Project aEmber : Real-time Painterly Rendering of 3-d Scenery

UIUC ACM SIGGraph
Parashar Krishnamachari

Development Team
Shanon Drone, Parashar Krishnamachari, Dmitry Lapan, Josh Michaels, Don Schmidt, Phil Smith.

Artists / Modelers
Michael Bach, Nasri Hajj

Abstract

Throughout the history of computer graphics, the holy grail has been photorealism. While we accept that it is numerically impossible, the idea is that a reasonably good approximation is achievable such that we can trick the human eye. Over the years, methods have progressed from Lambertian shading to Phong Shading, to Radiosity, and the more recent work on the various Monte Carlo methods for global illumination. But even more recently, there has been study in the opposite direction from photorealistic rendering. In short, non-photorealistic rendering treats computer graphics as a medium for art. Thus, we try to develop means by which to render a scene in a stylized artistic fashion.

Project aEmber was an effort to render 3-d scenery with the appearance of being hand-painted. aEmber has gone through 2 major revisions in its implementation since its original conception. The final algorithm is based on the one described by Meier[1] designed for use in Disney's Tarzan. Code and executables for aEmber 3 Final are available at
http://www.acm.uiuc.edu/siggraph/

Among the concerns for the development of a real-time painterly rendering engine, were speed, interframe coherency, screen coverage, quality, and artistic control. Each such concern was answered by various approaches over the course of its revisions and ultimately, we reached what we believed to be a reasonable algorithm for real-time display of painted 3-d scenery.

CR Categories

I.3.3 [Computer Graphics] Picture/Image Generation – Display Algorithms; J.5 [Computer Applications] Arts and Humanities – Fine Arts.

Keywords

Non-photorealistic rendering, painterly rendering, real-time, interactive.

Introduction

Prior to actually creating a 3-d painting, an ordinary 3-d scene is made in whatever tool the artist finds suitable. Currently, aEmber only supports loading of Wavefront .OBJ scene data in addition to its native .AE3D files.

Properties of the scene including object materials and lights are taken to wit. However, aEmber does ignore texture data in general. Vertex painted color information is stored in the source file as vertex color info rather than texture reference info, and therefore, will be preserved in the aEmber view window. Skybox textures will be ignored. However, aEmber has its own skybox support, so the images used in the original scene skybox can be placed in the "envmap" subdirectory for the aEmber viewer to look up.

aEmber uses a unique hierarchical scene graph description that allows a user to have control over the scene at various depths. In the process of developing a painted 3d scene, we use material properties for a given object to define color and lighting characteristics. Brushstrokes, however, must be assigned by the artist. The hierarchical scene graph allows an artist to generalize by assigning a brush shape at any particular level of the hierarchy. The entire scene can use a single brushstroke shape or one can assign different strokes to different objects. It is even possible to assign specific brushstrokes to individual points. Moreover, lower-level hierarchical assignments automatically override those at higher level. For instance, we can assign a particular brushstroke to an object, but also assign a different brush to a specific polygon or a specific point within the object.

At the highest level, there is the scene itself, which has a default brush. Beneath that, are objects, which can have their own separate brush assignments.

Further one step down from objects, there are polygons (generally triangles). The triangles are tessellated to varying degrees (which is another aspect controllable by the user at various hierarchy levels) to result in a point cloud. It is at these points that a brushstroke is actually rendered.

As the rendering pipeline recurses down the tree, we set the current brush to the scene default. As the loop works deeper own the tree, the brush setting is changed if a new brush assignment is found anywhere along the line. When a point/brush has been drawn, we reset the brushstroke to its prior setting based on the triangle to which it belongs. While this is a recursive process, the actual implementation is not recursive, else a stack overflow is inevitable. We simply simulate a recursion by making use of the fact that the hierarchy of the scene graph is limited in its depth.

Brush definitions themselves are small grayscale TGA image files. The grayscale data is treated as alpha-channel information for the renderer so that natural translucence of brushstrokes can be simulated in texture space. These files sit in a brushes subdirectory and are referenced by number in the scene data.

Brush images, in general need only be simplified iterations of strokes as there are other variables controllable by the user/artist.

Brushstroke size and direction are variables that are defined in much the same way as brush choice. In addition, the artist can enable mild jittering to be applied on top of the given definitions (jittering is also applicable to color). In the final version, there was also the option of allowing contour-controlled stroke directions so that the brushstrokes might appear to follow the curvature of an object in the scene. Direction definitions that are contour-controlled simply aligned the polygon renderings of brushes to the planes in which the existing polygon lied (alignment by vertex normal).

Interframe Coherency

The main advantage of 3-d scenery as opposed to 2-d images is the freedom of the

viewer to move about in the scene as if it were actual space. The fact that we are rendering 3-d scenery in a painted fashion implies that a viewer would wish to move about in the scene and that the painted view would be dynamic. There exist several 2-d image filtering approaches that can make an individual frame of animation appear painted, but the caveat of these approaches is that the painting is not coherent from frame to frame.

Maintaining inter-frame coherency was one of the major concerns in the first version of aEmber. The locations of brushstrokes was not decided based on the image, but by the 3d scenery. We would simply tessellate each triangle to a cloud of points at which textured quads of the brushstrokes were drawn. Making use of the consumer 3d hardware to render was key in maintaining the speed such that we could render painted scenery at multiple frames per second. With compiler optimizations, aEmber v1 was able to maintain 14 fps on an average scene of 25,000 brushstrokes (not necessarily all brushstrokes are within the FOV at once) on a PII-450 machine with a 3dfx Voodoo3 video chipset. Unfortunate side effects of this approach included the perspective-induced brush scaling. A hand-painted portrait generally has only a few scales of stroke sizes that do not scale quite so regularly. In the images rendered by aEmber version 1, brushstrokes on distant objects were smaller than those of nearby objects, and the scaling of those strokes progressed in a true-to-perspective fashion. Not only is this not a faithful representation of paintings, but it also decreases overall coverage of the screen, so there were always “holes” in the paintings.

While orthographic projections could cure this issue of brushstroke scaling, it creates a different problem in the fact that the perspective of the scene itself will be skewed. The resulting image is not very convincingly 3-dimensional. While the viewer is free to move about in the scene, the perception of vanishing points and solid objects is no longer present.

Screen Coverage

aEmber v1 had issues with coverage of the screen. There were areas where brushstrokes did not exist as well as areas within the model itself where brushstrokes did not completely fill the model in screen space. If we were to increase the size of the brushstrokes, or if we were to increase the tessellation depth that decided stroke placement, that could solve the problem. However, both approaches put a strain on the hardware. Brushstrokes are rendered as alpha-blended textured polygons so as to harbor some natural translucence. Alpha-blending of textures over several layers is a major drain even on modern 3d hardware. Increasing the area over which that drain occurs does not help either. Increasing the tessellation depth is equally if not more draining as it introduces more points to transform and more brushstrokes to render.

Figure 1

Figure 1 above shows a screenshot from aEmber v1. We can see that the brushstrokes are scaling very smoothly due to perspective. Also, the coverage of the screen is not very complete. Filling up the screen had to be faked by using a background texture or by using a skybox.

In aEmber v2, we applied brushstrokes based on a rendered image of the scene. The scene was first rendered using ordinary D3D Retained mode lighting. A sampling of pixels from the rendered image was used to generate color information for the brushes. The result filled the screen completely, and also performed

well. However, this method lacked the frame-to-frame coherency of aEmber v1. Frames adjacent in the time axis would often be so radically varying that it is no better than using a simple image filtering approach to stylized rendering.

Flexibility

Another problem with aEmber v2 is that by using the image alone, we have no information about the objects. Color information does not signify an object because multiple objects can have the same color. If every object had a discernibly unique color, we wouldn't be concerned. However, with lighting taken into account, there's also the possibility of highlights all of the same color and so on. Without information on the objects, we end up losing flexibility.

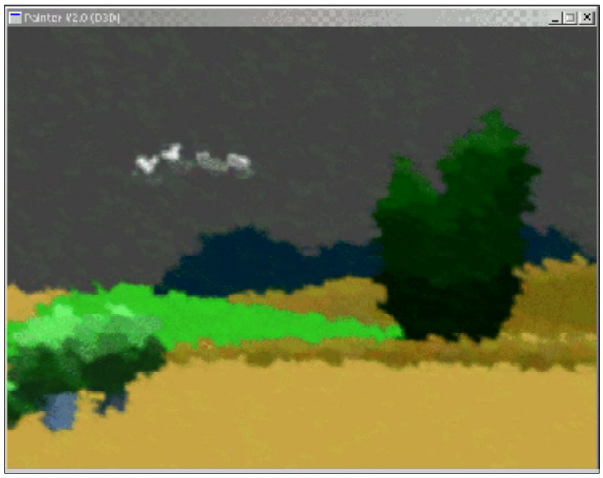


Figure 2

Figure 2 is a screenshot from aEmber v2. We can see that brushstrokes cover the entire screen. We can also see that the brushstrokes are relatively uniform in size and direction, albeit that those variables are lightly jittered. Color is also lightly jittered to produce some more natural variations. However, that jittering also implies that more random variation will be applied for subsequent frames. Thus, the same view will not be the same for the next frame, unless the program kept track of a brush state tied to every sampled pixel. This would be effective in keeping the incoherency down, but it would waste and fragment a great deal of memory. Moreover, there would be a need to keep track of pixel motion in the instance that the user moved the camera viewpoint.

Most significant, however, is the fact that all the brushes are the same stroke. Because we cannot discern object identification directly using only pixelmaps, we are forced to limit the scene to a single brushstroke. In a purely software-rendered engine, we could have used the alpha channel of the pixmap to hold object id numbers since the alpha channel does not affect 2d blits. However, in a hardware-accelerated engine, the alpha channels will not be written. Moreover, if we held id information in the alpha channel while rendering the brushes, they will be treated as actual transparency values.

This necessity to fill the screen with the same brushstroke defeats the purpose of our hierarchical structure which allowed the user to have total brushstroke choice control within a painting. Even if we could discern and identify objects, we would still have to go further and identify individual points.

Compromise

aEmber v3 uses the final approach, which works out as a compromise between v1 and v2's algorithm. In aEmber v3, we still sampled data from the pixelmap. However, the locations of sampling were chosen based on the model.

In v3, we continued to use the tree structure that allowed a user to have tight control over brushstrokes down to a single point. The points, again, were generated via a simple mesh tessellation. The tessellation takes place using the 2-d projected XY coordinates of the triangles. And those locations are sampled for color information. Because the sampling takes place at the mesh level, we can keep track of object ownership. Every sampled pixel will be within the triangle, which is a mesh member of the particular object being rendered at that time.

The resulting animation will have some slight lack of coherency due to the fact that the brushstroke orientations are calculated at render time. Brushstrokes can be either billboard textures or textured polygons that are aligned by surface normal. Slight errors will cause some variation from frame to frame. Also, the tessellated points will be very similar in Z-value, so the low-accuracy Z-buffers present in consumer-grade accelerators will cause some brushstrokes to pop back and forth with respect to other strokes.

However, this slight loss of coherency is acceptable. We also get full coverage of the screen so long as there is 3d scenery to cover the whole screen. aEmber 3 also supports skybox textures in the case that the model space is not enough to fill the screen.

Figure 3 at right is a screenshot from the final version of aEmber. In this case, we only have objects and no skybox textures. At SIGGraph 2000, we were provided with nVidia Quadro-powered workstations. The objects are tessellated to a depth that reflects that kind of a setup. On ordinary home hardware (GeForce2 MX), the same scene renders as fast as 45 fps. To illustrate the point of our hierarchical

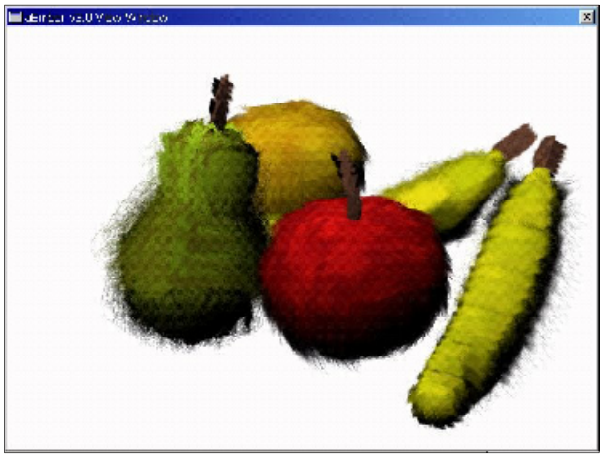


Figure 3

structure, which allowed individual polygon/ point control over brush styles, the original .OBJ file had the objects grouped together. Thus, the whole scene is stored in memory as being a single object. Because of this, brushes had to be assigned as deep as individual polygons and even points in some cases.

With more complex scenes, like the "Grass Huts" scene, there were several more brushstrokes to render, and so framerates around 21 fps are achievable on the same hardware.

Conclusion

With the final version of aEmber, we reached what we believe to be a reasonable compromise for the sake of interactive frame rates. While frame-to-frame coherency was slightly sacrificed, the quality of the renderings is significantly greater than we achieved in aEmber v1.

Many of the coherency issues can be hidden using animated brushstrokes and similar effects. Popping back and forth will still occur, but it becomes harder to pay attention to such a detail. On professional workstation-grade hardware, it has been shown that the coherency problems exist, but are not as severe. Supposedly, the only remaining coherency variables are those related to floating point error in calculating brushstroke locations and orientations. However, we cannot get such high framerates out of professional-grade hardware, as it is not designed for the purpose of drawing hundreds of textured alpha-blended triangle strips.

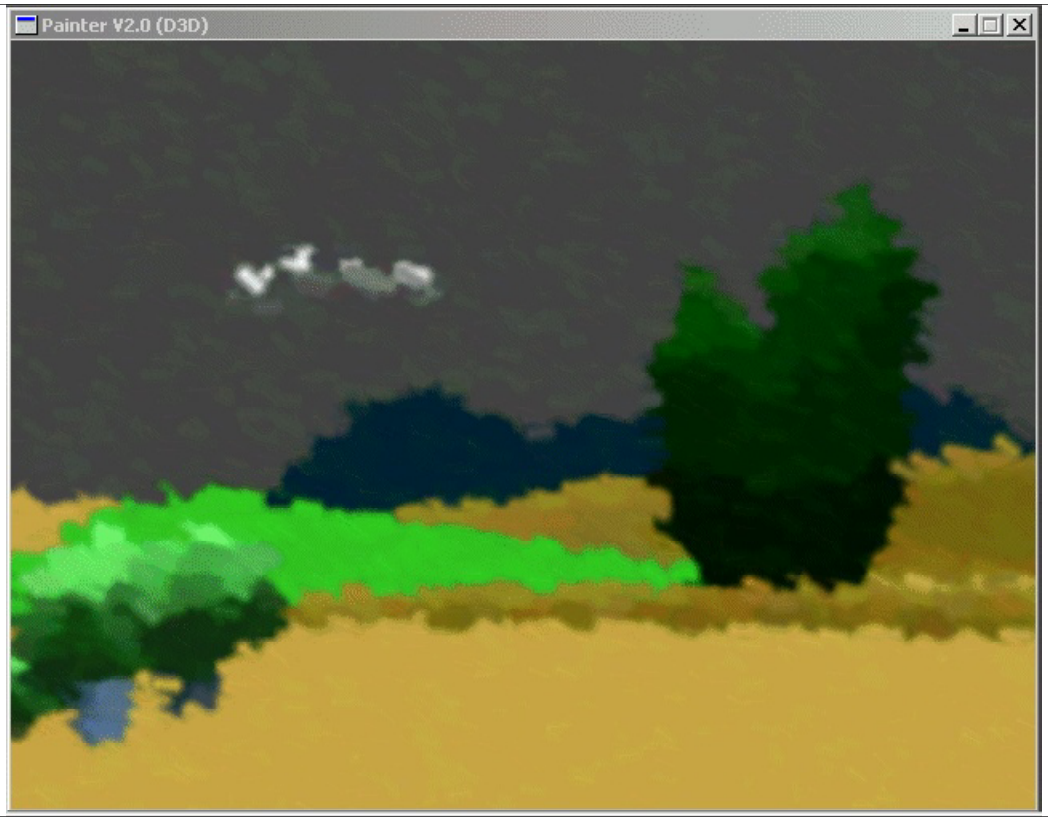
We also tend to lose some coherency due to the fact that brushes lie outside the contours of the object. We felt that this was necessary because of the fact that real brushstrokes will have such features. A painting of an impressionist style will make a point of visible brushstrokes, and more often than not, they will not "stay within the lines." However, when a point from the tessellated mesh is along one of the edges, it could become obscured if the viewpoint is rotated, and thus, the brushstroke will not be rendered. If that brushstroke contained pixels that lie outside the range of the object contours, we could see parts of those "fuzzy" edges vanish.

With current hardware, it is possible to re-implement aEmber's method through pixel and/or vertex shaders. While no such technologies existed at the time of aEmber's original implementation, it could serve to achieve far higher framerates, and quite possibly better quality and coherency. The only concern is as to whether the instruction limit on most pixel shaders allows enough complexity to achieve the desired effect. As for aEmber itself, we feel it has reached the desired results as far as quality and speed with the third version.

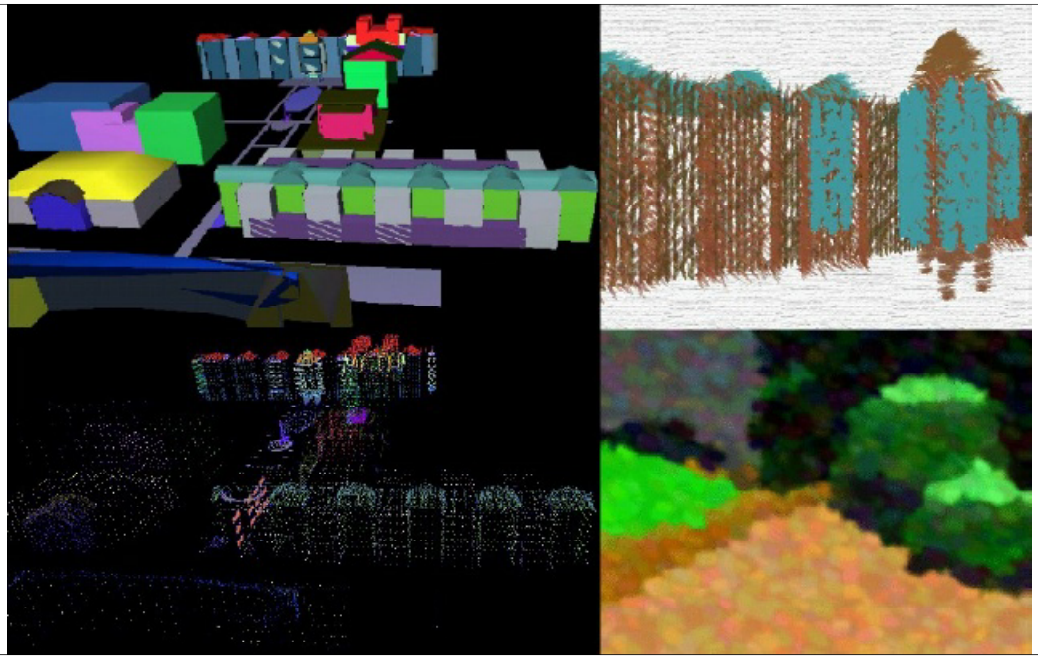
References

[1] Barbara J. Meier. Painterly Rendering for Animation. In *SIGGRAPH 96 Conference Proceedings*, Pages 477-484, August 1996.

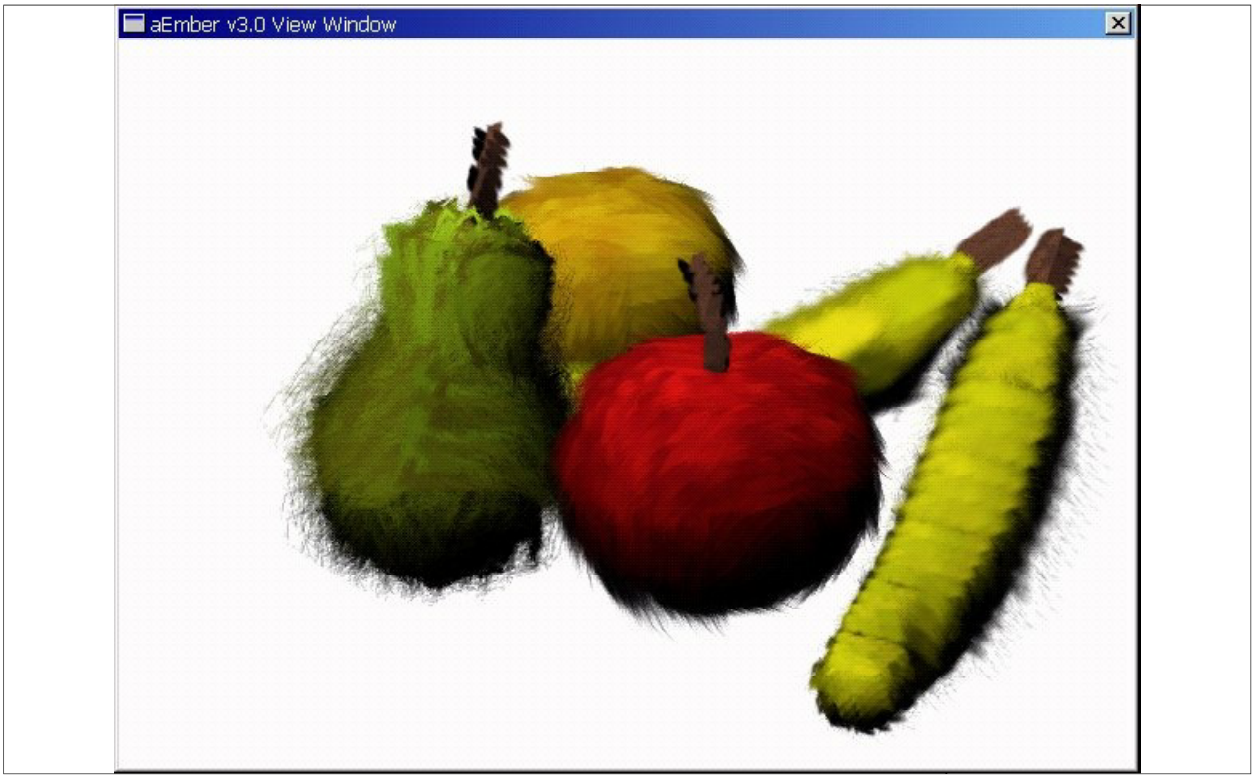
aEmber v1 screenshot. "Van Gogh" scene



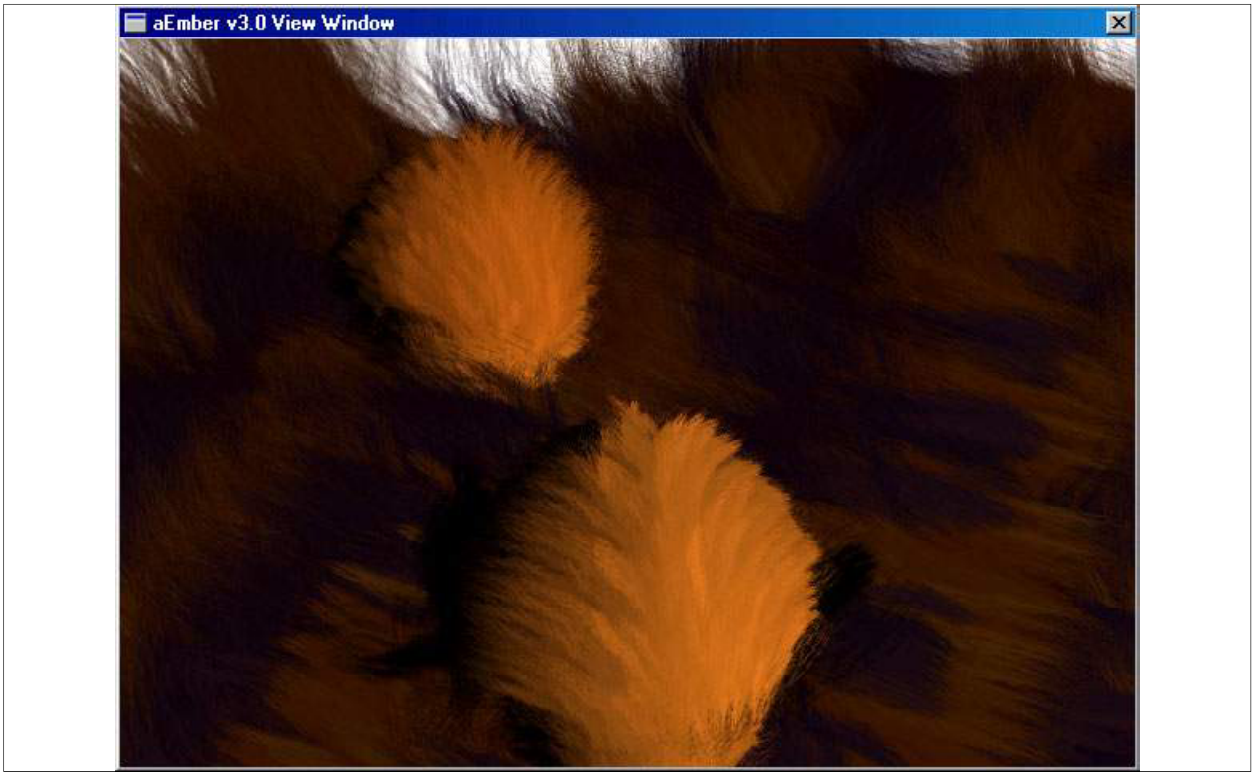
aEmber v2 screenshot. “Fields of Elysium” scene



Rendering stages. *Upper Left* : The initial model of the UIUC Engineering Quad. Rendered with GL lighting. *Lower Left* : North Quad scene has been tessellated and the point clouds are rendered directly in our editing/preview tools. *Upper right* : aEmber v1 screenshot. Shown is the Beckman Institute model from the same scene data. *Lower right* : aEmber v2 screenshot for comparison. Notably, brushstroke style is common throughout the scene.



aEmber v3 screenshot. “Fruits” scene.



aEmber v3 screenshot. “Grass Huts” scene.