Multiple Viewpoint Rendering

Michael Halle
Brigham and Women's Hospital

Abstract

This paper presents an algorithm for rendering a static scene from multiple perspectives. While most current computer graphics algorithms render scenes as they appear from a single viewpoint (the location of the camera) multiple viewpoint rendering (MVR) renders a scene from a range of spatially-varying viewpoints. By exploiting perspective coherence, MVR can produce a set of images orders of magnitude faster than conventional rendering methods. Images produced by MVR can be used as input to multiple-perspective displays such as holographic stereograms, lenticular sheet displays, and holographic video. MVR can also be used as a geometry-to-image prefilter for image-based rendering algorithms. MVR techniques are adapted from single viewpoint computer graphics algorithms and can be accelerated using existing hardware graphics subsystems. This paper describes the characteristics of MVR algorithms in general, along with the design, implementation, and applications of a particular MVR rendering system.

1 Introduction

Many of the important techniques and algorithms of computer graphics are specifically focused on accelerating the conversion of geometric primitives to images by using coherence of some kind. Published taxonomies of coherence [17] have presented the spectrum of possible coherence types, but common practice has put greater emphasis on some areas and left others generally untouched. In particular, most computer graphics algorithms heavily emphasize the use of image and geometric coherence to accelerate the rendering of a single image. These techniques include some of the most important in computer graphics: polygon scan conversion and incremental shading.

Less common rendering techniques have been used to exploit coherence over multiple views of an object. For example, temporal coherence can be used to speed the rendition of the frames of a computer animation. Coherence across several images, referred to under the blanket name *frame-to-frame coherence*, is very general and scene dependent because of the sheer variety of changes that an object in a scene can experience from one frame to the next. Fully general temporal coherence algorithms must deal with potentially complex camera motion as well as arbitrary object transformation and other changes to the scene. In part because of this generality, the observation made by Sutherland *et. al.* from 1974 is still mostly true today: "It is really hard to make much use of object and frame coherence, so perhaps it is not surprising that not much [use of it] has been made." Recent developments such as the Talisman graphics architecture [19] demonstrate both the promise and the complexities of using temporal coherence.

75 Francis Street, Boston, MA 02115, USA. Email: mhalle@bwh.harvard.edu.
This work was performed while the author was at the MIT Media Laboratory.

2 Perspective coherence

While temporal coherence of time-varying image sequences is an important subclass of frame-to-frame coherence, it is not the only subclass. Another coherence type, *perspective coherence*, is the similarity between images of a static scene as viewed from different locations. Simple observation demonstrates the prevalence of perspective coherence in common "real world" scenes: viewing typical objects by alternating between your left and right eyes produces little apparent change in appearance. Small shifts of your head side to side or up and down usually yields similarly small changes.

Because perspective coherence results from the apparent change of a scene's appearance due solely to a change in camera perspective, it is much more restricted and less general than temporal coherence. Geometric and shading changes to the scene's appearance are usually related to the change in the camera position in a simple way. With the appropriate rendering constructs, perspective coherence is easier to find and to exploit than more general frame-to-frame coherence. This paper describes a method of rendering whereby perspective coherence can be harnessed to efficiently render sets of perspective images.

3 Multiple viewpoint rendering

This text refers to rendering methods that generate perspective image sets as *multiple viewpoint rendering*, or *MVR*, and those algorithms that create single images as *single viewpoint rendering (SVR).* MVR algorithms treat the process of rendering a set of perspective images as a unit, and use the structured coherence of spatio-perspective space to accelerate the process of image data generation. For instance, using a relatively small number of transformation and shading calculations, MVR can interpolate location and appearance of an object through an entire range of views.

4 Applications

Perspective image sets such as those generated by MVR are used less frequently than are animations or other temporally varying image sequences. But perspective image sets have their own important class of emerging uses in computer graphics. This class of applications approximate optical capture, distortion, or display of a field of light emitted by a scene. Two diverse examples of potential applications for MVR-generated image sets include synthetic three-dimensional display and image-based rendering.

4.1 Three-dimensional displays

Multi-perspective 3D or parallax displays, a classification which include lenticular sheet displays, parallax panoramagrams, holographic stereograms, and holographic video displays, mimic the

appearance of three-dimensional scenes by displaying different perspectives of the scene in different directions [10][15]. Most multi-perspective displays are horizontal parallax only: they use a range of perspectives that vary only horizontally in order to provide stereopsis to a viewer. Depending on the exact technology used in the display, the number of perspectives required as input to the display device may range from two to tens of thousands. For three-dimensional images of synthetic scenes, these perspectives must be rendered. The high cost of computing this large amount of image information is currently a major impediment for the development of three-dimensional displays; MVR can be used to produce 3D images of virtual scenes much faster than existing rendering methods.

4.2 Image-based rendering

Another use for perspective image sets is as input to image-based rendering algorithms. Image-based rendering densely samples light traveling through a space as a set of images and transforms this data to produce new images seen from viewpoints spatially disparate from any of the originals. Image algorithms such as those developed by Gortler *et. al.* [8] and Levoy and Hanrahan [14] produce a single output image from a perspective image set, while similar algorithms developed for optical predistortion in synthetic holography derive not just one but an entirely new set of images [10]. Either of these types of image-based rendering algorithms requires a set of rendered perspectives in order to image a synthetic scene. MVR serves as a prefilter for the image-based rendering pipeline, transforming scene geometry into a basis set of the light field from which new images are derived. Perspective coherence, in turn, provides the means to efficiently compute this perspective image set.

5 Camera geometries

The exact relationship between a change in camera viewpoint and the resulting change in the appearance of a scene depends on the capture camera geometry. Choice of camera geometry depends on how the output data will be used and how well a particular geometry lends itself to efficient use of perspective coherence. Image-based rendering algorithms have shown that a sufficiently dense sampling perspectives in a single plane provides enough information to synthesize arbitrary perspectives within a volume free of occluders; this property permits the set of perspective images from one camera geometry to be converted to that of another, different camera geometry. The ability to convert sets of perspective images between different camera geometries allows the choice of a convenient geometry to maximize the use of perspective coherence to accelerate rendering. This paper will focus on a planar camera geometry specifically designed to simplify interpolation between different views.

6 PRS camera geometry

One of the simplest multi-perspective camera geometries consists of an planar array of cameras arranged in a regular grid, with all of the cameras' optical axes mutually parallel. The film plane of each camera in the grid is sheared in a plane orthogonal to the camera's view vector in order to recenter the image of points on an image plane located a constant distance from the camera plane. This camera geometry has been used in computer vision and synthetic holography since the late 1970's; it is identical to the one described by Levoy and Hanrahan [14].

We will refer to this geometry as a *planar regular shearing* camera geometry, or *PRS* camera. The one-dimensional analog of the PRS camera consists of a regularly spaced line of cameras; this geometry is called a *linear regular shearing* geometry, or *LRS*. The PRS camera can be decomposed into a set of simpler LRS cameras arranged in a regular array. Collectively, the PRS and LRS geometries are called *regular shearing* (*RS*) cameras. PRS and LRS camera geometries are shown in Figure 1.

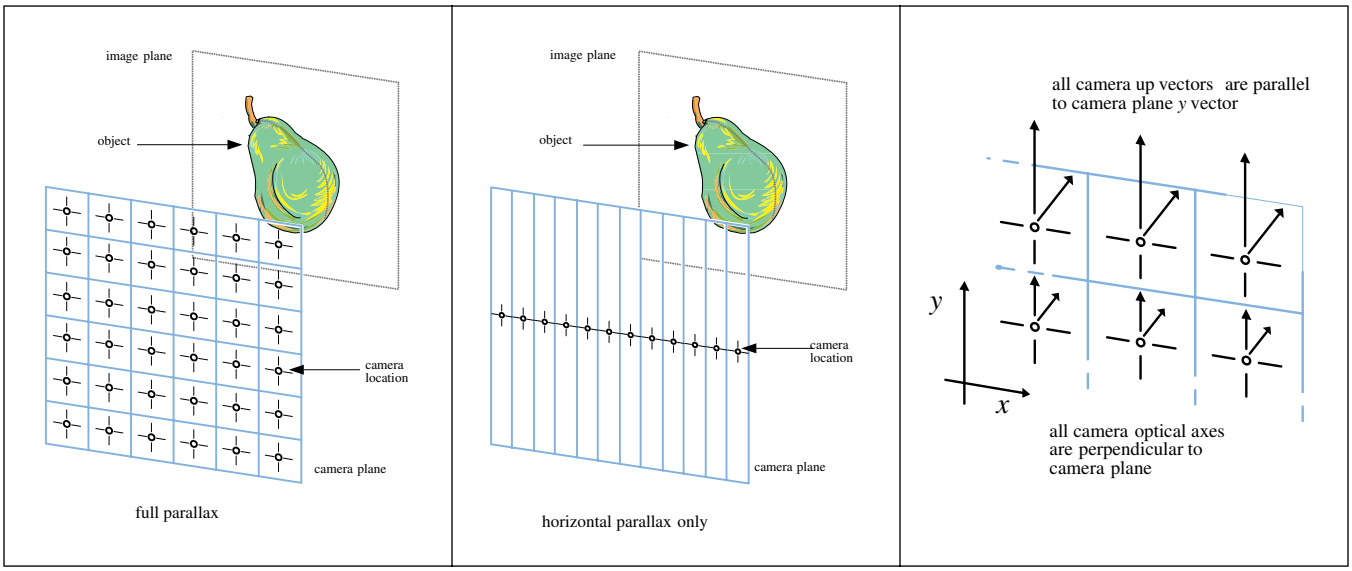

Figure 1: The left picture shows a two-dimensional array of cameras arranged in a PRS geometry, capturing an image of a three-dimensional object located at the recentering plane. The PRS camera geometry captures both vertical and horizontal parallax (full parallax information) of the object. The middle picture depicts a one-dimensional LRS camera geometry, which captures only horizontal parallax. A PRS camera can be created from a set of LRS camera positioned in a regular grid. The right picture shows a detail of the individual camera orientation for RS camera geometries: the cameras are positioned at regular grid locations, with their optical axes (view vectors) all parallel, and their film planes shifted so as to recenter the image plane in each view.

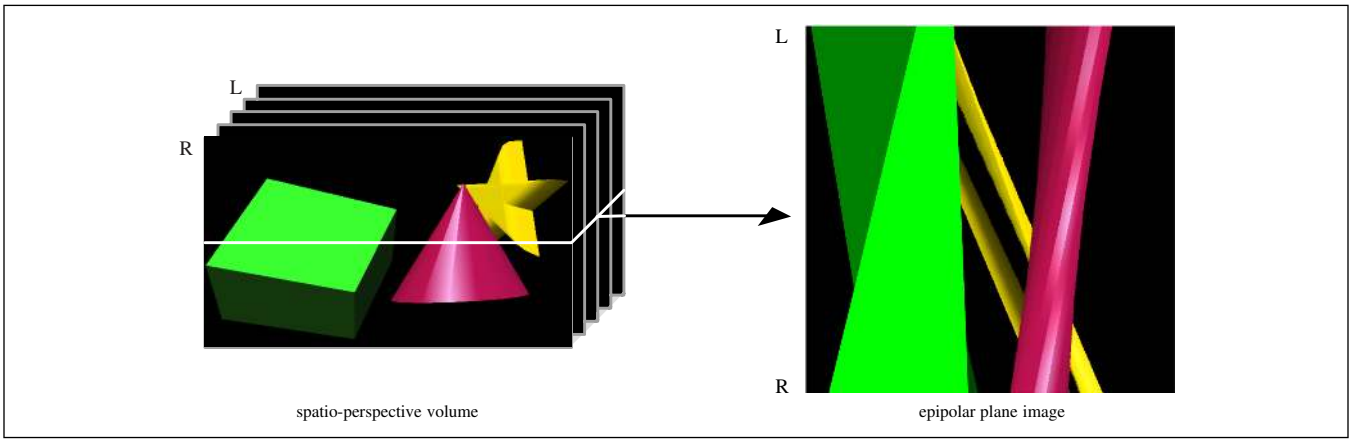

Figure 2: This spatio-perspective volume of a simple polygonal scene is formed by the frames captured by an LRS camera. An epipolar plane image (EPI) is a horizontal slice through this volume. In the scene, the cube and star are diffuse surfaces while the cone is shiny with a specular highlight.

The use of an RS camera introduces several simplifying constraints to the geometry and the mathematics of rendering multiple images. Assuming that a pinhole camera imaging model is used, an unoccluded point in the scene will translate in position from one camera image to the next at a velocity constant for all views and linearly proportional to the point's distance from the recentering plane and the spacing between the cameras in the grid. The relationship between camera position and the point's location in the corresponding image is separable. In other words, a point translating horizontally from one camera's image to another can only be due to a change in horizontal camera position (and similarly true in any other axis of lateral camera displacement). The separable linearity between camera location and image position is the key to maximizing perspective coherence.

7 Spatio-perspective image volume

Considered as a single unit, the set of perspective images from an RS camera form an image volume that spans a region of spatio-perspective space. The three-dimensional perspective image volume from an LRS camera is formed by stacking the individual camera images on top of each other like playing cards. A PRS camera forms an analogous four-dimensional volume. For the purposes of illustration, we will for the moment restrict our explanation to a LRS camera geometry where the camera is moving strictly horizontally.

The original perspectives of the RS camera are slices through the perspective image space. The volume can also be sliced in other ways. The computer vision community has used a construct known as an *epipolar plane image*, or *EPI*, to analyze the output of cameras arranged in (or moved through) a set of spatially disparate locations [4]. EPIs are slices of spatio-perspective space cut parallel to the direction of camera motion. The scanlines that make up EPI *n* are the *n*th scanline from each of the original camera views. Figure 2 shows the frames of a polygonal scene stacked up to form a spatio-perspective image volume. A horizontal slice through the volume at the location shown forms the EPI at right.

EPIs are useful because they expose the perspective coherence of the RS camera geometry. The linear relationship between camera position and the location of image detail manifests itself as linear features called tracks in the EPI. A point in the scene, for instance, sweeps out a linear *point track* in the EPI. Tracks are visible in the EPI as long as objects are visible and in frame as seen from a particular camera location. If an object is occluded by another object in one viewpoint, its track will be correspondingly occluded by the other point's track in the EPI. Since points in a scene tend to remain visible over a range of viewing locations, tracks tend to be fairly long in EPI space.

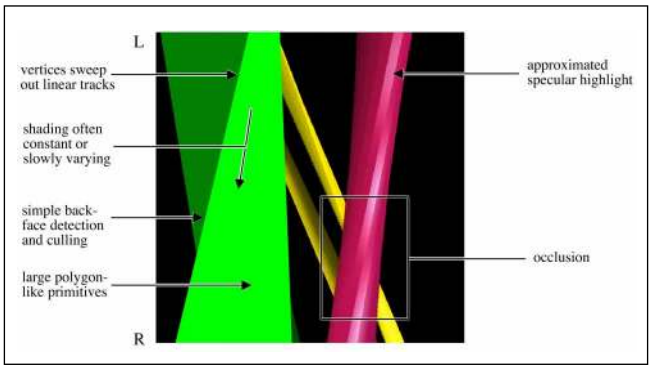

Figure 3: This figure shows some of the graphical properties of an LRS EPI. The large size of the track primitives, the regularity of the linear features, and the assimilation of shading make EPIs appealing to render. Note that specular highlights are not attached to the surface upon which they appear, but instead have their own slope. Since the renderer that produced this image implements Phong lighting but not Phong shading, the specular highlight on the cone is approximate.

Surfaces in the spatio-perspective volume swept out by lines in the scene are called *line tracks*. Line tracks are twisted quadrilaterals that interpolate between the point tracks of the line segment's two endpoints. A line in the scene that lies in an *epipolar plane* (a plane that includes the line of the camera track and a horizontal scanline) has a line track restricted to a single EPI. The EPI of the line track is formed by projecting its twisted 3D shape into 2D. A line track can occlude or intersect the tracks of other objects in the scene, or even twist itself into a bowtie shape as seen in 2D projection. The occlusion relationship between two different line tracks can be determined by interpolating the depth coordinate of the two endpoint tracks for each line track and occluding the more distant of the two line tracks at every point. These occlusion calculations are very similar to those performed in conventional single viewpoint rendering.

8 Properties of EPIs

The simple EPI shown in Figure 2 demonstrates some of the reasons why EPIs are useful for rendering. Linear track features are

compatible with interpolation algorithms implemented in conventional rendering software and hardware. The shape of line tracks are similar to, but even more regular than, the shape of the polygons that are frequently used to describe the geometry of scenes. Tracks are usually very long, spanning many pixels, so that the ratio of pixels spanned by a track versus the vertices needed to describe it is high. Occlusion relationships are essentially the same as in ordinary scenes. The color of objects tends to change slowly with varying viewpoint. Figure 3 shows examples of these and other graphical properties of objects in the EPI from Figure 2.

9 MVR rendering algorithm

The length of the tracks of geometric primitives in spatio-perspective space hints that EPIs of a scene contain more coherence than conventional perspective views of the same scene. This observation, combined with the other graphical properties of LRS EPIs, leads to the basic idea of the MVR algorithm: decompose a geometric scene into primitives that are rendered efficiently into EPIs, then render the spatio-perspective volume, EPI by EPI, until the entire volume is computed. To render a four-dimensional PRS spatio-perspective volume, decompose it into simpler three-dimensional LRS subvolumes that can be rendered individually. (A more sophisticated algorithm could render the 4D spatio-perspective volume as a single unit.) In many ways, MVR can be thought of as a higher-dimensional version of established scan conversion algorithms.

The basic steps of the MVR pipeline are as follows:

Preprocessing and transformation:

- Perform initial scene transformation and view independent lighting calculations for each vertex in the scene,
- Decompose the two-dimensional PRS camera geometry into a set of simpler horizontal LRS cameras,
- For each LRS camera, transform the vertices of the original scene geometry to find its position as seen from the two most extreme camera viewpoints,

Geometric slicing:

- Decompose the scene polygons into horizontal slices that lie along the scanlines of the final image,
- Sort these polygon slices by scanline into a scanline slice table,

Rasterization and hidden surface removal:

- For each scanline entry in the slice table, scan convert the slices for that scanline into tracks in EPI space, performing view-dependent shading calculations and hidden surface removal in the process,
- Combine all EPIs from all LRS cameras into a complete spatio-perspective volume.

The rest of this section describes more specific details of the different stages of the MVR rendering pipeline.

9.1 Preprocessing and transformation

Several of the computational steps used to calculate the appearance of a single image in conventional rendering can be performed once for the entire set of perspective images in multiple viewpoint rendering. For a Gouraud-shaded object, for instance, view independent lighting calculations such as Lambertian reflection can be performed once at each vertex of the scene. The cost of these lighting calculations is independent of the number of views to be rendered; their computational expense is amortized over the entire set of images.

Per-sequence MVR calculations proceed as follows. The geometry of the scene is either read from disk or accessed from memory. For each LRS image set to be rendered, the model is transformed into homogeneous screen space as seen from the two extreme camera viewpoints. (This paper uses a right-handed coordinate system where the x coordinate increases to the right of the screen, y increases up, and z increases out of the screen.) These two camera views differ only in the horizontal direction; because of the RS camera geometry, a vertex seen from these two viewpoints differs strictly in its x coordinate. The redundancy of the calculation means that the cost of performing both endpoint camera transformations is only 1.25 times the cost of performing a single transformation. Following this step, each vertex will have screen space coordinates (x_L, x_R, y, z, w), where x_L and x_R are the x coordinates of the vertex as seen from the extreme left and right camera views.

Next, clipping is performed on the transformed vertex coordinates. Polygons lying completely above or below the view window can be culled, as can those outside the near and far clipping planes. Clipping polygons that partially fall within the view window of at least one view is straightforward but somewhat more complicated than for SVR. In our prototype implementation, we chose to perform no polygon clipping at this stage for simplicity, while risking some performance penalty.

As another result of the RS camera geometry, both the z and w screen-space coordinates of each vertex remain constant over the entire range of viewpoints. Because w is fixed, the homogeneous divide required for perspective transformation are performed as a preprocessing step, reducing the computational cost of transformation. The w coordinate should be maintained, however, to permit perspective-correct shading and texture interpolation during scan conversion [3].

9.2 Geometric slicing

Polygon tracks, or *PT*s, are three-dimensional volume primitives in the spatio-perspective space of an LRS camera geometry. Two-dimensional scan conversion of a PT requires the original polygon be decomposed, or sliced, into a set of line segments that lie along the scanlines of the image set's final perspective images. These line segments are called *polygon slices*.

Slicing a polygon along scanlines is very similar to conventional scan conversion, except that the slices retain their continuous three-dimensionality. As each horizontal slice is generated, a data structure describing it is added to a scanline slice table. This table has one entry per scanline: each entry is a list of polygon slices that the scanline intersects. Thus, each scanline table entry contains all the information needed to render the scanline's EPI, which is in turn the scanline's appearance from all viewpoints.

9.3 Rasterization

When it is rendered, each polygon slice is converted into a special kind of line track called a *polygon slice track* (*PST*) and rendered into the appropriate EPI. The PST is the key rendering primitive of MVR. Figure 4 shows some of the PST shapes from which different slice orientations can result. PSTs have two types of edges. *P-edges* (projection edges) are the projections of the polygon slice as seen from two most extreme camera views. *I-edges* (interpolating edges) interpolate the position of a slice endpoint through the range of views. The geometry of PSTs is very regular; without clipping or culling and disregarding degenerate cases, p-edges are always horizontal and lie at the top and bottom scanline of the EPI, while i-edges cross all EPI scanlines.

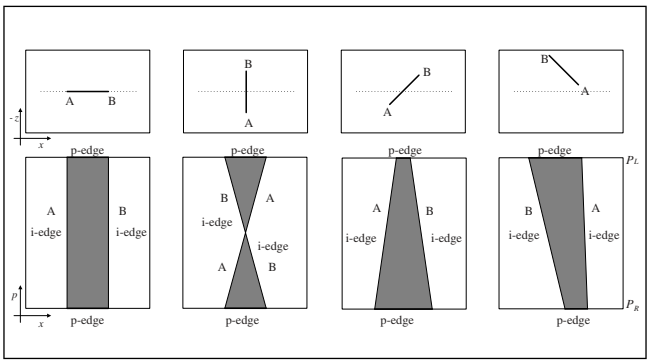

Figure 4: Polygon slice tracks (PSTs) can have a variety of shapes depending on the orientation of the corresponding polygon with respect to the image plane. P-edges are the edges of the PST that are projected onto the camera plane; i-edges interpolate the endpoints of the p-edges through a range of viewpoints. This figure shows several polygon slice orientations (top) and the corresponding PST shape (bottom). The dotted line represents the image or recentering plane of the capture camera.

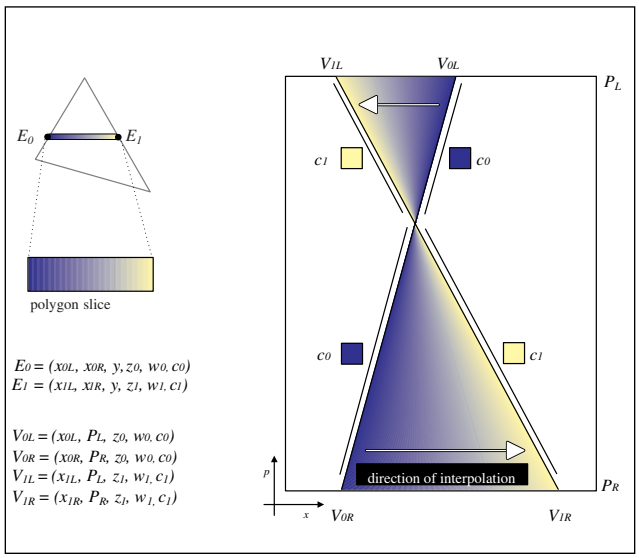

Figure 5: The picture on the left shows a slice extracted from a triangle from the original scene geometry. The coordinates E_0 and E_1 have been interpolated from the triangle's vertices and represent the homogeneous location of the vertex (x_L, x_R, y, z, w) and the view independent per-vertex color (c) calculated there. From these endpoint coordinates, the vertices of the PST V_{0L}, V_{0R}, V_{1L}, and V_{1R} are found. PST rasterization requires linear interpolation of geometric and Gouraud shading parameters in the horizontal direction.

Figure 5 shows how the coordinates of the PST vertices in spatio-perspective space are derived from the endpoints of the polygon slice in screen space. Rasterization of a PST of a diffuse Gouraud-shaded polygon proceeds by interpolating the perspective coordinate in the vertical (p) direction, and all other parameters in the horizontal (x) direction. Every horizontal scanline of the PST crosses through the same range of values for each parameter, but at a different rate of sampling depending on the width of the PST at that scanline. If the i-edges of a PST cross at a point, the direction of interpolation will be opposite from one side of the crossing point to the other, as the figure demonstrates.

Geometric slicing and PST rasterization most distinguish the MVR rendering process from that of a more conventional renderer. By way of example, Figure 6 shows how a single triangle is rendered using MVR and an LRS camera geometry.

9.4 Hidden surface removal

Hidden surface removal (HSR) is performed in spatio-perspective space in the same way it would be performed in image space, and many of the algorithms for HSR can be adapted to work on PSTs instead of polygons. The widely-used Z-buffer algorithm for HSR can be easily implemented by storing a depth value for each pixel in an EPI, and comparing the interpolated depth value for each pixel of the PST being rasterized to see if it is in front of all previous surfaces.

Some aspects of HSR can be simplified in MVR by using the inherent perspective coherence of the scene. Backface culling, for instance, can be performed once per PST instead of once per polygon per viewpoint by observing that the orientation of a polygon slice with respect to the camera changes slowly and predictably. If the i-edges of a PST do not intersect, a backfacing test is required only once for the entire PST, accepting or rejecting it as a whole. If a PST does cross itself, one of the triangles of the PST is back facing and the other is front facing. The back facing piece of the PST need not be rendered.

9.5 Texture mapping

Texture mapping is a type of shading that applies image detail to the surfaces of geometric objects. The appearance of texture maps is view independent: while the geometry onto which the texture is mapped may change depending on the location of the viewer, the appearance of the texture itself does not. Other types of image-level mapping algorithms such as reflection or environment mapping are not view independent; a reflection on a surface can change in appearance as the viewer moves around it. Figure 7 shows a simple polygonal scene with both texture and reflection maps applied to different objects. The corresponding EPI shows how the texture mapped onto the cube and star changes gradually over the entire range of views, in contrast to the less predictable reflection mapped cone. The view independence of texture mapping lends itself to an efficient MVR implementation.

In its simplest form, MVR texture mapping is similar to the analogous algorithms in SVR. Texture coordinates are assigned to each vertex of the original scene geometry, hyperbolic texture coordinates[3] are interpolated to find the texture of each slice endpoint, and textures are further interpolated across the surface of the PST. This simple texturing technique can be used when adapting existing rendering algorithms to render PSTs.

When rendering a large number of views, a more efficient MVR texture map algorithm can be implemented using the fact that each horizontal line of a PST is a resampling of the same texture at a different rate. Figure 8 outlines the steps of this algorithm. The texture for each polygon slice is extracted from two-dimensional texture memory using MIP [21] or area sampling techniques and stored in a one-dimensional texture map at a sampling rate appropriate for the slice's greatest width under transformation. Non-linear sampling of the two-dimensional texture takes care of distortions due to perspective: further resampling of the one-dimensional texture can be performed linearly without need for complex and expensive hyperbolic interpolation when rendering every pixel.

This 1D map is the basis for a new MIP map that is applied to different regions of the PST. Using the 1D map has several advantages over existing texturing algorithms. The width of a PST changes slowly and regularly, so the appropriate level of the MIP map needed to avoid sampling artifacts can readily be chosen. 2D texture memory is probed in a more predictable way, improving

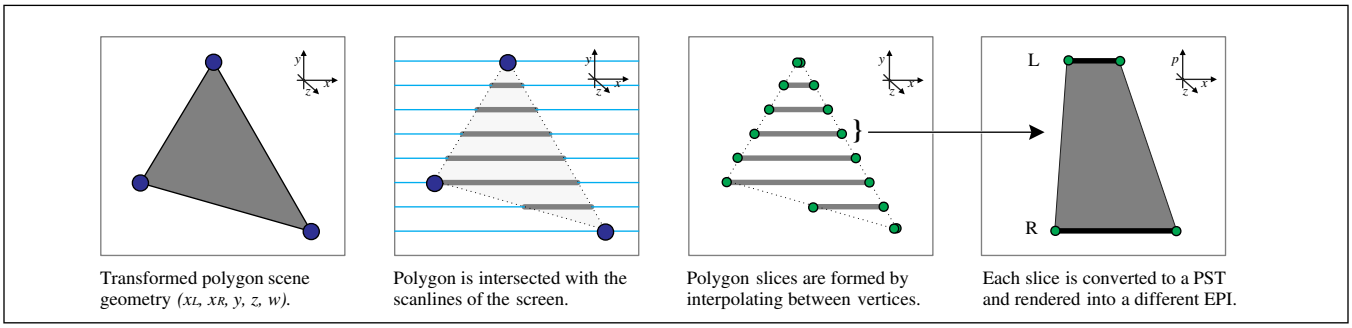

Figure 6: The basic MVR rendering pipeline for the LRS camera geometry, applied to rendering a single triangle.

texture prefetching strategies. The size of the 1D map is small enough to allow efficient caching in fast memory, where the texture of a PST pixel can be computed using linear indexing and simple interpolation.These simple operations suggest the possible use of specialized hardware including image warping subsystems to perform MVR texturing.

9.6 View-dependent shading

View dependent shading of surfaces in MVR can, like texture mapping, be implemented using modified SVR algorithms. Reflection and environment mapping are the most common view dependent shading algorithms in current use. Reflection algorithms calculate reflection vectors at each polygon vertex based on eye, light, and surface orientation vectors, interpolating the reflection vectors across the polygon, and perform a lookup into a reflection map using the interpolated vector. MVR reflection mapping works the same way, except that the eye vector can potentially vary over a large angle through the range of camera positions. Figure 9 describes the relationship between camera geometry and the assignment of reflection vectors to a PST.

When implementing reflection mapping in MVR, care must be taken to assure that reflection calculations are accurate over a large angular change in eye vectors. For example, spherical reflection mapping [16] substitutes complex spherical interpolation with simpler linear interpolation across the extent of polygons. Over large angles, this approximation becomes invalid and results in noticeably incorrect shading of PSTs. One recourse to solve this problem with spherical maps is to uniformly subdivide PSTs in the perspective dimension at the cost of performance. Cubic reflection maps, on the other hand, correctly interpolate reflection vectors and can be used without subdivision.

Although view dependent shading does not attain the level of efficiency in MVR as does view independent shading, it can still be more efficient than a comparable SVR algorithm. Computational savings result from the precalculation and incremental interpolation of reflection vectors over PSTs and the regularity of the reflection vector's mapping into the reflection map memory.

10 Implementation

The MVR algorithm described in this paper generates a set of perspective images from cameras arranged in a PRS camera geometry, using computer graphics hardware to accelerate the rendering process. The prototype implementation described here is designed to fairly compare the relative efficiency of MVR and conventional SVR algorithms. The implementation consists of shared modules for file input and output, scene transformation, and texturing and reflection mapping, as well as MVR or SVR-specific modules for primitive generation, rendering and image

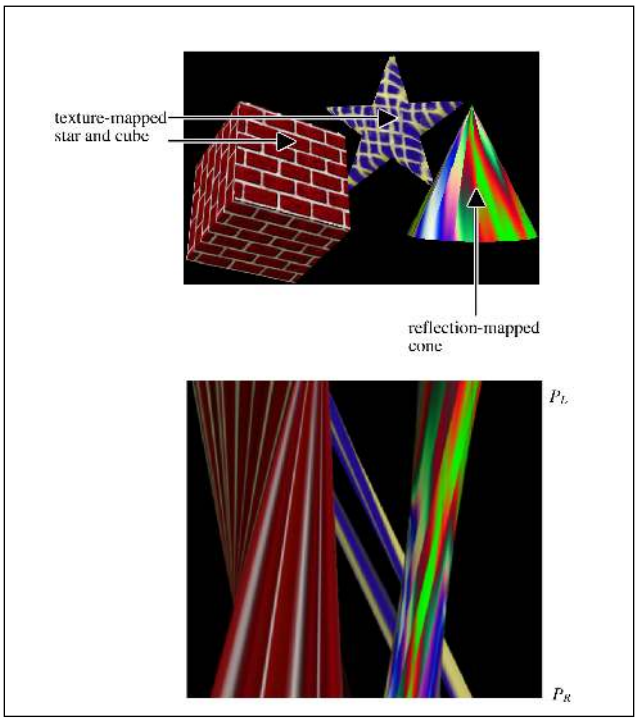

Figure 7: The cube and star in this polygonal scene are texture mapped, while the cone is reflection mapped. The EPI of a scanline of the scene is extracted and shown in the lower picture.

assembly. The code for the implementation is written in ANSI C and uses the OpenGL™ graphics library to provide device-independent graphics acceleration. The algorithm has been tested on a range of workstations from Silicon Graphics Inc., including an Indigo² workstation with Maximum Impact graphics and a 150 MHz R4400 CPU, and an Onyx with RealityEngine² graphics and two 150 MHz R4400 CPUs. Further tests were done using a Sun Microsystems Ultra 1 workstation with Creator3D graphics.

Input data for the tests consisted of two polygonal models: one of a teacup, the other of a Ferio automobile body shell provided by the Honda R&D Company (Figure 10). These scenes were created using Alias/Wavefront Corporation's Alias Studio modeling program. From original surface models, Alias Studio calculates per-vertex view independent lighting and texture coordinate values, and outputs a collection of independent triangles to a file. This triangle data eliminates the need for view independent lighting to be implemented in the rendering testbed itself. In

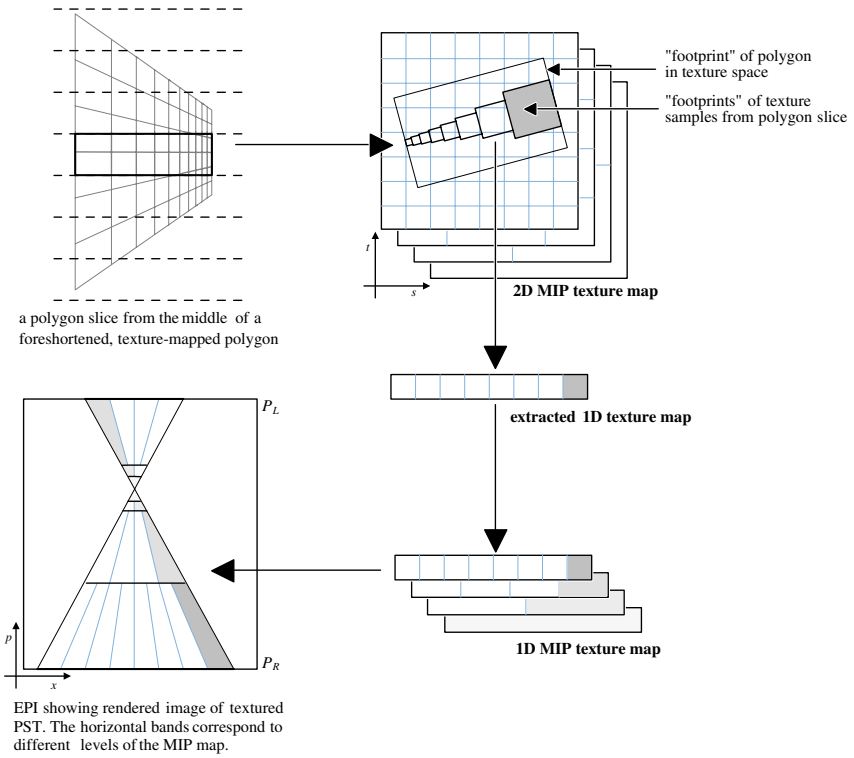

Figure 8: An MVR-specific algorithm for texture mapping extracts the texture for a polygon slice from two-dimensional memory, builds a 1D MIP map, and repeatedly resamples it to apply the texture to scanlines of the PST. This process eliminates the need for a per-pixel homogeneous divide.

addition the teacup was tessellated to produce 4K, 16K, 63K, 99K, and 143K triangle count models in order to compare the efficiency of MVR when rendering polygons of different average sizes.

10.1 MVR renderer

The MVR module consists of a polygon slicer, a scanline slice table, a slice-to-PST converter, and an EPI rasterizer. PSTs are rendered in hardware by approximating them as polygons. Although PST shading and interpolation is actually easier than shading triangles because of their regular geometry, current rendering hardware designed for optimized triangle rendering produces shading artifacts across the PST (and many other quadrilaterals [22]). Such errors can be reduced by minimizing the size of polygons, reducing the range of viewpoints, or decomposing the PST into smaller primitives. During testing, a minimal decomposition of PSTs into no more than four triangles was used for both timing and rendering accuracy tests. A hardware rendering system specifically designed for MVR could provide higher quality rendering at rates faster than the fastest rates described here.

Texturing and reflection mapping were implemented in the MVR module without using MVR-specific algorithms in order to use existing graphics hardware to accelerate these operations. Reflection mapping uses spherical environment maps to simulate surrounding objects and Phong-like specular highlights. Figure 11 shows a two-dimensional array of images rendered using MVR with both texture and reflection mapping. Figure 12 is an EPI from the teacup, from a scanline near the cup's lip.

10.2 SVR renderer

The SVR module was designed to minimize redundant operations consistent with rendering a set of images. For instance, the initial transformation of the scene triangles, performed on the CPU and not in the graphics engine, is done only once for all views. The graphics hardware's transformation matrices were not updated from view to view for the SVR speed tests: only the image of the central view was used as an approximation of the per-frame rendering speed. If anything, this approximation should underestimate the rendering time for the SVR algorithm. For both the SVR and MVR modules, speed tests do not include the time required to read back data from framebuffer memory. Applications that use a set of perspective images are almost certain to need the rendered images as data in main memory, not just on the screen, but reading framebuffer memory requires approximately the same time in either SVR or MVR.

The next section presents the results of speed and rendering accuracy tests performed using the prototype rendering implementation.

11 Performance

11.1 Timing tests

The graph in Figure 13 shows the performance of the different stages of the MVR pipeline when rendering the Ferio database at a resolution of 640 by 480 pixels per view over a varying number of views of an LRS camera. Only view independent shading was performed for this test. Timings were performed on the SGI Indigo^2. The cost of reading triangles, transforming them (includ-

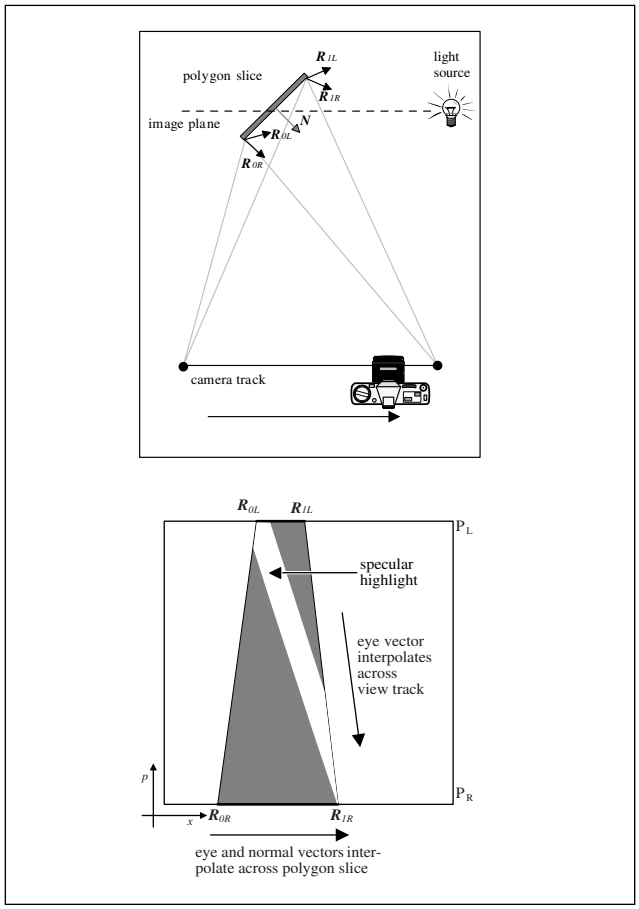

*Figure 9: The top figure shows an LRS camera viewing a polygon slice with a surface normal **N** across its surface. The slice endpoints each have two reflection vectors **R** that result from view vectors from the two camera track endpoints. These four reflection vectors are assigned to the PST shown in the EPI in the bottom figure.*

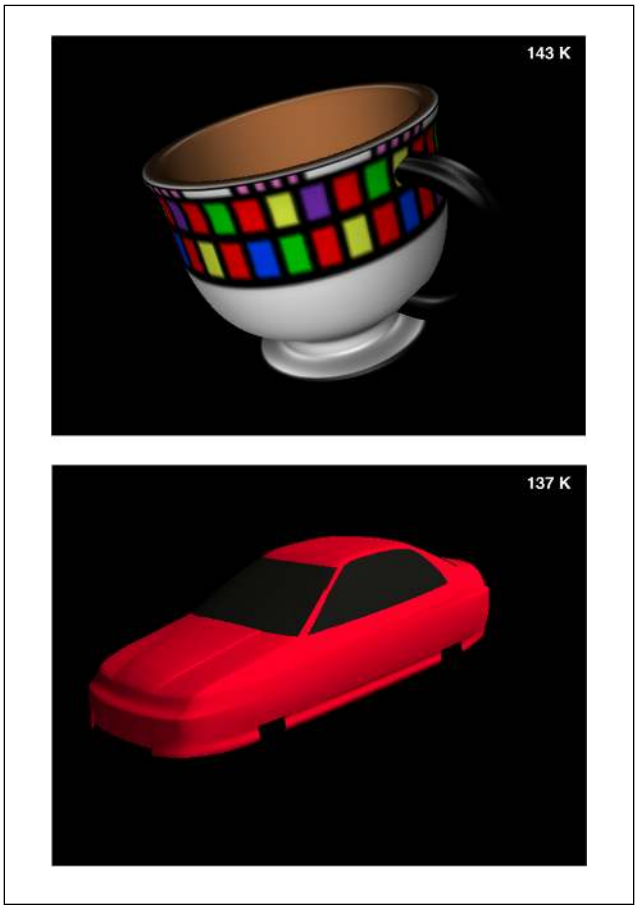

Figure 10: These objects are the test data for the prototype renderer. Numbers indicate the triangle count of the two models shown here. These images are extracted from a set of MVR-rendered perspective images.

ing conventional transformation and the additional cost of calculating horizontally and vertically varying parallax information), and slicing the polygons is constant. The cost of rendering less than about 400 primitives is also constant: the hardware setup time for the PSTs is greater than the cost of concurrently painting the pixels of the PSTs onto the screen. The line on the far left side of the graph shows the relative cost of rendering using the SVR module. In this test, MVR is more efficient than SVR for images sets larger than about 10 views.

Figure 14 directly compares the time required to render scenes of different tessellation using both MVR and SVR. The teacup models of different tessellation densities were rendered using both algorithms using the SGI Onyx. The graph shows that the smaller the polygons, the better MVR performed relative to SVR. This behavior is due to the increased spatial coherence in the smaller polygon scenes: SVR better amortizes per-polygon setup costs over a larger number of pixels drawn to the screen. At best case for this resolution, MVR is about 26 times faster than SVR.

Figure 15 shows a comparison between SVR and MVR using a polygon database of fixed size, but with a varying pixel resolution of the output images. The Indigo^2 system was used to render the Ferio database for this test. In the SVR timing results, the hardware is not pixel fill rate limited at any of the resolutions. Thus, rendering times are independent of the pixel size of the image. MVR performance is, on the other hand, dependent on image resolution. Smaller images result in fewer PSTs to render. At the lowest image resolution, for example, MVR is more than 200 times faster than SVR. These low-resolution images have application in three-dimensional display devices, where light modulators may have a low pixel count.

The shape of the MVR timing curves reflects the different types of cost savings when rendering different numbers of views. For a small view count, geometry costs dominate pixel fill and adding more views are essentially free. The knee of the curve represents the point where the cost of the concurrently-performed geometry and fill operations are equal. The slope to the right of the knee of the curve levels off as the costs of sequential preprocessing operations are amortized over increasingly many views.

11.2 Rendering accuracy

Ideally in the most common case, there should be no difference between the image sets rendered using MVR and those rendered conventionally with MVR. However, some errors in shading may occur because of algorithm-dependent differences in rasterization or shading. To test the accuracy of MVR, the 143K teacup was used as a model for both MVR and SVR to render sixteen views at 640 by 480 pixels. From these two collections of

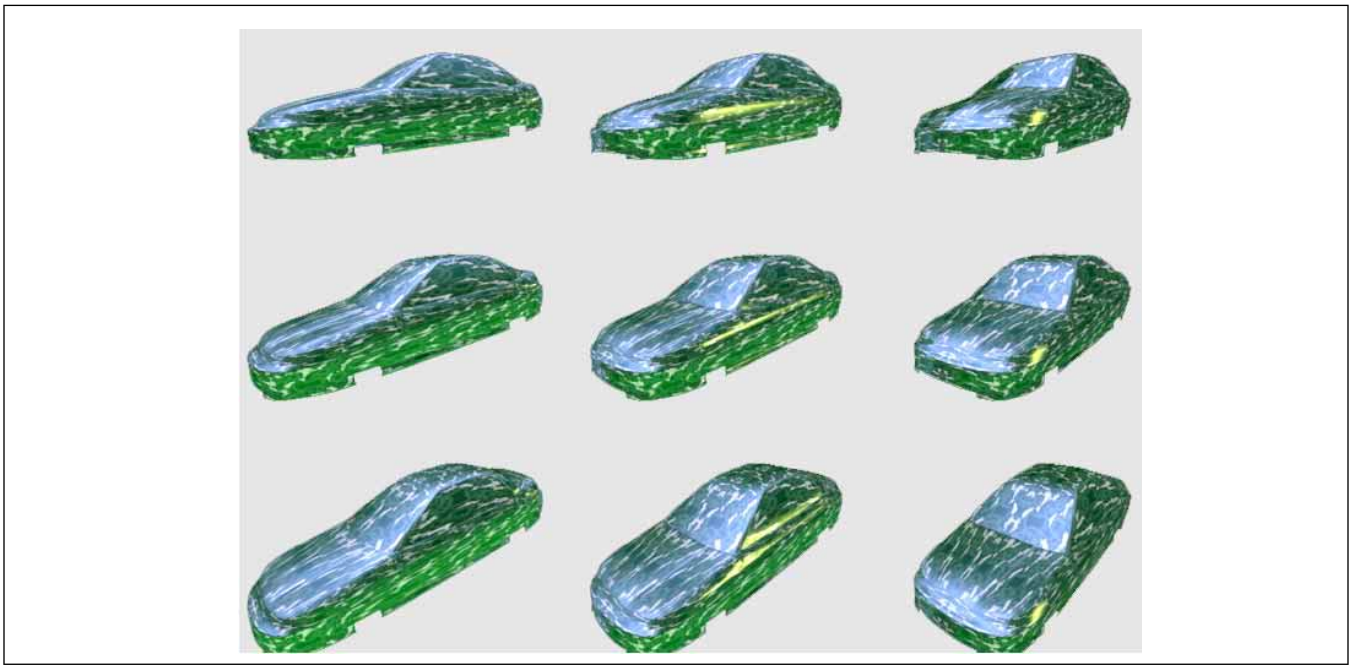

Figure 11: A set of texture mapped and reflection mapped images computed using MVR.

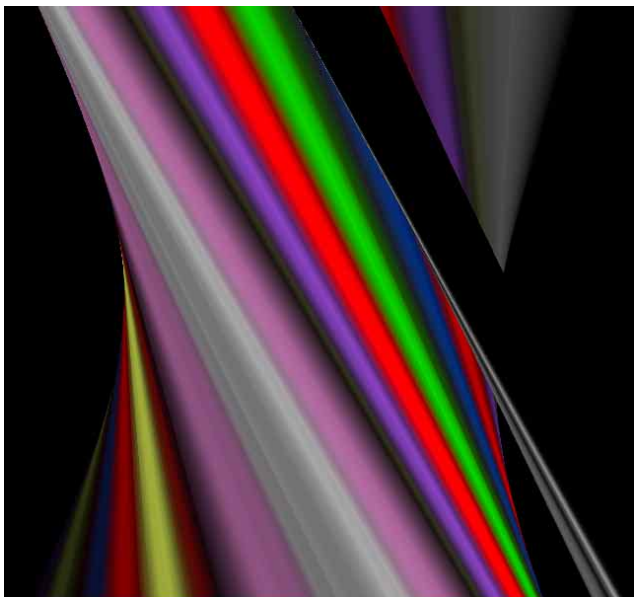

Figure 12: An EPI of the teacup.

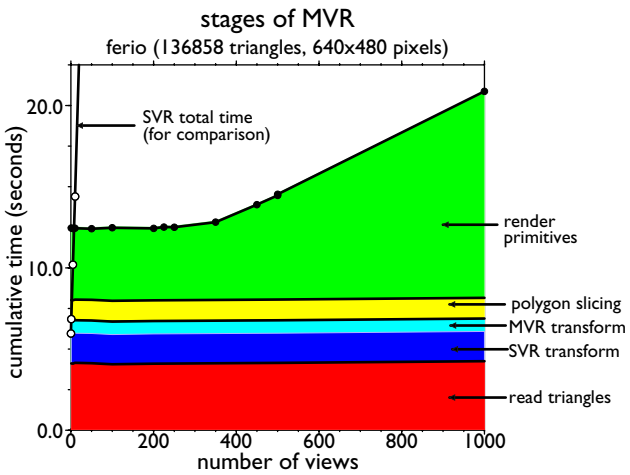

Figure 13: Relative costs of the different stages of MVR, using an LRS camera, when rendering different numbers of views. Total rendering time of SVR is included for reference.

views, four pairs were extracted, and the absolute values of their pixel-by-pixel differences computed. Rendering was done using the Sun Ultra 1. No PST subdivision was used when rendering using MVR.

The result of this difference is shown in Figure 16. An enlarged piece of the error image is shown in Figure 17. The largest errors located along the edge of the cup are most likely due to small differences in transformation between the two rendering modules; these differences are never more than one pixel wide in any view. Errors on the interior of the teacup result from differences in shading between the two algorithms. Neither error is generally noticeable in practice.

When rendering small numbers of widely disparate views, however, shading differences between MVR and SVR can be significant. The reason for this difference is SVR makes no guarantees that the track of a object in spatio-perspective space is continuous, while MVR does; the track of any MVR-rendered feature is bandlimited so that the images of the feature abut from view to view. The MVR behavior, while different than that of SVR, provides sufficient sampling to avoid aliasing artifacts in image-based rendering and synthetic holographic displays [12].

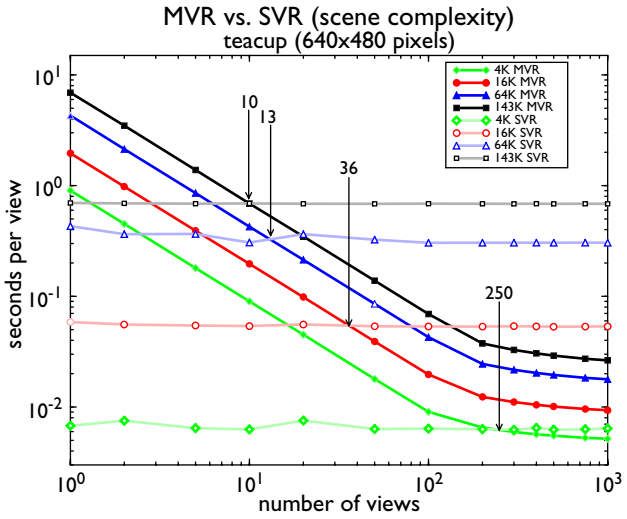

Figure 14: This graph compares the performance of SVR and MVR algorithms while rendering the same scene at different tessellation densities.The arrows in the graph show the "break even" points where SVR and MVR take the same amount of time to render the image volume.

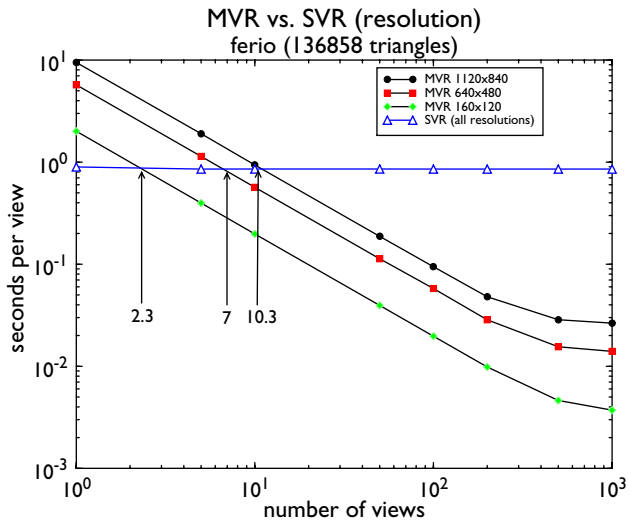

Figure 15: The graph compares SVR and MVR performance at different image resolutions.

11.3 Interpreting the results

These results for scenes with view independent shading demonstrate that MVR is capable of exceeding the performance of SVR algorithms by one to two orders of magnitude. Further testing confirms that these savings are also true for texture- and reflection-mapped scenes, and for PRS cameras constructed from multiple LRS cameras. MVR is faster than SVR for rendering large sets of perspective images for several reasons. First, a significant number of transformation and shading operations are performed as preprocessing steps, incurring a constant cost that is amortized over the entire set of images. In the RS camera geometry, this preprocessing can include the otherwise-costly homogeneous divide required during perspective transformations.

Second, the ratio of the pixel size of rendered primitives to the number of vertices that describe those primitives' geometry is much higher for PSTs in MVR than for ordinary polygons when rendering many viewpoints. Since rendering hardware often uses more expensive floating point representations to describe and transform vertices, and fixed-point or integer calculations to deal with pixels, improving the pixel-to-vertex ratio of geometric primitives can often lead to dramatic improvements in performance. Many other techniques exist for changing the pixel-to-vertex ratio, including building geometry strips and compressing the scene's geometric description [7]. The use of these techniques and the tuning of software and hardware that makes up a specific rendering pipeline can control whether rendering of a given scene is geometry or pixel fill limited. MVR is an additional technique that shifts the balance towards high pixel-to-vertex ratios; it can also be combined with other techniques such as geometry strips or compression to achieve still more pixels per vertex.

Third, shading and texturing PSTs is less complex than the equivalent operations on polygons. PSTs have a more regular shape and size than do polygons from the same scene. PSTs can be shaded and textured using only horizontal interpolation between i-edges for each scanline of a PST. Perspective-correct texture mapping can be performed using only linear resampling of a perspective-predistorted subtexture, eliminating a per-pixel divide, improving memory access and cache performance, and simplifying possible hardware implementation. Backface culling and hidden surface removal can both be implemented to take advantage of perspective coherence.

The exact impact of these cost-saving properties on the time required to render a perspective image set depends on the architecture of the computer rendering subsystem (including the relative costs of vertex operations, pixel fills, memory access and communication), the resolution and count of the output images, and the properties of the particular scene being rendered. A graphics system with a very limited pixel fill rate, for example, may experience little or no savings from MVR. The following rule can be used to determine the general applicability of MVR to a particular application: if the height in pixels of an average polygon in a scene is smaller than the number of viewpoints to be rendered, MVR will likely be as fast or faster than an SVR algorithm. For this number of views, the pixel-to-vertex ratio for SVR and MVR is approximately equal.

12 Comparison to other work

Several other researchers have developed computer graphics algorithms that use some form of frame-to-frame coherence. Badt [2], Chapman *et. al.* [5], and Groeller and Purgathofer [9] have produced ray-tracing algorithms that use temporal coherence to improve multi-frame rendering performance. Each of these algorithms produce modest computational savings over conventional ray-tracing techniques. Tost and Brunet characterized a variety of frame coherent algorithms in a 1990 taxonomy[20]. Adelson *et. al.* [1] used perspective coherence to compute pairs of images for stereoscopic displays. Because their algorithm only computes pairs of images, only limited acceleration due to perspective coherence is possible.

The computer vision and image processing fields have used epipolar plane image analysis as a way to interpolate intermediate viewpoints from a set of photographically acquired images. Takahashi *et. al.* [18] have used these methods to generate images

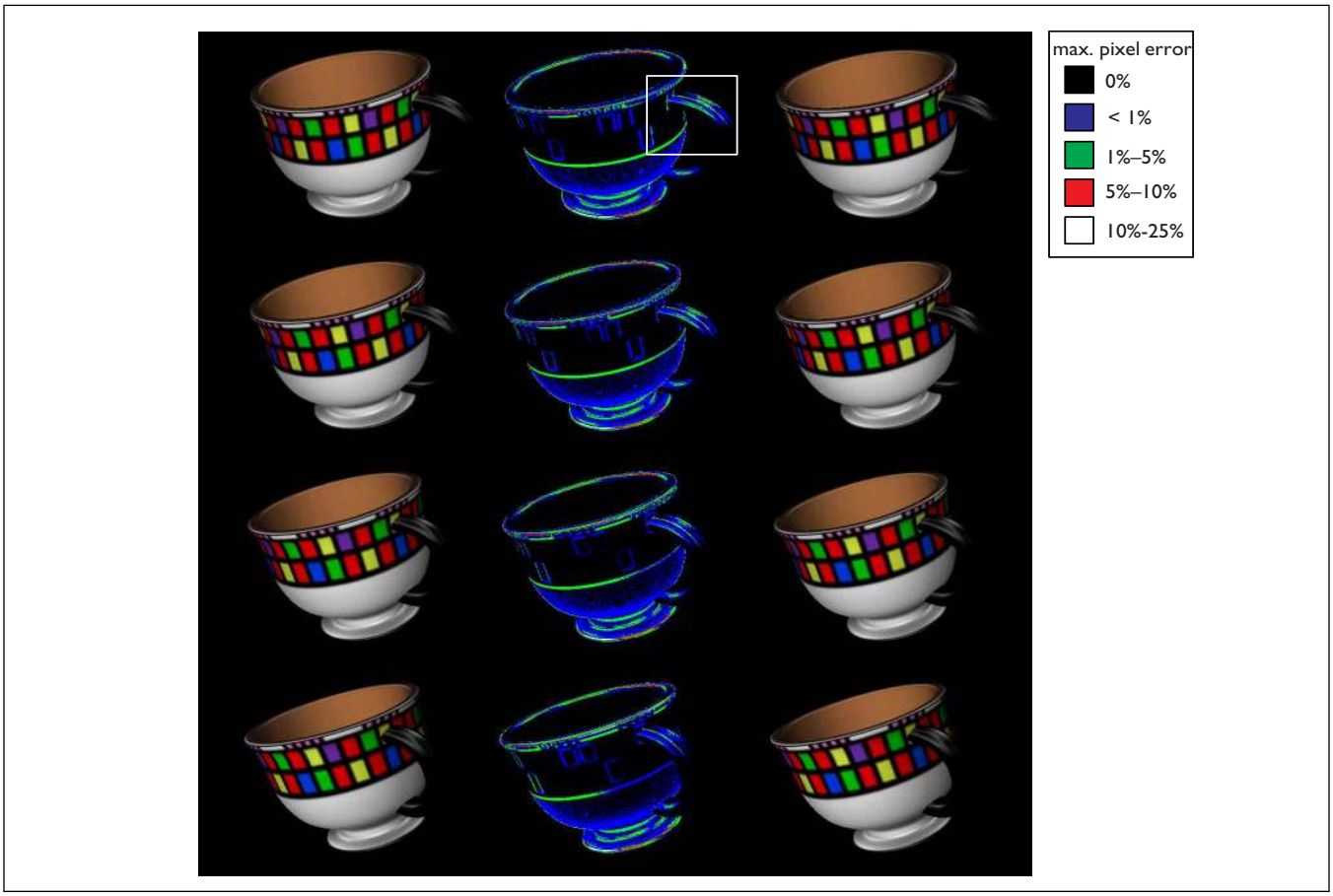

Figure 16: The four images on the left side of the figure were calculated using conventional SVR rendering techniques. The set on the right was rendered using MVR, with the four images extracted from a set of sixteen. In the middle is the per-pixel absolute value of the difference between the two sets (as a percentage of the maximum possible difference), colorized to more clearly show error regions. The rectangular area is enlarged in Figure 17.

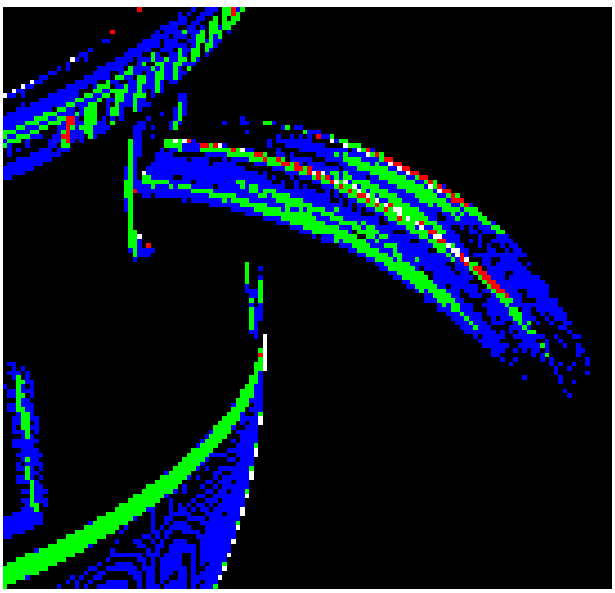

Figure 17: An enlargement of the difference image from Figure 16. The blue and green interior areas are the result of shading differences, while the broken red and white lines are probably the result of slight misregistration of geometry.

for holographic stereograms. Image interpolation of this kind requires finding corresponding points in different images (the underconstrained "correspondence problem" of computer vision).

Hybrid computer graphics and image processing algorithms reduce the need to solve the correspondence problem by augmenting image information with more data from the original scene. Zeghers *et. al.* [23] use motion-compensated interpolation to produce a series of intermediate frames in an image sequence of fully computed frames. The disadvantages of this algorithm are the need to compute a motion field, and the loss of fine detail in the scene because of image space interpolation.

Chen and Williams [6] also use geometry information to guide viewpoint interpolation. Using known camera geometries, their algorithm builds an image space morph map from two images and depth buffers. This technique can be used to reduce the cost of shadow and motion blur generation, since intermediate images can be computed in time independent of the geometric complexity of the scene. They propose methods for reducing the "overlaps" and "holes" in the data between two images. These problems can only be minimized, not eliminated, however. The image space interpolation cannot correctly deal with anti-aliased input images, specular highlights, and other view dependent scene changes.

Instead of using a limited amount of scene information from an image space buffer, MVR interpolates intermediate views using the object precision of the original scene geometry. It incorporates

image interpolation as part of the rendering process without any need to deal with the difficult computer vision problems of correspondence. The cost of rendering intermediate views is dependent on the complexity of the scene geometry. However, MVR is more compatible with the interpolation hardware found in hardware graphics systems.

13 Conclusions and future work

New applications in computer graphics such as three-dimensional display and image-based rendering need large sets of perspective images as input. MVR extends conventional scanline rendering algorithms to provide these sets of images at rates one to two orders of magnitude faster than existing methods. MVR can generate high quality images using texture and reflection maps, and can be accelerated using both existing and future graphics hardware. Many rendering techniques not described here can be adapted for use with MVR. MVR techniques can also be extended to alternate camera geometries, geometry compression, and transmission of three-dimensional geometry information.

Acknowledgments

The work described in this paper was done as part of my doctoral dissertation at the Spatial Imaging Group of the MIT Media Laboratory. Thanks to everyone in the group who helped me think through these ideas and reviewed drafts of this paper, including Wendy Plesniak, Ravikanth Pappu, and John Underkoffler. Wendy Plesniak also modeled the teacup used in the figures. My advisor Stephen Benton and my thesis committee members V. Michael Bove and Seth Teller provided great comments and support. Ron Kikinis and Ferenc Jolesz at BWH supported me during the last year of this work, in part through funding from Robert Sproull at Sun Microsystems. Both Silicon Graphics and Sun Microsystems provided equipment used in this research. Also, thanks to the SIGGRAPH reviewers for their thoughtful comments and suggestions.

This work was funded in part by the Design Staff of the General Motors Corporation, the Honda R&D Company, IBM, NEC, and the Office of Naval Research (Grant N0014-96-11200).

References

[1] Stephen J. Adelson, Jeffrey B. Bentley, In Seok Chung, Larry F. Hodges, and Joseph Winograd. Simultaneous Generation of Stereoscopic Views. *Computer Graphics Forum*, 10(1):3-10, March 1991.

[2] Sig Badt, Jr. Two Algorithms for Taking Advantage of Temporal Coherence in Ray Tracing. *The Visual Computer*, 4(3):123-132, September 1988.

[3] James F. Blinn. Jim Blinn's corner: Hyperbolic Interpolation. *IEEE Computer Graphics and Applications*, 12(4):89-94, July 1992.

[4] R. C. Bolles, H. H. Baker, and D. H. Marimont. Epipolar-Plane Image Analysis: An Approach to Determining Structure from Motion. *Inter. J. Computer Vision*, 1:7-55, 1987.

[5] J. Chapman, T. W. Calvert, and J. Dill. Exploiting Temporal Coherence in Ray Tracing. In *Proceedings of Graphics Interface '90*, pages 196-204, May 1990.

[6] Shenchang Eric Chen and Lance Williams. View Interpolation for Image Synthesis. In James T. Kajiya, editor, *Computer Graphics* (SIGGRAPH 93 Proceedings), volume 27, pages 279-288, August 1993.

[7] Michael Deering. Geometry Compression. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 13–20, August 1995.

[8] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The Lumigraph. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 43-54, August 1996.

[9] E. Groeller and W. Purgathofer. Using Temporal and Spatial Coherence for Accelerating the Calculation of Animation Sequences. In Werner Purgathofer, editor, *Eurographics '91*, pages 103-113. North-Holland, September 1991.

[10] Michael W. Halle. Autostereoscopic Displays and Computer Graphics. In *Computer Graphics,* ACM SIGGRAPH. 31(2), pages 58-62.

[11] Michael W. Halle. The Generalized Holographic Stereogram. Master's thesis, Department of Architecture and Planning, Massachusetts Institute of Technology, February 1991.

[12] Michael W. Halle. Holographic Stereograms as Discrete Imaging Systems. In *Practical Holography VIII,* vol. 2176, pages 73–84, SPIE, May 1994.

[13] Michael W. Halle. Multiple Viewpoint Rendering for Autostereoscopic Displays. Ph.D. thesis, Media Arts and Sciences Section, Massachusetts Institute of Technology, May 1997.

[14] Marc Levoy and Pat Hanrahan. Light Field Rendering. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings, Annual Conference Series*, pages 31-42, August 1996.

[15] T. Okoshi. *Three-Dimensional Imaging Techniques*. Academic Press, New York, 1976.

[16] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul E. Haeberli. Fast Shadows and Lighting Effects using Texture Mapping. In Edwin E. Catmull, editor, *Computer Graphics* (SIGGRAPH 92 Proceedings), volume 26, pages 249-252, July 1992.

[17] Ivan E. Sutherland, Robert F. Sproull, and Robert A. Schumacker. A Characterization of Ten Hidden-Surface Algorithms. *Computing Surveys*, 6(1), March 1974.

[18] S. Takahashi, T. Honda, M. Yamaguchi, N. Ohyama, and F. Iwata. Generation of Intermediate Parallax-images for Holographic stereograms. In *Practical Holography VII: Imaging and Materials*, pages 2+, SPIE, 1993.

[19] Jay Torborg and Jim Kajiya. Talisman: Commodity Real-time 3D Graphics for the PC. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 353-364, August 1996.

[20] Daniele Tost and Pere Brunet. A Definition of Frame-To-Frame Coherence. In N. Magnenat-Thalmann and D. Thalmann, editors, *Computer Animation '90*, pages 207-225, April 1990.

[21] Lance Williams. Pyramidal Parametrics. In *Computer Graphics* (SIGGRAPH '83 Proceedings), volume 17, pages 1-11, July 1983.

[22] Andrew Woo, Andrew Pearce, and Marc Ouellette. It's Really Not a Rendering Bug, You See.... *IEEE Computer Graphics and Applications*, 16(5):21-25, September 1996.

[23] Eric Zeghers, Kadi Bouatouch, Eric Maisel, and Christian Bouville. Faster Image Rendering in Animation Through Motion Compensated Interpolation. In *Graphics, Design and Visualization*, pages 49+. International Federation for Information Processing Transactions, 1993.