac.in

Center for Visual Information Technology
International Institute of Information Technology, Hyderabad

Abstract

We present a non-photo realistic, real-time painterly rendering technique for terrains. The painterly appearance and the impression of terrains is created by effectively rendering several brush strokes. The strokes have fixed locations on the surfaces of the terrain during animation to enable frame to frame coherency. The strokes are rendered as alpha blended sprites in two-dimensions and are oriented along the slope of terrain analogous to the way artists paint on canvas. By exploiting the regular nature of terrain data, we create pre-decided rendering depth orders for primitives for any camera orientation. With this, we avoid the necessity of sorting the primitives of sprites required for alpha blending. We use DirectX10/SM4.0 based shaders to render strokes to improve performance. Being distributed on terrain, strokes get cluttered when they are closely located on screen. We follow a level of detail scheme that maintains a uniform stroke density in screen space. Various styles can be achieved with different stroke variations. Phong shading the rendered output in real-time is possible for more varied styles. We achieve painterly rendering in real-time with a combination of object space positioning and image space rendering of strokes. We illustrate our method with images and performance results.

1. Introduction

The intentions of an artist come out as the aesthetics and expressiveness of the painting. The accurate rendering done by computers fails to provide images with a such a feeling. Animations are therefore often created by artists by painting a number of frames and is a tedious job. Computers have been used over the years to generate the surrounding environments of the main characters. This reduces the artist's effort, but leads to a visual disparity between the hand drawn objects and the environment. Painterly rendering, a non-photo-realistic rendering technique, can bring artistic abstraction to the rendering and thus mix the computer generated scenes with the hand drawn elements. Therefore, painterly rendering has attracted the attention of graphics researchers. Creating abstraction of landscape and terrains seems an interesting problem since they are common in artistic creations and animations. Painterly rendering technique for general polygons exists. These cannot be applied directly to terrains because of level of detail complexities and richness due to long view range. An optimal composition of terrain rendering methods and painterly rendering is essential for real-time performance and high quality output.

The regular nature of the terrain data make them a specific type of model. We exploit this special nature of terrains to provide efficient painterly rendering for them. A technique to order the triangles of a terrain from back to the front is at the heart of this. We achieve an fps of 120 on Puget Sound terrain data on the Nvidia 8800GTX GPU. In this paper we present a real-time painterly rendering technique to make abstractions of terrains. We also emphasise our results with post processing for varied stylizations.

The organization of the paper is as follows: We describe related work in the next section (section 2). A brief overview of the system is mentioned in section 3. In section 4 we show the representation of terrain data and stroke textures. Here we also explain view frustum culling and level of detail management. Section 5 shows the method in which we are ordering the strokes in back to front order. Technique for rendering the strokes is mentioned in section 6. Illustrations and the performance of our system are discussed in section 7. We conclude with a discussion on technical aspects and aesthetic considerations with some future works in section 8.

2. Related Work

Abstract representation of still images was introduced by Haeberli [5] using image color gradient and user interactivity for painting. Hertzmann [8] places curved brush strokes of multiple sizes on images for painterly rendering. The

technique fills color by using big strokes in the middle of a region and uses progressively smaller strokes as one approaches the edges of the region. Shiraishi and Yamaguchi [19] improves the performance of above method by approximating the continuous strokes by placement of rectangular strokes discreetly along the edges to create painterly appearance. Santella and DeCarlo[18] uses eye tracking data to get points of focus on images and create painterly rendering with focus information. All these techniques work well on single images but involve iterative techniques that make them cumbersome for real-time applications [10]. Also if they are applied on each frame of an animation independently, it can lead to a flickering of strokes due to incoherence of strokes between frames. Painterly rendering has been tried and made coherent on videos as well [11], [7], but these techniques are not well suited for 3D rendering.

Painterly rendering for animation was introduced by Meier [17]. She eliminated shower door effect and achieved frame to frame coherence by rendering several brush strokes whose positions stick the 3D model's surfaces. However, view dependent sorting of these strokes is required for alpha compositing, making it unsuitable for real-time animations. Recent work [6, 1] describe a real-time painterly process inspired by Meier using programmable graphics hardware. They render the polygonal model first and store the depth map. A second pass uses the depth map to remove occluded strokes so that the strokes/billboards can be rendered in any order. For a complex and distant scene, such as a terrain, the inaccuracies due to precision in the depth map and comparison at boundaries can reduce the visual quality. Terrains are rich models containing many samples and should be rendered with large view distances. Other modes of NPR have been created in past for terrains. Pen and Ink approaches [12, 3] exist which mostly focus on silhouette of the terrain. These are, however, different than painterly rendering process.

Partitioning the terrain into fixed size square patches at different resolutions is gaining popularity due to fast hardware. The tiled structures provide compact representation and easy rendering [20, 4]. Losasso and Hoppe introduced geometry clip-maps, a multi-resolution, fixed memory size scheme for efficient representation and rendering of large terrains [15]. While other terrain rendering schemes could also have been followed, we choose the tile based representation since it promises better regular spacings in screen space between samples which will be used as stroke positions while rendering. We built our painterly rendering system over the terrain rendering system explained in section 4 which can achieve 150 fps with an average rate of 84 million triangles per second and a highest of 200 million triangles per second on current GPUs.

3. Overview of our Approach

Terrains are heavy objects, often involving millions of triangles in each frame. Conventional two-pass painterly rendering techniques will be inefficient for them. We combine painterly rendering with terrain rendering optimally for real-time performance. We treat each height in the elevation map of the terrain as a stroke's location in the 3D world. Fixing the positions of strokes in 3D keeps them coherent between frames while animations [17]. The point location is projected on 2D screen using projection transformation and a rectangular stroke is rendered at that location, orientated along the projected slope of the terrain (see Figure 1). Real-time performance is obtained using the following.

1. Only the strokes of the visible part of terrain are rendered for efficiency. This is achieved with a view frustum culling algorithm.

2. The strokes are rendered in a back to front order for alpha compositing. We exploit the special property of terrain representation to obtain the back to front ordering in one pass. This is explained in Section 5.

3. The level of detail of the terrain is changed smoothly with distance from the viewpoint. This avoids the problem of strokes getting cluttered at far distances, which can be visually distracting. Level of detail also reduces the rendering load.

The whole terrain is kept in the CPU memory. A section of it needed for rendering is cached on the GPU memory as elevation maps. Corresponding section of color texture, normal map, and the slope map are also stored in the GPU's texture memory. The terrains are cached in terms of 1024×1024 blocks and are rendered in terms of 64×64 tiles. The tile is the basic unit for rendering, view frustum culling, and LoD management.

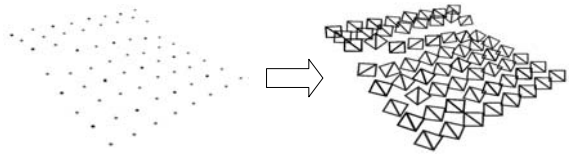

Figure 1. Each height in the height-map is converted into a rectangle which is oriented along the terrain's slope at that point. An 8×8 grid is shown as example.

Each stroke is sent by the CPU as a single point primitive as a geometry template, which gets converted into a rectangle on which a stroke texture is mapped. This is accomplished with DirectX10/SM4.0 based shaders explained in

section 6. Each point on the terrain is rendered as a stroke. The stroke is aligned in the direction of the slope at the 3D terrain point to imitate how artists draw such scenes. We render the strokes in the back-to-front order by exploiting the regular grid structure of tiled terrains. Points of a tile can be scanned and rendered as strokes in the back-to-front order, based on the view orientation. Eight such orderings are sufficient to handle any view orientation. The tiles that survive frustum culling are also rendered in the same order to provide a back to front ordering for the entire terrain without sorting. This procedure enables us to render large terrains at frame rates of 120 and above in the painterly style.

4. Terrain Representation

Our base terrains are 2D grids of heights with a fixed post-distance in the X and Y directions.Our focus is on painterly rendering of the terrain at real-time rates without the CPU, the GPU, or the bandwidth between them becoming the bottleneck. The available terrain data is loaded in the CPU memory and a contiguous window of the terrain is kept in the video RAM of the GPU based on the current viewpoint.

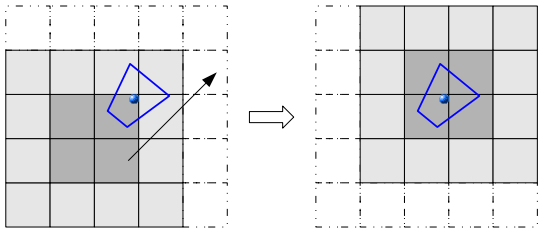

Figure 2. Reference point is at the center of ground-plane projection of the view frustum (marked as blue). Reference point is kept within the 2×2 blocks. As it goes out it is re-centered. The figure assumes 4×4 cache size.

4.1. Representation of data

Terrains are divided into fixed memory-size *blocks*, each of which is divided into a number of *tiles*. A tile is the basic rendering unit for the CPU. Currently, blocks are of size 1024×1024 and tiles of size 64×64. These blocks are loaded as textures on the GPU memory. We maintain a *GPU cache* consisting of $N \times N$ blocks which gets updated periodically to hold all the data needed for rendering. We try to keep the GPU cache symmetric with respect to the projection of the view frustum on an average "ground" plane. We do that with the use of a *reference point* which is kept close to the center of the GPU cache (Figure 2). We use the center of ground-plane image of the view frustum as the reference point currently. This ensures fixed in memory representation for the terrain.

If the reference point goes beyond the central 2×2 block of the GPU cache, the cache is re-centered by bringing another row or column of blocks (Figure 2). Since the cache is maintained in memory as an array of texture ids, re-centering involves downloading a few blocks to the GPU and adjusting pointers on the CPU. The data transfer time is kept small using a job-queuing scheme. The blocks to be brought in the GPU cache are not done at once, but done successively in following frames to avoid possible jerks. The basic terrain system is able to render large, CPU resident terrains at above 100 fps along with the cache updating in the background.

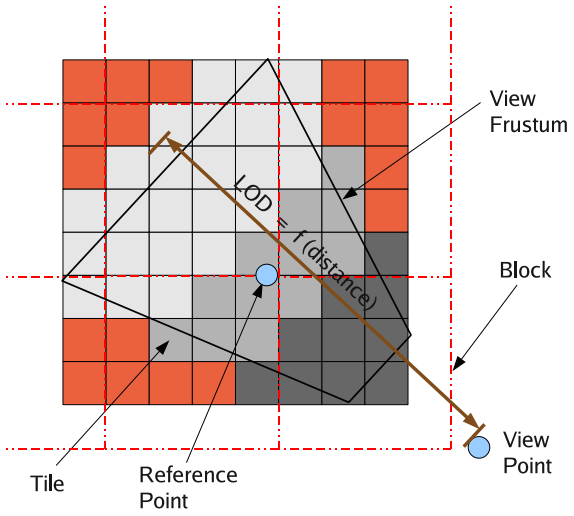

Figure 3. Tiles outside view frustum (marked red) are eliminated. Tiles totally inside (grey shaded) are rendered with strokes at each of its sample's locations. LODs of tiles to be rendered and the blending factor is calculated as a function of distance. Fewer strokes are drawn for a lower LOD tile.

4.2. Level of Detail

The view frustum culling algorithm treats each tile as basic units. The bounding sphere of tiles are tested against the six planes of the frustum. On the basis of this, tiles are marked to be inside or totally outside the frustum, and are assigned with a *LOD* number. LODs (Levels of detail) for a tile include different resolutions of an area on the ground.

A particular LOD of a tile can be computed by dropping alternate samples from the better LOD available. Highest LOD for a tile contains all the samples. We calculate the rendering LOD of a tile using its distance from the viewpoint (Figure 3). Farther the distance, lower the LOD. LOD l becomes a function of distance d as the integer part of $l = \log(1 + d/d_t)$, where d_t is a pre-decided LOD transition distance. When the LOD of a tile changes from one to another, many samples/strokes may pop up suddenly. For this, we morph the tile from one LOD to other by fading the alternative strokes away as they go out and vice versa. The fractional part of l is used as the *morphing factor* and is multiplied to the opacity of alternative strokes in the vertex shader. While Wagner [20] uses the morphing factor to geomorph two different heights at that same location, we use it to fade in or fade out the strokes which are coming in and going out respectively, giving a smooth transition without popping artifacts.

5. Back-to-Front Stroke Ordering

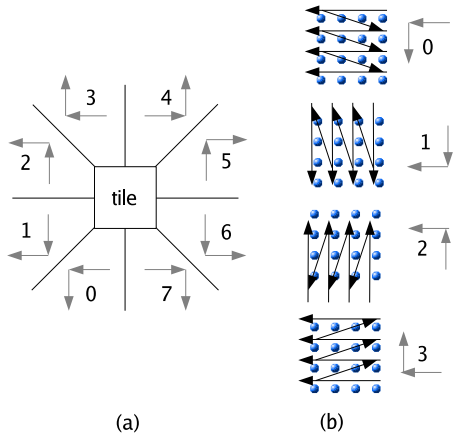

Figure 4. (a) A tile can be viewed from many yaw directions, but only eight zones are sufficient for a back to front ordering of samples in it. (b) Four possible arrangements of samples for some ranges shown in (a); Other ranges can be handled in the similar way.

A back-to-front ordering of samples/strokes of the terrain is at the heart of our algorithm. We discretize the camera yaw into 8 zones of each $45\,\mathrm{deg}$ each shown in Figure 4(a). Each zone corresponds to a particular order of scanning the heights for guaranteed back-to-front ordering of triangles. The 8 zones have unique ordering, four of which are shown in Figure 4(b). The same scan order

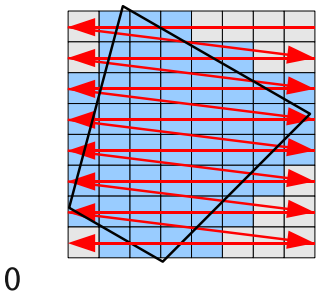

Figure 5. View frustum culling algorithm testing tiles in a specific order depending upon the camera's orientation. Here zone 0 is shown. Such eight orders of testing are possible as explained in Figure 4.

applies to the tiles inside the view frustum as seen in Figure 5. In practice, we switch the ordering a little while after the viewpoint is into the new zone to avoid unnecessary toggling of the ordering at the boundaries between zones. Tiles are rendered as *VBOs* (vertex buffer objects) for good performance. A single VBO can render any tile, as other parameters like tile's world origin, blending factor etc. is packed up in texture coordinates. For a given range of orientation of the camera, an ordering is fixed. Thus each zone corresponds to a unique VBO.

The same order is used to scan the tiles for view frustum culling. Figure 5 shows one out of eight of the possibilities for tile scanning shown in Figure 4(a). All the tiles farther from the camera get rendered before the nearer ones. Because of this, all the strokes in the screen in that view become ordered from back to front without the cumbersome need of sorting. This method is similar to shear-warp volume rendering [14] in which axis aligned 2D slices of volume are rendered off-screen, and are stacked into the desired orientation and scale to display the 3D volume. We are handling 2D surfaces, our method only decides the order of samples and does not suffer from any less accurate sampling problem that [14] faces.

6. Stroke Rendering

We send points to the graphics pipeline for each stroke to be rendered. Vertex shader computes the exact world location of the stroke at this point. It also calculates the color from the texture and normal map of the terrain with other lighting information (the unified architecture of latest GPUs allow fast texture access from any shader [2]). The alpha of the point is changed according to the morphing factor decided for that tile from the CPU. The vertex shader forwards

these things to the pipeline.

Geometry Shader of the GPU can generate primitives [2]. It converts the single point primitive sent from the CPU into a rectangle for the brush sprite (Figure 6). The perspective division of the graphics pipeline makes the strokes smaller when they go farther, while painterly rendering needs constant sized strokes. To compensate for this process, the vertices are multiplied with the w value (the perspective scale factor) before rasterization. This reverses the division (Haller and Sperl [6]) and the strokes always maintain the same size on the screen. This process can lead to holes in the surface if the camera goes very close to the ground for a given point density. We disable the multiplication at such distances when the strokes start to lose density.

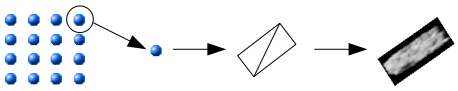

Figure 6. Overview of rendering of stroke. Each vertex from the VBO gets converted into a rectangle which is mapped with a stroke texture.

The generated rectangle is subsequently oriented in screen space along the slope of the terrain at that location since artists tend to place their strokes along the slopes of mountains running down to the valleys. We pre-compute a *slope-map* that gives the direction of maximum gradient at every point in the terrain (Figure 7). Slope-map stores the gradient vector in the world space, which is accessed by the Geometry Shader for every sample, is transformed to camera coordinates and to the image space to get the stroke orientation. The fragment shader accesses the stroke texture, and modulates its color with the color coming in from the pipeline. Alpha blending happens between these rendered strokes so that they mix among themselves for a smooth output. The outline of the whole method is described in Algorithm 1.

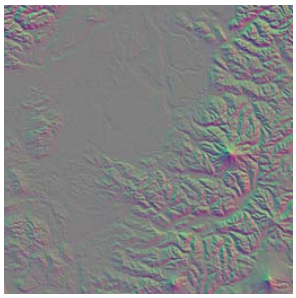

Figure 7. Slope-map, Puget Sound dataset

Algorithm 1 Painterly Rendering of Terrains

Load stroke textures st
Load terrain height H, color C, normal N, slope S map
Create 8 VBOs for different camera yaw-ranges
for each frame **do**
  Update GPU Cache if necessary (Section 4.1)
  Find zone q based on the yaw-range of the camera
  Perform VFC and LOD assignment based on q.
  for each tile **do**
    Send VBO[q]
    Vertex Shader: Calculate color using lighting $c = f(C, N)$. Calculate position p using height H
    Geometry Shader: Generate a quad at p, orient along slope S, assign color c
    Frag. Shader: Output color $c_o = mix(c, c_{st})$. At a different render target, output color as normal of stroke texture N_{st}
  end for
  Phong shade the output using the normal map
end for

For more stylizations, we render the normal maps of these strokes separately as well. We do this with multiple render targets supported by modern GPUs. In a different pass, these two outputs are treated as a texture and its normal map respectively, and are mapped on a screen aligned quad. With the help of the normal map, the scene can be Phong shaded with a varying lighting source (Figure 8). This process is inspired by [9] but we do it in real-time on rendered outputs harnessing the power of modern GPUs.

7. Results

We built our system and experimented on a Intel Pentium Core 2 Duo E6400 as the CPU and an NVIDIA 8800GT as the GPU. We used the OpenGL 2.1 graphics library and GLSL 1.20 shaders. We chose different screen resolutions to render upon for speed of alpha blending is screen size dependent. Performance is dependent on stroke size as well. We choose an optimal stroke size; Small enough to give good performance but not as small to leave holes in the terrain. With a resolution of 1024×768, we got seamless performance with an average triangle rate of 40 million triangles per second (Figure 9). With a resolution of 1280×1024 we get 35 million triangles per second and 120 fps (average). Traditional two pass painterly rendering technique (with depth map computed in the first pass) had half the performance of our system. We did our experiments on Puget Sound terrain data available from Georgia Tech website. Blue marble data set was also included in our experiments. We used some real satellite terrain data-set and some synthetically created ones as well. We show the

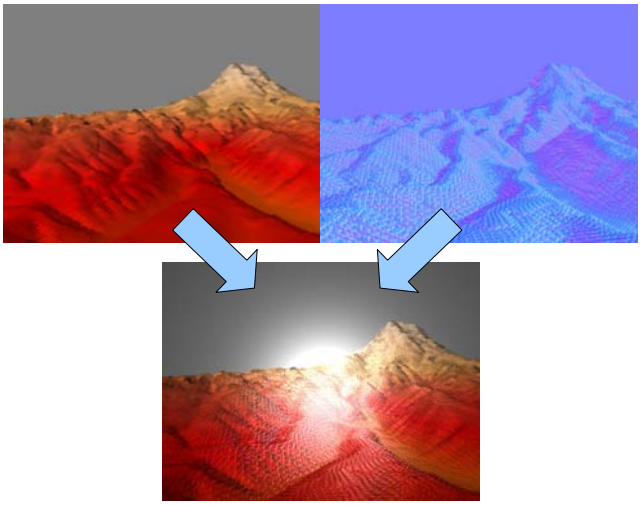

Figure 8. The color output and the normal map output of the scene are used to Phong shade on top of it to stylize it. The effect is that of shining a spotlight on the painting. The normal map is contrast stretched here for visibility.

effects of different stroke directions, with along the slope direction. In Figure 12 and 10(top-right), the strokes are oriented along a perpendicular direction to the XY projection of the normal vector. This simulates the effect of strokes flowing over the ridges instead of along the slopes. An artist drawing with strokes of fixed orientation is shown in Figure 10(bottom-left). Effect of adding small randomness to orientations is shown in Figure 10(top-left). Figure 10(bottom-right) shows the use of a small brush with sharp strokes. The accompanying video contains painterly walk-through on Puget Sound data. Some of our results are shown in Figure 11, 13, 14, 16, 17, 15.

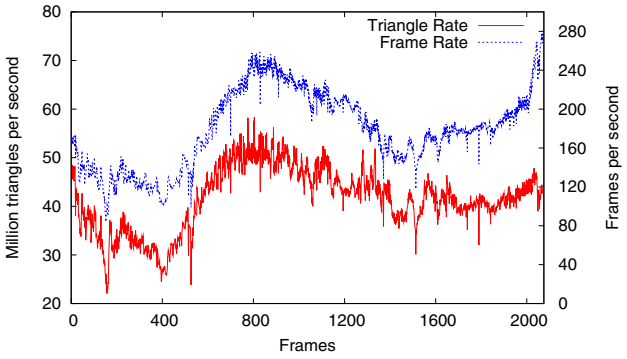

Figure 9. Walk through over the terrain

8. Conclusion

We presented a real-time painterly rendering technique for terrains. We get nice visuals with frame to frame coherence on animation of the scene. Considering rich nature of terrains and cumbersome nature of painterly rendering processes, we get good performance with our system using latest graphics hardware. Our system being single pass only, is faster than traditional painterly rendering techniques involving two passes. With varied stroke textures, and orientations of strokes, different artistic styles can be achieved with variety of taste. In future, we wish to render terrains with procedural stroke textures similar to geo-graftals mentioned in [16] and [13] to create even varied visuals and improve performance by optimizing the techniques specifically for terrains.

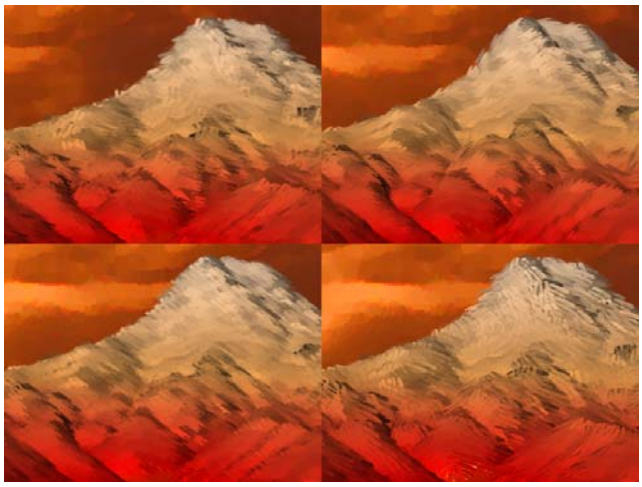

Figure 10. (top-left) Strokes placed along slope with some perturbations in orientation. (top-right) Strokes placed along the perpendicular to the normal. (bottom-left) Strokes placed with a fixed orientation. (bottom-right) A sharp stroke texture. Sky is a pre-painted texture.

Acknowledgments: We gratefully acknowledge the partial financial support of Microsoft Research's Virtual Earth Programme.

References

[1] S. Bhattacharjee and N. Adabala. Texture guided real-time painterly rendering of geometric models. In *5th Indian Conference, ICVGIP 2006*, pages 311–320. LNCS 4338, 2006.

[2] D. Blythe. The Direct3D 10 system. In *SIGGRAPH '06: ACM Trans. Graph.*, pages 724–734, 2006.

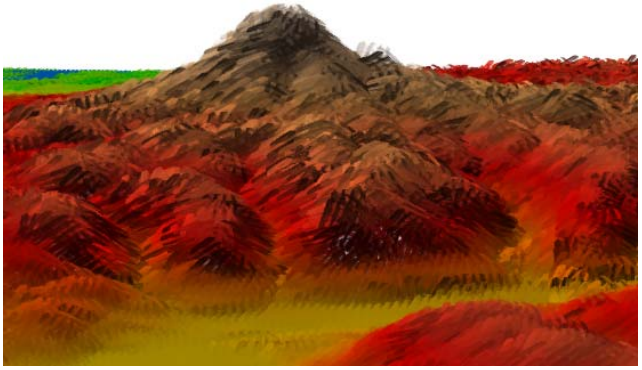

Figure 11. Distant view of Mount Rainer

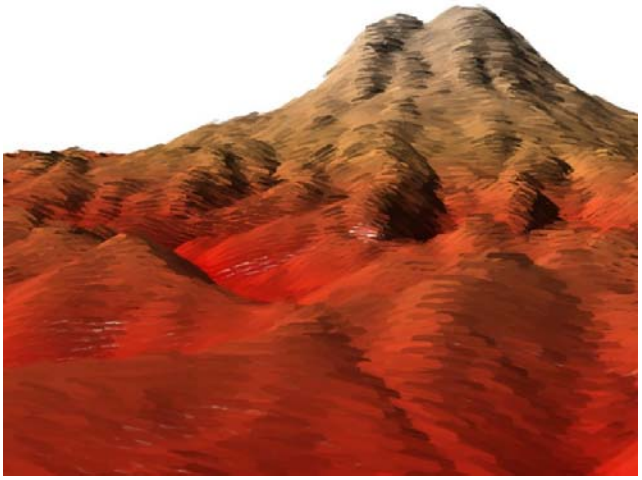

Figure 12. Strokes running along perpendicular to normals.

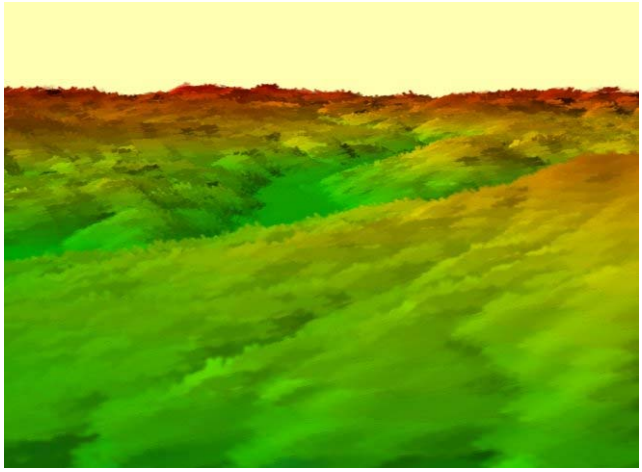

Figure 13. A region in Puget Sound with low variations.

[3] L. Coconu, O. Deussen, and H.-C. Hege. Real-time pen-and-ink illustration of landscapes. In *NPAR '06: Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, pages 27–35, 2006.

[4] S. Deb, S. Bhattacharjee, S. Patidar, and P. J. Narayanan. Real-time streaming and rendering of terrains. In *ICVGIP '06*, pages 276–288. LNCS 4338, 2006.

[5] P. Haeberli. Paint by numbers: abstract image representations. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 207–214, 1990.

[6] M. Haller and D. Sperl. Real-time painterly rendering for m.r. applications. In *GRAPHITE '04: Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 30–38, 2004.

[7] J. Hays and I. Essa. Image and video based painterly animation. In *NPAR '04: Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, pages 113–120, 2004.

[8] A. Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. *Computer Graphics*, 32(Annual Conference Series):453–460, 1998.

[9] A. Hertzmann. Fast paint texture. In *NPAR '02: Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, 2002.

[10] A. Hertzmann. Tutorial: A survey of stroke-based rendering. *IEEE Comput. Graph. Appl.*, 23(4):70–81, 2003.

[11] A. Hertzmann and K. Perlin. Painterly rendering for video and interaction. In *NPAR '00: Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, pages 7–12, 2000.

[12] M. V. J C Whelan. Formulated silhouettes for sketching terrain. In *Proceedings of Theory and Practice of Computer Graphics 2003*, pages 90–97, Birmingham, UK, 2003.

[13] M. Kaplan, B. Gooch, and E. Cohen. Interactive artistic rendering. In *NPAR '00: Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, pages 67–74, 2000.

[14] P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 451–458, 1994.

[15] F. Losasso and H. Hoppe. Geometry clipmaps: terrain rendering using nested regular grids. *ACM Trans. Graph.*, 23(3):769–776, 2004.

[16] L. Markosian, B. J. Meier, M. A. Kowalski, L. S. Holden, J. D. Northrup, and J. F. Hughes. Art-based rendering with continuous levels of detail. In *NPAR '00: Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, pages 59–66, 2000.

[17] B. J. Meier. Painterly rendering for animation. *Computer Graphics*, 30(Annual Conference Series):477–484, 1996.

[18] A. Santella and D. DeCarlo. Abstracted painterly renderings using eye-tracking data. In *NPAR '02: Proceedings of*

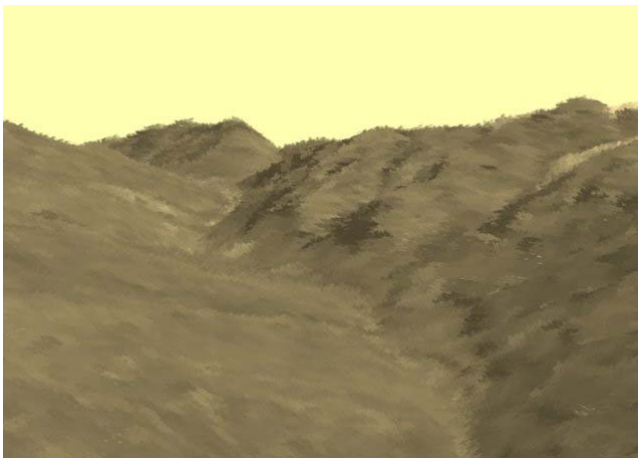

Figure 14. A real textured dataset rendered in a painterly style.

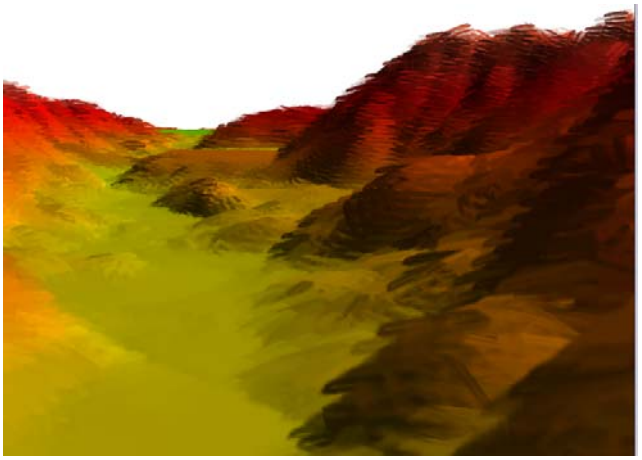

Figure 15. A valley region

the 2nd international symposium on Non-photorealistic animation and rendering, 2002.

[19] M. Shiraishi and Y. Yamaguchi. An algorithm for automatic painterly rendering based on local source image approximation. In *NPAR '00: Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, pages 53–58, 2000.

[20] D. Wagner. Terrain geomorphing in the vertex shader. *ShaderX2, Shader Programming Tips and Tricks with DirectX 9, Wordware Publishing*, 2004.

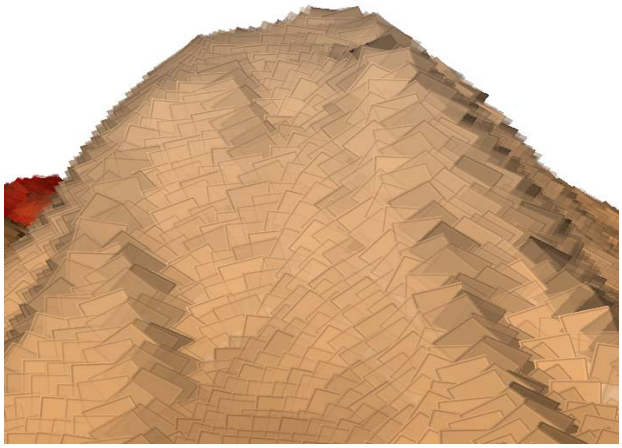

Figure 16. Simple rectangles are used instead of proper brush strokes to illustrate the flow of strokes along a hill

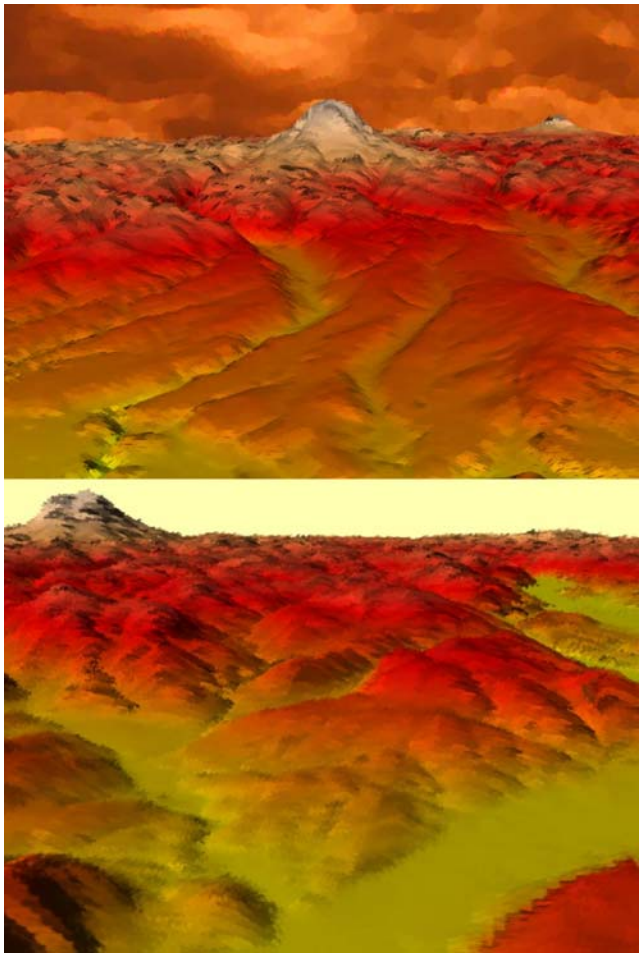

Figure 17. Mountains and valleys in Puget Sound painterly rendered.