A Hybrid LOD-Sprite Technique for Interactive Rendering of Large Datasets

Baoquan Chen[1] J. Edward Swan II[2] Arie Kaufman[1]

Abstract

We present a new rendering technique, termed ***LOD-sprite*** rendering, which uses a combination of a level-of-detail (LOD) representation of the scene together with reusing image sprites (previously rendered images). Our primary application is an acceleration technique for virtual environment navigation. The LOD-sprite technique renders an initial frame using a full-resolution model of the scene geometry. It renders subsequent frames with a much lower-resolution model of the scene geometry and texture-maps each polygon with the image sprite from the initial full-resolution frame. As it renders these subsequent frames the technique measures the error associated with each low-resolution polygon, and uses this to decide when to re-render the scene using the full-resolution model. The LOD-sprite technique can be efficiently implemented in texture-mapping graphics hardware.

The LOD-sprite technique is thus a combination of two currently very active thrusts in computer graphics: level-of-detail representations and image-based modeling and rendering (IBMR) techniques. The LOD-sprite technique is different from most previous IBMR techniques in that they typically model the texture-map as a quadrilateral, as opposed to a lower-resolution scene model. This scene model, even if only composed of a few polygons, greatly increases the range of novel views that can be interpolated before unacceptable distortions arise. Also unlike previous LOD techniques, the LOD-sprite algorithm dynamically updates the image sprite every several frames. The LOD-sprite technique can be implemented with any LOD decomposition.

Keywords: Image-Based Modeling and Rendering, Texture Mapping, Acceleration Techniques, Multi-Resolution, Level of Detail, Virtual Reality, Virtual Environments.

1 INTRODUCTION

As virtual environments become more complex (into the millions of polygons), even the most advanced rendering hardware cannot provide interactive rates. This presents two problems, which can be quite severe for many applications: 1) the provided frame rate may be insufficient, and 2) the system latency may be too high. For many virtual reality systems, latency is a more pressing issue than frame rate or even image quality [17]. Recently, there has been a major effort dedicated to finding ways to trade off image quality for frame rate and/or system latency. Many of these recent efforts fall into two general categories:

Level-of-detail (LOD): These techniques model the objects in the scene at different levels of detail. They select a particular LOD for each object based on various considerations such as the rendering cost and perceptual contribution to the final image.

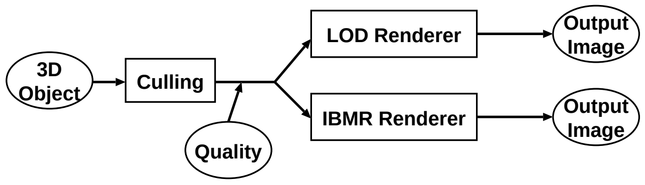


Figure 1: *Traditional Method.*

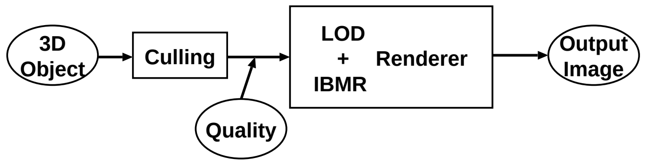


Figure 2: *LOD-sprite Method.*

Image-based modeling and rendering (IBMR): These techniques model (some of the) objects in the scene as image sprites. These sprites only require 2D transformations for most rendering operations, which, depending on the object, can result in substantial time savings. However, the 2D transformations eventually result in distortions which require the underlying objects to be re-rendered from their full 3D geometry. IBMR techniques also typically organize the scene into separate non-occluding layers, where each layer consists of an object or a small group of related objects. They render each layer separately, and then alpha-channel composite them.

Some hybrid techniques use both multiple LODs and IBMR methods [12, 26, 21]. A general pipeline of these techniques is shown in Figure 1. Each 3D object is first subject to a culling operation. Then, depending upon user-supplied quality parameters, the system either renders the object at a particular LOD, or it reuses a cached sprite of the object.

This paper presents the ***LOD-sprite*** rendering technique. An overview of the technique is shown in Figure 2. The technique is similar to previous hybrid techniques in that it utilizes view frustum culling and a user-supplied quality metric. Objects are also modeled as both LOD models and sprites. However, the LOD-sprite technique differs in that the 2D sprite is *coupled* with the LOD representation; the renderer utilizes both the LOD and the sprite as the inputs to create the output image. The LOD-sprite technique first renders a frame from high-resolution 3D scene geometry, and then caches this frame as an image sprite. It renders subsequent frames by texture-mapping the cached image sprite onto a lower-resolution representation of the scene geometry. This continues until an image quality metric requires again rendering the scene from the high-resolution geometry.

We have developed the LOD-sprite technique as part of an effort to accelerate navigating through large virtual environments, and that is the application which is discussed in this paper. However,

[1]Department of Computer Science, State University of New York at Stony Brook, Stony Brook, NY 11794-4400, USA. Email: {baoquan|ari}@cs.sunysb.edu

[2]Virtual Reality Laboratory, Naval Research Laboratory Code 5580, 4555 Overlook Ave SW, Washington, DC, 20375-5320, USA. Email: swan@acm.org

LOD-sprite is a general-purpose rendering technique and could be applied in many different contexts.

The primary advantage of LOD-sprite over previous techniques is that when the sprite is transformed, if the 2D transformation is within the context of an underlying 3D structure (even if only composed of a few polygons), a much larger transformation can occur before image distortions require re-rendering the sprite from the full 3D scene geometry. Thus the LOD-sprite technique can reuse image sprites for a larger number of frames than previous techniques, which allows the system to achieve interactive frame rates for a larger scene database.

The next section of this paper places LOD-sprite in the context of previous work. Section 3 describes the LOD-sprite technique itself. Section 4 discusses our implementation of LOD-sprite and compares our results to a standard LOD technique. In Section 5 we give some conclusions and ideas for future work.

2 RELATED WORK

The previous work which is most closely related to the LOD-sprite technique can be classified into *level-of-detail* techniques and *image-based modeling and rendering* techniques. We first revisit and classify previous IBMR techniques while also considering LOD techniques. In particular we focus on those techniques which have been applied to the problem of navigating large virtual environments.

2.1 Image-Based Modeling and Rendering

Previous work in image-based modeling and rendering falls primarily into three categories:

(1) The scene is modeled by 2D image sprites; no 3D geometry is used. Many previous techniques model the 3D scene by registering a number of static images [3, 2, 14, 24, 25, 15]. These techniques are particularly well-suited to applications where photographs are easy to take but modeling the scene would be difficult (outdoor settings, for example). Novel views of the scene are created by 2D transforming and interpolating between images [3, 2, 20, 14]. Some of these techniques create panoramic images [2, 14, 15, 16, 25], which allow additional areas of the virtual space to be navigated. By adding depth [13] or even layered depth [22] to the sprites, more realistic navigation, which includes limited parallax, is possible. Another category samples the full *plenoptic function*, resulting in 3D, 4D or even 5D image sprites [11, 8], which allow the most unrestricted navigation of this class of techniques.

However, all of these techniques lack the full 3D structure of the scene, and so restrict navigation to at least some degree. The techniques in this class all either interpolate between multiple image or from a single panoramic image. In contrast, the LOD-sprite technique reprojects a single image sprite to represent the scene from a novel viewpoint.

(2) The scene is modeled using either 3D geometry *or* 2D image sprites. Another set of previous techniques model each object with either 3D geometry or a 2D image sprite, based on object contribution to the final image and / or viewing direction [16, 12, 5, 18, 21, 26, 10]. There are a number of examples where this technique is used to accelerate the rendering of large virtual environments [16, 12, 5, 18, 21, 26, 10]. The LOD-sprite technique differs from these techniques in that it integrates both 3D geometry and 2D image sprites to model and render objects.

(3) The scene is modeled using a combination of 3D geometry *and* 2D image sprites. There are two techniques which add very simple 3D geometry to a single 2D image [9, 1]. The 3D geometry serves to guide the subsequent warping of the image. The advantage is that the geometry adds 3D information to the image which allows the warping to approximate parallax, and generally increases the range of novel views which are possible before image distortion becomes too severe. The LOD-sprite uses underlying geometry to achieve a similar advantage. The difference is that with the LOD-sprite technique this underlying geometry comes from a low-resolution version of the scene geometry itself.

In any real-time visualization system the frame rate that can be generated changes as the viewing parameters change and different parts of the scene move in and out of view [5, 7]. Because IBMR techniques are less sensitive to scene complexity, they tend to generate frames at a more constant rate than LOD techniques. Even if frames become quite warped waiting for the next frame generated from the full scene database, the negative usability effect of looking at a warped or distorted image may well be less than the effect of variable latency [17, 27]. This makes IBMR techniques particularly well-suited for real-time visualization systems.

2.2 Level-of-Detail

There is a large body of previous work in level-of-detail (LOD) techniques, which is not reviewed here. The LOD-sprite technique requires that geometric objects be represented at various levels of detail, but it does not require any particular LOD representation or technique.

The LOD-sprite technique is related to the LOD technique of Cohen et al. [4]. Their technique creates a texture and a normal map from a full-resolution 3D object, and then represents the object at a lower resolution while utilizing the previously-created texture and normal maps. The result are 3D objects that retain many perceptually important features while being represented by relatively few polygons. In particular, the normal maps compactly preserve surface curvature information, which greatly improves the appearance of the low-resolution objects. However, normal-mapping rendering hardware is not yet available, and so the technique cannot yet be applied to real-time systems. Soucy et al. [23] describe a very similar technique which only applies the texture map to the lower resolution object. Both techniques are similar to LOD-sprite in that they use a low-resolution object representation which is texture-mapped with higher-resolution information. However, the LOD-sprite technique uses image sprites for the higher-resolution information, while the other techniques use information from the object itself. It would be easy to combine LOD-sprite with these techniques.

2.3 Interactive Virtual Environment Navigation

We have applied the LOD-sprite technique to the problem of interactively navigating a large virtual environment. As stated above, many LOD and IBMR techniques have been applied to this problem. In particular, our implementation of LOD-sprite is related to the techniques of Shade et al. [21], Maciel and Shirley [12], and Schaufler and Stuerzlinger-Protoy [18]. All three papers present a similar hybrid LOD / IBMR technique for navigating large virtual environments. They create a hierarchy of image sprites based on a space partition of the scene geometry (respectively a binary space partition, a k-d tree, and an octree). In subsequent frames, for each node the techniques either texture maps the node's sprite onto a polygon, or re-renders the node's 3D geometry if an error metric is above a threshold. Each reused image sprite means an entire subtree of 3D geometry need not be rendered, which yields substantial speedup for navigating large virtual environments. The main limitation of these techniques is that creating a balanced space partition is not a quick operation, and it must be updated if objects move. The LOD-sprite technique differs from these techniques in

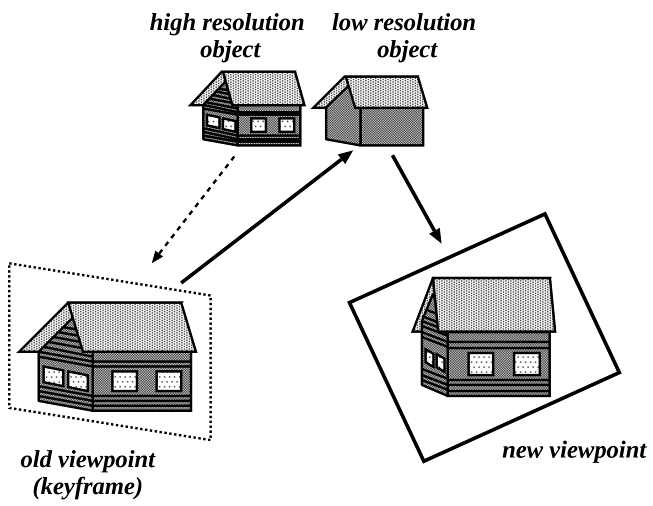


Figure 3: *The main idea behind the LOD-sprite technique.*

that they interpolate the image sprite on a single 2D polygon, while the LOD-sprite technique interpolates the image sprite on a coarse LOD of the 3D scene geometry.

3 THE LOD-SPRITE TECHNIQUE

3.1 Algorithm Overview

The main idea of the LOD-sprite technique is demonstrated in Figure 3. From a certain viewpoint the house is rendered from a high-resolution object representation, which creates a ***keyframe***. To render the house from a new viewpoint, which is not very different from the old viewpoint, the LOD-sprite technique uses a lower-resolution object representation. The sprite obtained from the keyframe is projected onto the new view, so that even though the object is simplified, the final image contains the detail of the high-resolution model.

The mapping function is determined by the viewing parameters when the keyframe is created. This mapping can be efficiently implemented in hardware using OpenGL's *projective texture mapping* [19], and the keyframe image in the frame buffer can be efficiently copied to texture memory by using the *glCopyTexImage2D* OpenGL function.

LOD-sprite algorithm consists of the following steps:

step 1 Render an image from a high-resolution model of the scene. Cache the image as a keyframe, along with the projection matrix for texture coordinate calculation.

step 2 For a new viewpoint, calculate the error associated with rendering each scene object (see Section 3.2 below), and then compare this error with a threshold. If the error exceeds the threshold, return to **step 1** above. Otherwise, continue with **step 3** below.

step 3 From a new viewpoint, render the scene objects using a lower-resolution model of the scene. Texture map each polygon with the keyframe.

step4 Goto **step 2** above.

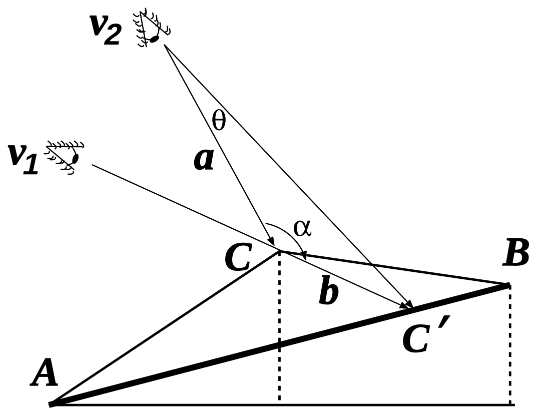


Figure 4: *Calculating the error metric.*

3.2 Error Metric

We decide when to switch back to the full-resolution representation based on an error metric similar to that described by Shade et al. [21]. Figure 4 gives the technique, which is drawn in 2D for clarity. Consider rendering the full-resolution dataset from viewpoint position v_1. In this case the line segments AC and CB are rendered (in 3D these are polygons). From this view, the ray passing through vertex C intersects the edge AB at point C'. After rendering the full-resolution dataset, the image from v_1 is stored as a texture map. Now consider rendering the scene from the ***novel*** viewpoint v_2, using the low-resolution representation of the dataset. In this case the line segment AB will be rendered, and texture mapped with the sprite rendered from v_1. Note that this projects the vertex C to the position C' on AB. From v_1 this projection makes no visible difference. However, from v_2, vertex C' is shifted by the angle θ from its true location C. This angle can be converted to a pixel distance on the image plane of view v_2, which is our measure of the error of rendering point C from view v_2:

$$\theta < \beta \times \epsilon, \tag{1}$$

where β is the view angle of a single pixel (e.g. the field-of-view over the screen resolution), and ϵ is a user-specified error threshold. As long as Equation 1 is true, we render using the LOD-sprite technique. Once Equation 1 becomes false, it is again necessary to render from the full-resolution dataset.

Theoretically, we should evaluate Equation 1 for all points in the high-resolution dataset for each novel view. Clearly this is impractical. Instead, in our implementation we calculate θ for the central vertex of each low-resolution quadtree square. We then calculate the average sum of squares of the error for all evaluated vertices and compare this with $(\beta \times \epsilon)^2$:

$$\frac{\sum_{i=1}^{n} \theta_i^2}{n} < (\beta \times \epsilon)^2, \tag{2}$$

where n is the number of low-resolution quadtree squares. We re-render the whole dataset at a high-resolution as soon as this test fails.

To calculate θ for each vertex, we have to know a (the length of edge v_2C), b (the length of edge CC'), and angle α (the angle between view vectors v_1 and v_2 corresponding to vertex C). Theoretically, α is different from vertex to vertex, thus, calculating α for each vertex is expensive. We therefore assume that all vertices have the same α value. Although not accurate, in practice, this seems to be a good assumption. As shown in Figure 5,

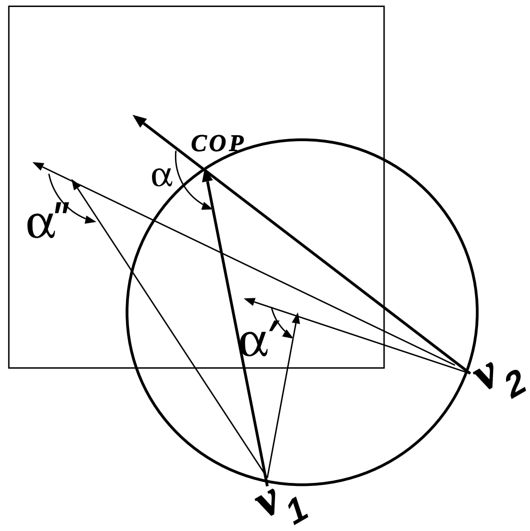


Figure 5: *The error from assuming that all vertices have the same α value.*

the circle passes through the COP and v_1 and v_2. All vertices on the circle (in 3D, on the sphere) have the same α value. Inside the circle, α' is smaller than α; while outside the circle, α'' is larger than α. Since α is larger than 90° (assuming a small view direction change), given a and b, the larger the α value, the smaller the calculated θ value, which means vertices outside the circle get less error than they should receive. In contrast, vertices inside the circle receive a larger error. A justification for this is that in practice, viewers pay more attention to closer objects (which receive a more conservative error evaluation), and less attention to far away objects (which receive a more liberal error evaluation).

To calculate b, we use the distance between vertex C and edge AB instead. This assumption also causes an error in evaluating the error, because when AB is more tilted to view v_1, the calculated b is shorter than it should be. However, the more tilt of edge AB, the smaller its screen projection becomes, and thus the precision of the error calculation can be relaxed.

3.3 Visibility Changes

As the viewpoint changes, the visibility of the rendered geometry changes in two different ways:

1. the original occluded scene geometry becomes visible, and

2. the original culled scene geometry becomes visible.

For the LOD-sprite frames, both visibility changes mean that either the wrong texture or even no texture is mapped to visible polygons.

The first visibility change occurs when the viewpoint changes. Therefore objects originally occluded become visible, and objects originally visible become occluded. The is not addressed by the rendering hardware, because OpenGL's projective texture mapping does not test for depth. To solve this, when we store the keyframe, we store both the z-buffer and the viewing matrix. Then, for each vertex, we calculate the (x, y) screen coordinate and the z-depth value for the *keyframe* viewpoint. We compare this depth value to the z value at location (x, y) in the z-buffer. This tells us whether the vertex is occluded from the current viewpoint. If all the vertices for a given polygon are unoccluded, then we texture map the

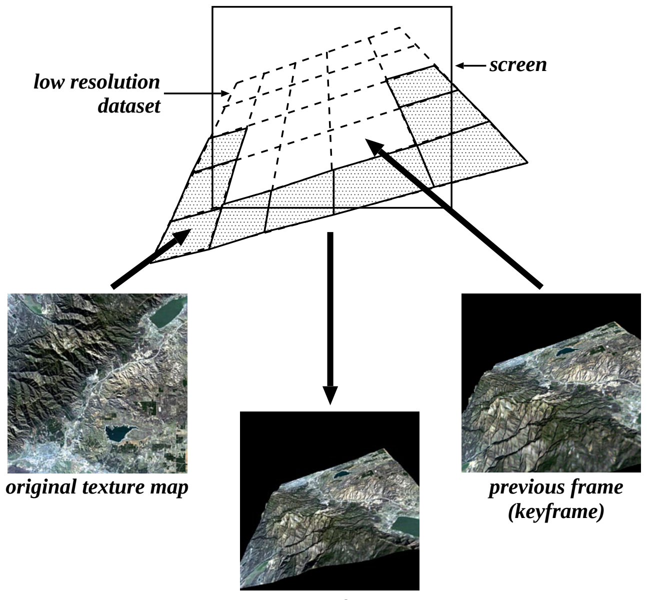


Figure 6: *Handling view frustum visibility changes by texture mapping from both the keyframe and the original texture map.*

polygon from the keyframe. However, if any of the vertices are occluded, we texture map the polygon from the original texture map, which lets the hardware z-buffer properly test for occlusion.

The second visibility change is not a problem when the moving view frustum hides scene geometry (e.g. when zooming in). However, it is a problem when new scene geometry is revealed — if there is no texture to map to the new geometry, the result is a severe visibility artifact. Figure 6 shows an example of this problem, which only occurs when zooming out. In Figure 6, the image sprite is created when the portion of the dataset represented by the dashed squares is visible. As the user zooms out, new parts of the dataset become visible, represented by the shaded squares. Our solution is to texture map the dashed squares with the image sprite from the keyframe, and texture map the shaded squares with the original texture map.

Because the resolution of the two texture maps is different, both technique can sometimes result in a visible line between the polygons texture mapped from the keyframe and those mapped from the original texture. However, this line is certainly less visible and less distracting than either rendering occluded texture, or rendering blank polygons for the newly visible geometry.

4 IMPLEMENTATION AND RESULTS

Although the LOD-sprite technique is a general-purpose rendering technique that can be applied to any application area, we have developed the technique as an acceleration method for the real-time navigation of large virtual environments. In particular, we are applying the technique to the problem of 3D battlefield visualization [6].

This paper does not cover how to create LOD representations of a terrain. However, the LOD-sprite technique works with all LOD methods, either view-dependent or view-independent. Here we take a simple approach and use a quadtree representation for the terrain. Since we receive our input data in the form of a rectilinear height field, which easily lends itself to multiple LODs by down sampling

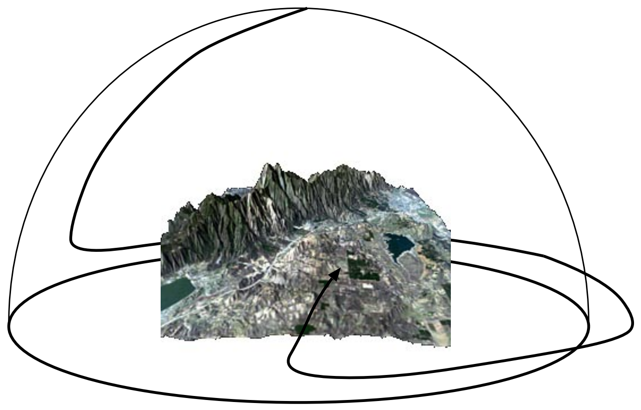


Figure 7: *The camera path for Figures 8–12.*

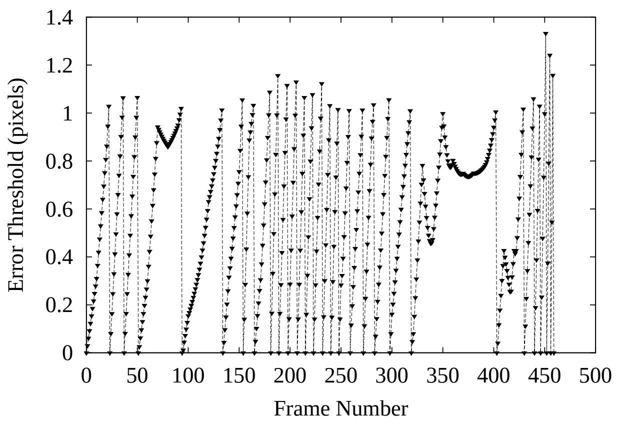


Figure 8: *The error in pixels versus frame number. 460 frames; path from Figure 7.*

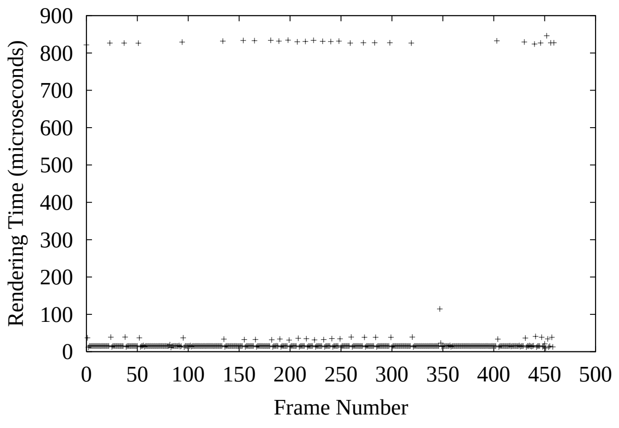


Figure 9: *The rendering time in microseconds versus frame number. 460 frames; path from Figure 7.*

the terrain mesh, a quadtree is a natural data structure.

Results are shown in Figures 13 and 14. The input is a 512×512 height field and 512×512 texture map. The full-resolution scene geometry is $512^2 = 262,144$ quadrilaterals, while the low-resolution scene geometry is $16^2 = 256$ quadrilaterals, a reduction of three orders of magnitude.

We first evaluate the image quality of the LOD-sprite technique. Figures 13(a)–(c) compare the LOD-sprite technique to a standard LOD technique. In order to better display differences in surface orientation, the terrain is texture-mapped with a red and green checkerboard pattern instead of the real terrain data. Figure 13(a) shows the terrain rendered from the full 512×512 height field, while Figure 13(b) shows the terrain rendered from a 16×16 height field. Comparing 13(a) to 13(b), we see that, as expected, many surface features are smoothed out. Figure 13(c) shows the same frame rendered with the LOD-sprite technique, using the same 16×16 height field as Figure 13(b) but texture mapped with Figure 13(a). Unlike Figure 13(b) the surface features are quite well preserved, yet Figures 13(b) and 13(c) take the same amount of time to render.

Figure 14(a)–(e) show similar results but are texture mapped with the actual terrain data. Figure 14(a) is the image rendered from the full 512×512 height field. Figure 14(b) is rendered from the 16×16 height field, and the difference between it and Figure 14(a) is shown in Figure 14(d). Figure 14(c) is rendered with the LOD-sprite technique, using the 16×16 height field and texture mapped with Figure 14(a). The difference image is shown in Figure 14(e). Comparing Figures 14(d) and 14(e), we see that the LOD-sprite technique generates an image that is much closer to the full-resolution image, and yet requires no more texture-mapped polygons than the standard LOD technique.

Figures 7–12 give the algorithm's timing behavior for a given camera path. Figure 7 shows the camera path. The camera starts from the zenith, rolls down to the horizon, rotates around the terrain about 270 degrees, and then zooms into the center of the scene. This path represents all typical camera motions, including rotation, translation, and zooming. The animation contains 460 frames for all the figures except for Figure 10, where the frame count is varied. Each frame was rendered at a resolution of 512×512. The animation was rendered on an SGI Power Challenge with 3G RAM, Infinite Reality Graphics, an R10000 CPU, and an R10010 FPU.

Figure 8 shows how the error changes as each frame is rendered. The error always starts from zero for the keyframe. As more novel views are interpolated from the keyframe, the error increases. When the error exceeds 1.0 pixels, we calculate another keyframe from the full-resolution scene geometry, which again drops the error to zero.

Figure 9 shows the amount of time required to render each frame. Note that the results congregate at three levels. The upper set of points gives the rendering time for the keyframes. For this animation the system generated 28 keyframes using the full 512×512 height field, at an average time of 830 microseconds per frame. The lower set of points gives the rendering time for the frames rendered using the LOD-sprite technique. The system generated 403 such frames, at an average time of 15 microseconds per frame. Just above the lower set of points is another line of points. These are also rendered with the LOD-sprite technique, but they represent the time for a frame rendered directly after a keyframe is rendered. These frames take an average of 36 microseconds to render; the extra time is spent reading the texture map created in the previous frame into texture memory. The system generated 28 such frames (one for each keyframe).

Figure 10 shows the fraction of the total number of rendered frames which are keyframes. This is plotted against the total number of frames rendered for the path shown in Figure 7. As expected, as more frames are rendered for a fixed path, the distance moved between each frame decreases, and so there is more coherence between successive frames. This figure shows how our system takes

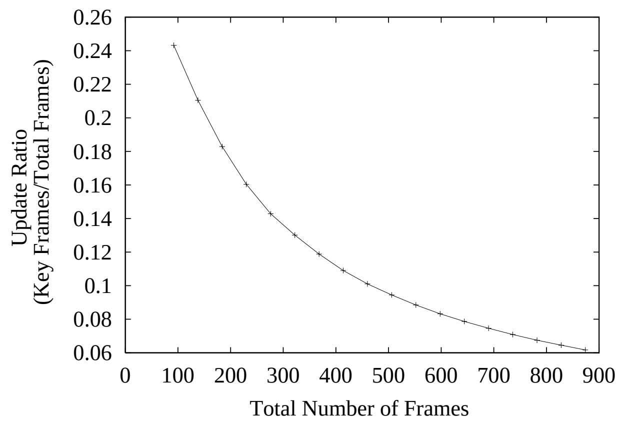


Figure 10: *The fraction of keyframes versus the total number of frames rendered. Path from Figure 7.*

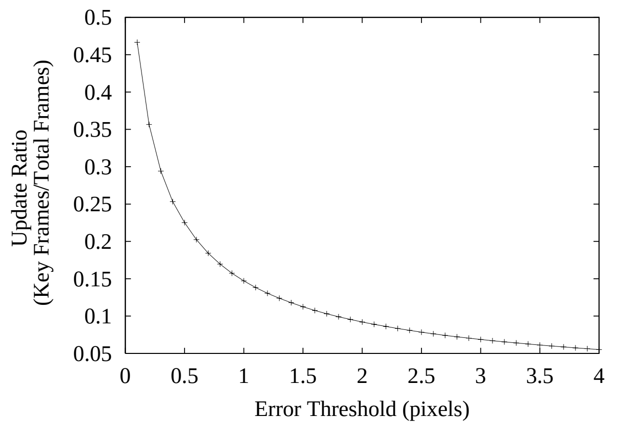


Figure 11: *Update rate as a function of error threshold. 460 frames; path from Figure 7.*

advantage of this increasing coherence by rendering a smaller fraction of keyframes. This figure also illustrates a useful property of the LOD-sprite technique for real-time systems: as the frame update rate increases, the LOD-sprite technique becomes even more efficient in terms of reusing keyframes.

Figure 11 also shows the fraction of the total number of rendered frames which are keyframes, but this time plots the fraction against the error threshold in pixels. As expected, a larger error threshold means fewer keyframes need to be rendered. However, the shape of this curve indicates a decreasing performance benefit as the error threshold exceeds 1.0 pixels. For a given dataset and a path which is representative of the types of maneuvers the user is expected to make, this type of analysis can help determine the best error threshold versus performance tradeoff.

The LOD-sprite technique results in a substantial speedup over rendering a full-resolution dataset. Rendering 460 frames of the full-resolution dataset along the path in Figure 7 takes 383.2 seconds. Rendering the same 460 frames with the LOD-sprite technique, using an error threshold of 1.0 pixel, takes 31.1 seconds. This is a speedup of 12.32. Figure 12 shows how the speedup varies as a function of the error threshold.

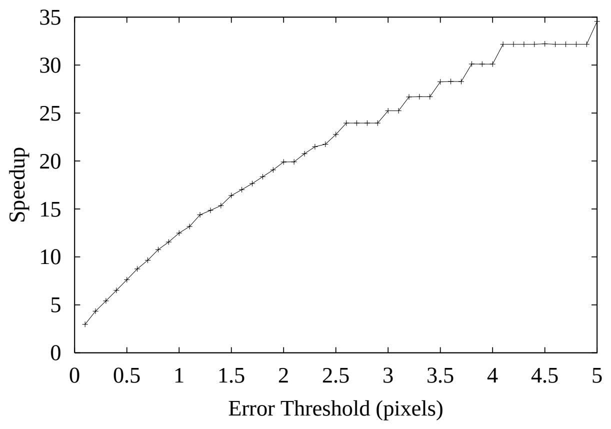


Figure 12: *Speedup as a function of error threshold. 460 frames; path from Figure 7.*

5 CONCLUSIONS AND FUTURE WORK

This paper has described the LOD-sprite rendering technique, and an implementation which is geared to accelerating rendering for virtual reality applications. The technique is a combination of two rich directions in accelerated rendering for virtual environments: multiple level-of-detail (LOD) techniques, and image-based modeling and rendering (IBMR) techniques. It is a general-purpose rendering technique that could accelerate rendering for any application; it could be built upon any LOD decomposition technique and utilize a number of different IBMR interpolation techniques. It improves upon LOD techniques by preserving surface complexity, and it improves upon IBMR techniques by increasing the range of novel views that are possible before requiring the scene to be re-rendered from the underlying 3D geometry. The LOD-sprite technique is particularly well-suited for real-time system architectures that decompose the scene into coherent layers.

Our primary applied thrust with this work is to fully integrate it into the Dragon battlefield visualization system. However, the work also has numerous areas for future research efforts, including:

- When the previously rendered image is texture-mapped onto an object to create a new image, the texel from the original texture is resampled twice. It should be possible to characterize this process using the language of sampling theory. This would not only be interesting, but it might lead to better error metrics.
- Our current implementation utilizes a fixed LOD representation. We plan to integrate the LOD-sprite technique into a system which utilizes dynamic, viewpoint-dependent LODs.
- Another issue is the latency required to render the keyframe. One optimization is to use a dual-thread implementation, where one thread renders the keyframe while another renders each LOD-sprite frame. Another optimization is to render the keyframe in advance by predicting where the viewpoint will be when it is next time to render the keyframe. We can predict this by extrapolating from the past several viewpoint locations. Thus we can begin rendering a new keyframe immediately after the previous keyframe has been rendered. If the system makes a bad prediction (perhaps the user made a sudden, high-speed maneuver), two solutions are possible: 1) we could use the previous keyframe as the sprite for additional

frames of LOD-sprite rendering, with the penalty that succeeding frames will have errors beyond our normal threshold. Or, 2) if the predicted viewpoint is closer to the current viewpoint than the current viewpoint is to the previous keyframe, we can use the predicted viewpoint as the keyframe instead. We are currently implementing both techniques.

- We also intend to implement a cache of keyframes. This will accelerate the common user navigation behavior of moving back and forth within a particular viewing region. Issues include how many previous keyframes to cache, and to evaluate different cache replacement policies.

- Finally, an important limiting factor for the performance of the LOD-sprite technique, as well as other image-based modeling and rendering techniques ([21], for example), is that OpenGL requires texture maps to have dimensions which are powers of 2. Thus many texels in our texture maps are actually unused. The LOD-sprite technique could be more efficiently implemented with graphics hardware that did not impose this constraint.

ACKNOWLEDGMENTS

This work was supported by The Office of Naval Research grant N000149710402, and The National Science Foundation grant MIP-9527694. We acknowledge Larry Rosenblum for advice and direction during this project.

References

[1] G. U. Carraro, J. T. Edmark, and J. R. Ensor. Techniques for handling video in virtual environments. In *SIGGRAPH 98 Conference Proceedings*, pages 353–360, July 1998.

[2] S. E. Chen. Quicktime vr - an image-based approach to virtual environment navigation. In *Computer Graphics Proceedings, Annual Conference Series (Proc. SIGGRAPH '95)*, pages 29–38, 1995.

[3] S. E. Chen and L. Williams. View interpolation for image synthesis. In *Computer Graphics Proceedings (Proc. SIGGRAPH '93)*, pages 279–288, 1993.

[4] J. Cohen, M. Olano, and D. Manocha. Appearance-preserving simplification. In *SIGGRAPH 98 Conference Proceedings*, pages 115–122, July 1998.

[5] D. Cohen-Or, E. Rich, U. Lerner, and V. Shenkar. A real-time photo-realistic visual flythrough. *IEEE Transactions on Visualization and Computer Graphics*, 2(3), Sept. 1996.

[6] J. Durbin, J. E. Swan II, B. Colbert, J. Crowe, R. King, T. King, C. Scannell, Z. Wartell, and T. Welsh. Battlefield visualization on the responsive workbench. In *Proceedings IEEE Visualization '98*, Oct. 1998.

[7] T. A. Funkhouser and C. H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 247–254, Aug. 1993.

[8] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. In *Computer Graphics Proceedings (Proc. SIGGRAPH '96)*, pages 43–54, 1996.

[9] Y. Horry, K. ichi Anjyo, and K. Arai. Tour into the picture: Using a spidery mesh interface to make animation from a single image. In *SIGGRAPH 97 Conference Proceedings*, pages 225–232, Aug. 1997.

[10] J. Lengyel and J. Snyder. Rendering with coherent layers. In *SIGGRAPH 97 Conference Proceedings*, pages 233–242, Aug. 1997.

[11] M. Levoy and P. Hanrahan. Light field rendering. In *Computer Graphics Proceedings (Proc. SIGGRAPH '96)*, pages 31–42, 1996.

[12] P. W. C. Maciel and P. Shirley. Visual navigation of large environments using textured clusters. In *1995 Symposium on Interactive 3D Graphics*, pages 95–102, Apr. 1995.

[13] L. McMillan. *An Image-Based Approach to Three-Dimensional Computer Graphics*. PhD thesis, Department of Computer Science, University of North Carolina at Chapel Hill, 1997.

[14] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. In *Computer Graphics Proceedings (Proc. SIGGRAPH '95)*, pages 39–46, 1995.

[15] P. Rademacher and G. Bishop. Multiple-center-of-projection images. In *SIGGRAPH 98 Conference Proceedings*, pages 199–206, July 1998.

[16] M. Regan and R. Post. Priority rendering with a virtual reality address recalculation pipeline. In *Proceedings of SIGGRAPH '94*, pages 155–162, July 1994.

[17] P. Richard, G. Birebent, P. Coiffet, G. Burdea, D. Gomex, and N. Langrana. Effect of frame rate and force feedback on virtual object manipulation. *Presence*, 5(1):95–108, 1996.

[18] G. Schaufler and W. Stuerzlinger-Protoy. A three dimensional image cache for virtual reality. In *Proceedings of Eurographics '96*, pages 227–235, Aug. 1996.

[19] M. Segal, C. Korobkin, R. van Widenfelt, J. Foran, and P. E. Haeberli. Fast shadows and lighting effects using texture mapping. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 249–252, July 1992.

[20] S. M. Seitz and C. R. Dyer. View morphing: Synthesizing 3D metamorphoses using image transforms. In *SIGGRAPH 96 Conference Proceedings*, pages 21–30, Aug. 1996.

[21] J. Shade, D. Lischinski, D. Salesin, T. DeRose, and J. Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In *SIGGRAPH 96 Conference Proceedings*, pages 75–82, Aug. 1996.

[22] J. W. Shade, S. J. Gortler, L. He, and R. Szeliski. Layered depth images. In *SIGGRAPH 98 Conference Proceedings*, pages 231–242, July 1998.

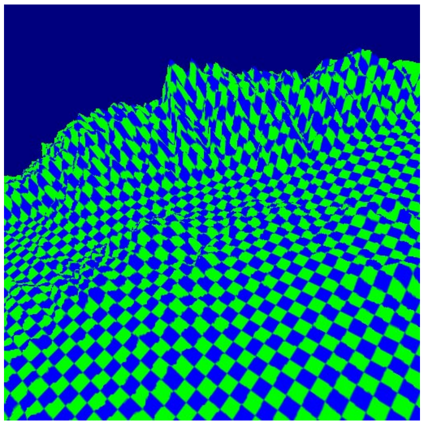
[23] M. Soucy, G. Godin, and M. Rioux. A texture-mapping approach for the compression of colored 3D triangulations. *The Visual Computer*, 12(10):503–514, 1996.

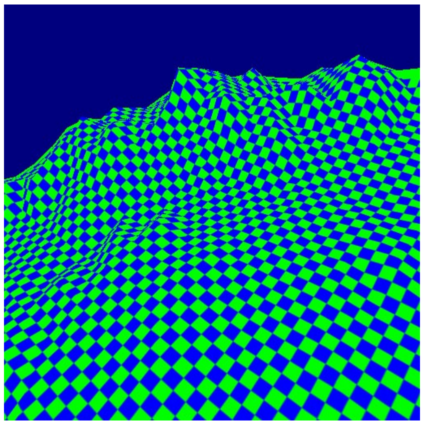
[24] R. Szeliski. Video mosaics for virtual environments. *IEEE Computer Graphics and Applications*, pages 22–30, Mar. 1996.

[25] R. Szeliski and H.-Y. Shum. Creating full view panoramic mosaics and environment maps. In *SIGGRAPH 97 Conference Proceedings*, pages 251–258, Aug. 1997.

[26] J. Torborg and J. Kajiya. Talisman: Commodity Real-time 3D graphics for the PC. In *SIGGRAPH 96 Conference Proceedings*, pages 353–364, Aug. 1996.

[27] C. Ware and R. Balakrishnan. Reaching for objects in vr displays: Lag and frame rate. *ACM Transactions on Computer-Human Interaction*, 1(4):331–356, 1994.


(a) *Standard LOD technique with* 512×512 *height field.*


(b) *Standard LOD technique with* 16×16 *height field.*

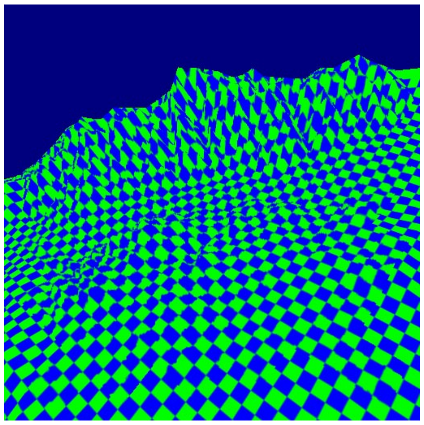
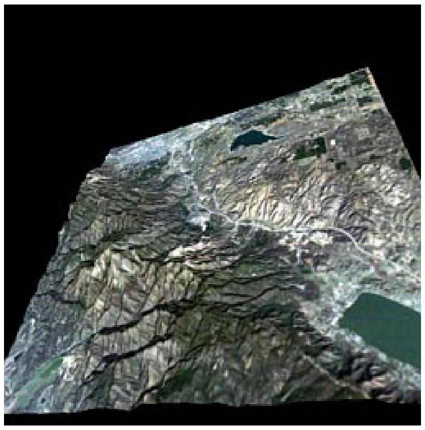
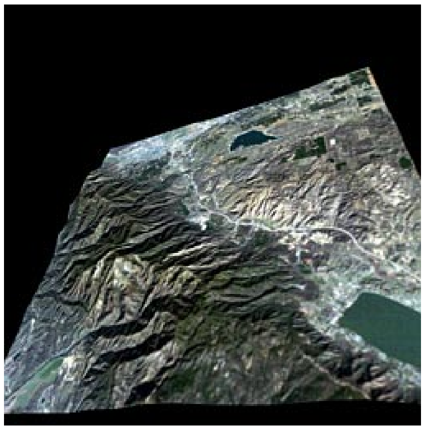
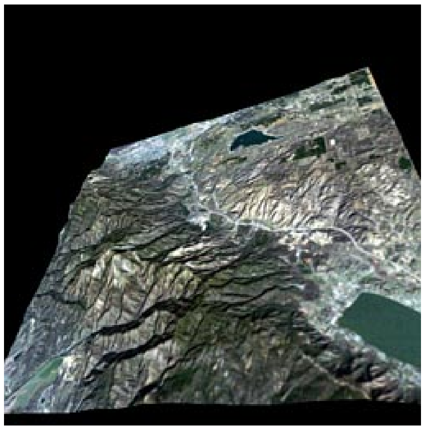
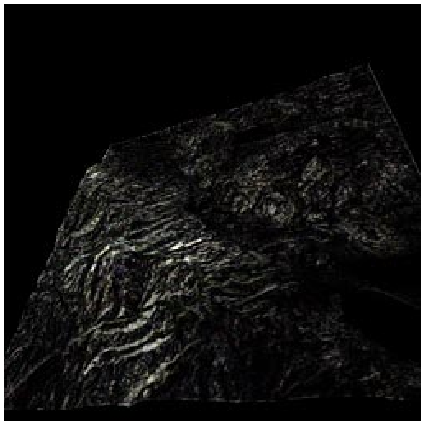

(c) *LOD-sprite technique with* 16×16 *height field* (b).

Figure 13: *Comparing the LOD-sprite technique to a traditional LOD technique, using a synthetic texture.*


(a) *Standard LOD technique with* 512×512 *height field.*


(b) *Standard LOD technique with* 16×16 *height field.*


(c) *LOD-sprite technique with* 16×16 *height field* (b).


(d) *Difference between high-resolution LOD* (a) *and low-resolution LOD* (b).

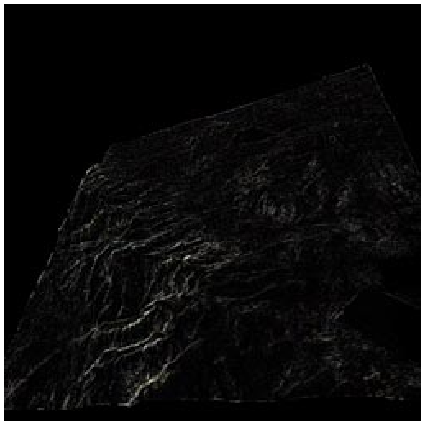

(e) *Difference between high-resolution LOD* (a) *and LOD-sprite* (c).

Figure 14: *Comparing the LOD-sprite technique to a traditional LOD technique, using the actual terrain texture.*