

HoloFlicks

Michael Deering[†]
Sun Microsystems

1 Abstract

This paper introduces the concept of HoloFlicks. The equivalent of holographic movies are produced by rendering real-time streams of compressed geometry in a head-tracked stereo viewing environment. Linear narratives may be pre-computed using arbitrarily complex animation, physics of motion, shadowing, texturing, and surface modeling techniques, but the viewpoint-dependent rendering is deferred to playback time. The geometry is produced by a *geometry shader*: outputting micropolygons in world space via programmable shaders via a realistic rendering package (RenderMan). The technology is also applicable to scientific visualization and live 3D television.

CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.7 [Computer Graphics]: Three Dimensional Graphics and Realism.

Additional Keywords and Phrases: Animation, Rendering, Compression Algorithms, Image-Based Rendering, Virtual Reality, Head Mounted Displays.

2 Introduction

If it were possible to inexpensively create and display full-color, high-resolution motion holograms, this would probably be the ultimate visual display for linear storytelling: television and movies in full 3D. Unfortunately, it is highly unlikely that conventional holographic technology will soon, if ever, be practical for building such displays.

An alternate technique for achieving the same visual effect is based on virtual reality technology: dynamically render a pair of 3D images based on the physical location of the human viewer's head relative to the physical world on a stereo display device. Such head-tracked stereo systems produce a look-around holographic feel qualitatively different from older fixed-image-pair stereo displays.

Unfortunately, the key to this alternate technique is the ability to render in real-time from new viewpoints; thus the quality of the imagery is limited to the quality of the real-time image generator. This

[†]901 San Antonio Road, UMPK27-101
Palo Alto, CA 94303-4900
michael.deering@Eng.Sun.Com (650) 786-6325

certainly applies for truly interactive applications, such as CAD and most 3D computer games. But what about linear story telling? Here, most of the details of the animation or real-world events are fixed; only the perspective viewpoint is not.

To date, most real-time 3D rendering systems go through the same process as their orders-of-magnitude slower batch realistic cousins: geometry is generated or modified on the fly, and most shading calculations are re-computed per frame. However, the advent of high triangle rendering rate hardware with real-time geometry decomposition, coupled with the falling prices and amazing capacity increases of storage media make another alternative possible. The idea is to batch render each frame of an animation, but stop the rendering process with micropolygons in world space, not screen space; use geometry compression techniques to compress the results, and store each frame as totally separate data on a storage device. Then, during playback, stream the compressed geometry from storage to the real-time rendering hardware. Because the viewing matrix need not be specified until just before the final display, head-tracked stereo display techniques can be used to display the pre-computed geometry as "nearly holographic" images.

This paper will describe the techniques involved in producing such animations, and some of the limitations. Three experimental sequences are described; one each in the areas of entertainment, real-world capture, and scientific visualization. Some comments on the applicability of these techniques to live 3D television are also made.

3 Batch Realistic Rendering

Modern 3D realistic rendering technology is capable of generating richly detailed synthetic imagery of virtuality *anything* the human mind can imagine. Much of this power comes from the use of programmable shaders that support sophisticated creation and manipulation of geometry and texture. Geometric primitives of choice are NURBS, subdivision surfaces, displacement maps, or procedural definitions; surface shaders use as primitives image texturing, procedural texturing, bump and shadow mapping, and build up from there, combining as many layers as needed.

But this sophistication comes at a price: the rendering time for highly realistic animations can be six orders-of-magnitude slower than real-time. Even if computer processing power continues at its current trend of doubling every 18 months, it will be 30 years before we can expect to see Toy Story rendered in real time.

4 Scripted Linear Story-Telling

While some forms of entertainment, such as interactive 3D games, rely upon real-time physics-based interaction of the user with a 3D data base, this is not the case for 3D animation or movies. Paradox-

ically, to make it look more “real”, directors of scripted 3D action will violate the laws of physics and lighting wherever it helps get the look they want. Off camera fill lights, spot lights and colored gels are the norm for physical filming. Similar and even more audacious “cheats” are done with digital light sources in high-end 3D animation. While the draping of a net over an object must look physically correct in the final imagery, *where* the net falls is specifically pre-ordained by the script.

The point is that no matter how sophisticated future 3D display technology might become, a linear story viewer will likely be akin to Scrooge dragged along by the ghost of Christmas past: the viewer can see everything in full three dimensions, but cannot touch; and the viewer can only go to the places where the ghost goes.

3D computer graphics will continue to make great contributions to many areas of entertainment in the future. However, the techniques of this paper are most appropriate for those entertainment forms amenable to the constraint that the only interaction allowed by the user is control of the viewpoint over a limited range. Note that this covers not only pre-scripted linear story telling, but most forms of remote viewing, specifically including live sporting events.

5 Real-Time Rendering

Real-time computer graphics hardware is also making impressive advances, but such systems do not support complex deformations of surfaces or accurate physics of motion, let alone programmable shaders.

Looked at another way, current general purpose computers take much longer to place and create non-trivial geometry than special purpose rendering hardware does to render it (albeit with simplistic hardwired surface shaders), and the trend is likely to get worse rather than better for some time to come. But for pre-scripted linear story telling, nearly *all* the geometry creating and much of the surface shading can be pre-computed; the primary run-time task is the rendering of large numbers of colored micropolygons.

This brings up another important point: while real-time renderers are still far behind batch systems for the creation of geometry and sophisticated shading effects, the actual raw non-textured polygon rendering rates are getting quite close. At present, low end 3D rendering hardware is in the million triangle per second range, with the medium end approaching 10 million. Given current trends, these numbers are likely to continue to improve soon by another order of magnitude. On the batch animation side, movie film resolution imagery uses about 5 million micropolygons per frame; at NTSC resolutions the number drops to under a million. Thus for a VGA-resolution stereo panel at a 24Hz frame rate, the current hardware triangle rendering rates are almost fast enough for real-time display of the micropolygons (as will be seen in the results section).

6 Previous Work

Certainly some of the ideas described here have been applied before in simpler forms. Perhaps the closest are interactive walk-throughs of pre-computed (viewpoint independent) radiosity renderings converted to colored triangles. For the most part, these have been single static frames, frozen in time, and any texturing was applied in real-time (modulated by the lower-frequency radiosity-lit solution).

Séquin [5] described a partial computation of the raytracing calculations for realistic rendering, to speed up re-rendering when certain shading parameters changed, but here the viewpoint was required to be fixed.

Another comparison can be made to image-based rendering techniques. When the micropolygons are generated by natural imagery rather than a geometry shader, our technique can be viewed as the

ultimate extreme of image warping for interpolated new viewpoints. The closest real-world capture work is Virtualized Reality [3].

7 The Technique

First, in a batch phase, individual frames are rendered using a geometry shader. This rendering is very similar to the normal programmable shader rendering, except that the rendering process stops after the generation of colored (mostly lit) micropolygons in world space, rather than micropolygons in screen space. All shading and most lighting effects are applied to generate the colors. The resulting geometry is compressed and stored, e.g., on disk.

For playback, the frames of compressed geometry are streamed off disk and rendered in real time using real-time user-controlled view parameters. Some view-dependent lighting effects can be applied during the real-time rendering, such as specular highlights and some atmospheric effects.

Because the final viewing parameters are controllable during playback, the person viewing the animation can interactively change his or her viewpoint. If the viewpoint is controlled by a head-tracker and stereo images are generated, the results are visually similar to a motion hologram.

An important aspect of the technique is that an explicit 3D volume of possible eventual real-time viewpoints can be pre-constrained before running the geometry shader. This constrains the amount and tessellation of micropolygons, and also avoids many run-time level-of-detail issues.

8 Geometry Shader

Most programmable shaders, such as RenderMan's [1][6] surface shader, are a super set of the processing we need. Surfaces are tessellated into polygons (typically quads or triangles) that meet a specified criteria: surface curvature, size in screen space, or size in world space. Given explicit constraints on playback viewpoint and screen resolution, screen-space-size criteria for rendering can be chosen that will result in controlled screen-space-size polygons at playback. Shaders apply a lighting and shading model to these micropolygons, including texture and shadow map look-up, ambient and diffuse lighting calculations, or even radiosity computations. These calculations are applied in world space, if effects such as displacement shaders are to be supported. Normally, as a final step, micropolygons are projected into screen space, and then rasterized into a z-buffered frame buffer.

Some shading languages (such as [4]) explicitly support the concept of a geometry shader, in which the final projection and rasterization steps are not applied; instead the lit micropolygons are output. Our experiments used the print command in RenderMan version 3.8 to extract the micropolygons. Because RenderMan internally defers final mesh connectivity to the rasterization stage, the world space micropolygons generated by this print command are un-stitched quad meshes from a higher order surface tessellation. As a post process, we had to recompute the connectivity information and add stitching strips to fill in the cracks. This is a non-trivial task, and nearly doubled the number of micropolygons. (In the future, a more intelligent re-tessellator should reduce this overhead considerably.)

To handle the deferral of specular lighting to playback time, we wrote custom shaders that excluded specular computations. To make sure Renderman did not get too aggressive with occlusion culling, all opaque surfaces were set to be very slightly transparent (0.9999). Otherwise, the shaders differ little from more traditional RenderMan shaders. Clearly, effects that RenderMan applies after we extracted the micropolygons, such as motion blur and final anti-

aliasing, do not show up in the compressed triangle stream. Some view dependent effects, such as reflection or environment mapping, and atmospheric shaders need to be excluded or used with care. Transparency also requires special handling.

Alternately, colored micropolygons can be extracted from real-world scenes by the use of 3D scanners. We created a 108 frame clay animation by scanning a clay head, frame by frame, during animation. Diffuse lights and shadows were generated by physical light sources. The scanner produced detailed triangle models and texture maps. These were run through a geometry shader, once again producing colored micropolygons in world space. These were compressed into the same standard HoloFlick file format as were the RenderMan files; playback proceeds identically.

Eventually more sophisticated real-time 3D capture, conversion to geometry, compression and transmission could support a form of 3D TV. While such geometry acquisition is technically challenging, the playback would be identical to the system described here.

9 Geometry Compression

With the enormous number of micropolygons generated by each frame, some form of geometry compression is mandatory to be able to store the frames on disk. During playback, use of compression greatly reduces the bandwidth necessary for real-time transfer of the frames from disk. To render the compressed geometry in real-time requires a hardware decompressible compression format. Thus for our experiments, we used the compression format described in [2].

A geometry compressor was added as an output processor on the stream of micropolygons generated by RenderMan. The triangles all had individual vertex normal and color components. Changes in specular color and specular power were encoded at a higher object level for use at run-time. The same was done with the color and placement of the specular light sources. Shadowing of objects from specular lights can be achieved by breaking up and regrouping objects by light sources visible to micropolygons.

Because of the presence of both per vertex colors and normals, the compressed geometry averaged about 7 bytes per triangle. We are currently investigating more specialized microtriangle compression techniques that should reduce the average microtriangle storage requirement to the range of 2-3 bytes.

10 Playback

Our initial run-time triangle budget was a quarter of a million triangles per frame, at 24 frames per second. Because of the 70% additional triangles added during the stitching process, our initial experimental runs use 400 thousand triangles per frame, and run at about 15 Hz. In either case, at 7 bytes per triangle, the sustained data transfer rate required is about 42 megabytes per second. We have achieved in excess of this rate for traditional 2D image playback with three-way interleaved fiber-channel disk drives. We are currently looking at an eight-way interleaved array of consumer 5X DVD drives. With a storage capacity of 8.5 gigabytes per side, eight DVD disks would support 17 minutes of continuous playback. Of course, faster 3D rendering hardware would encourage more triangles per frame, and thus increase storage requirements, but better compression techniques would offset this.

For our initial experiments, each frame was stored as a separate compressed object in a separate file (3 megabytes in size). For playback, the files were loaded into a main memory and rendered sequentially.

In a more general format, it would be good to also associate with each frame:

- A list of specular light sources
- A bounding volume constraining allowable viewpoints
- A list of specular materials
- An array of compressed geometry objects
- A list of bindings of specular materials and light sources to compressed geometry objects

11 Experimental Results

Figure 1 is the first frame from a 70-frame test animation produced in RenderMan. The tree was generated by deforming a skeleton and skinning with a custom tree bark shader (using displacement maps). The cobblestone was generated by a simple displacement shader. A standard shadow light shader was used to cast shadows of the tree onto the tree and the cobblestones. The output image size was 640×480, and the shading rate was set to 1.6 pixels. Each frame took only a few minutes to render.

Figure 2 is the first frame from the corresponding 70 frame 210 megabyte HoloFlick animation of the micropolygons extracted from RenderMan. Each frame contains about 400 thousand microtriangles. When putting both the RenderMan output and the real-time rendering up on the same screen it was hard to tell the two apart. Reconnecting and compressing the output of RenderMan to produce each frame of compressed geometry took about two minutes.

Figure 3 is a close-up rendering of the same microtriangle data as Figure 2, but from a viewpoint down near the cobblestones. This is a much greater zoom than would normally be allowed during effective playback.

Figure 4 is a frame from a 100-frame particle system simulation. Each frame has 10,000 particles. Each particle is an individual, and has complete control (at simulation time) of its shape, size, orientation, and color. The total data file is about 45 megabytes in size; the average particle takes about 5.5 bytes of storage. The particles were not generated by RenderMan, but directly generated as compressed data from a physics simulation system.

We also created an 108-frame 52 megabyte HoloFlick using clay animation and a 3D scanner. Each frame contained about 70K triangles.

Our system includes the ability to display HoloFlicks in a variety of head-tracked stereo environments. The displacement mapping and shadows enormously adds to the realism of the head-tracked stereo playback of the cobblestone and carnivorous tree sequence.

12 HoloFlick Cinematography

HoloFlicks represent a new presentation medium and thus may require some different “filming” techniques than existing ones. The very earliest motion picture films were just static camera placement recording of existing plays, it took nearly a decade for the current “language” of cuts, close-ups, etc. to be developed. While a complete treatise is beyond the scope of this paper, early results concerning constraints on viewpoint positioning are quite relevant to the technology described here.

Effective linear story telling does not allow the viewer to wander randomly around the virtual set, but constrains the viewer to a particular region specified by the director/cinematographer. Such a region still must allow for movement of the viewer’s body within a several foot volume for effective stereo display. Within the virtual set, we call this generalized camera platform the “viewer’s couch”. The importance here is that the micropolygon generation does not have to be uniform in world space, but only within a range of tolerances over the limited sighting angles. This allows objects much

further away to be tessellated into corresponding larger micropolygons. There is also the opportunity for backface and occlusion culling of objects or portions of objects that are not visible from any allowed view angle on a given frame. This is also important for capturing micropolygons from the physical world; 3D cameras do not have to see all possible surfaces, only those (potentially) visible from the possible view angles.

13 Future Work

A more complex playback system could perform some viewpoint-dependent occlusion-culling operations to reduce the depth complexity. If objects were compressed at multiple levels-of-detail and decoded in a view-dependent way, the constraints on allowable viewing positions could be reduced, though at the cost of increasing the amount of data stored and transferred for each frame. Objects whose non-specular shading does not change between frames (such as backgrounds) need not be transmitted each frame, and thus only stored at key frames in the file (something like MPEG 2's I-frames). Allowing some dynamically animated objects per frame can add interaction. Branchpoints to different animation sequences could be added (similar to CD-I).

14 Conclusions

By using a geometry shader, it is possible to extract highly detailed image frames as compressed 3D geometry. If this data is streamed from a storage device to a real-time geometry decompression and rendering engine coupled to a head-tracked stereo display, a visual playback system is produced that is similar in effect to a motion hologram.

15References

- 1 Cook, Robert, L. Carpenter, and E. Catmull. The Reyes Image Rendering Architecture. Proceedings of SIGGRAPH '87 (Anaheim, CA, July 27-31, 1987). In *Computer Graphics* 21, 4 (July 1987), 95-102.
- 2 Deering, Michael. Geometry Compression. Proceedings of SIGGRAPH '95 (Los Angeles, California, August 6-11, 1995). In *Computer Graphics* (August 1995), 13-20.
- 3 Kanade, Takeo, P. Rander, S. Vedula, and H. Saito. "Virtualized Reality: Digitizing a 3D Time-Varying Event As Is and in Real Time", in *Mixed Reality, Merging Real and Virtual Worlds*, Yuichi Ohta, Hideyuki Tamura, ed., Springer-Verlag, 1999, pp. 41-57.
- 4 mental images. *mental ray 1.9 User's Manual and Reference*. mental images, Berlin, Germany, February 1996.
- 5 Séquin, Carlo, and E. Smyrl. Parameterized Ray Tracing. Proceedings of SIGGRAPH '89 (Boston, MA, July 31 - August 4) In *Computer Graphics* 23, 3 (July 1989), 307-314.
- 6 Upstill, Steve. *The RenderMan Companion*. Addison-Wesley, 1990.

7FIGURES

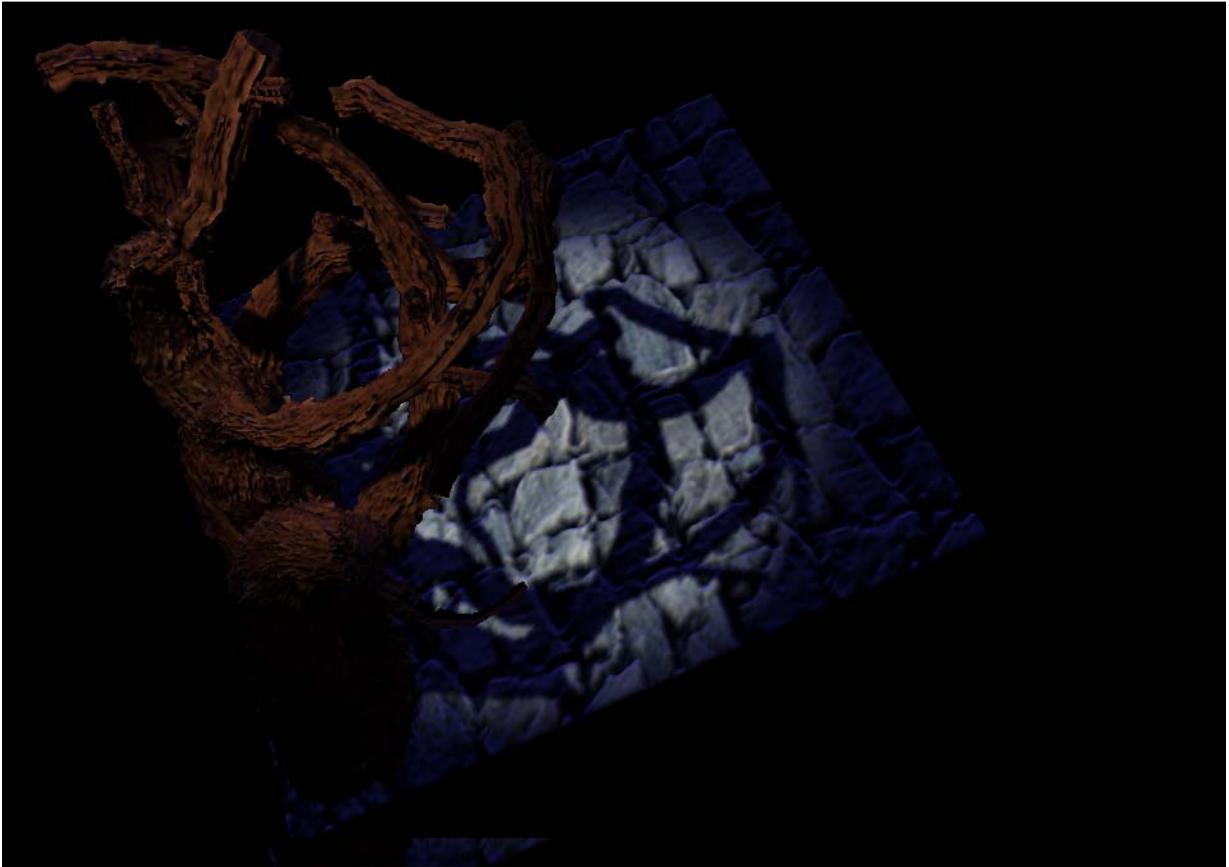
Image of tree trunk and cobbles.

Zoom into micropolygon detail of above.

Image of fountain particle system.

Schematic of possible display physical configurations.

Figure 1: Image of tree trunk and cobbles.



HoloFlicks

HoloFlicks

This page intentionally left blank. Honest.

This page intentionally left blank. Honest.