# Forward Rasterization

VOICU POPESCU and PAUL ROSEN
Purdue University

We describe forward rasterization, a class of rendering algorithms designed for small polygonal primitives. The primitive is efficiently rasterized by interpolation between its vertices. The interpolation factors are chosen to guarantee that each pixel covered by the primitive receives at least one sample which avoids holes. The location of the samples is recorded with subpixel accuracy using a pair of offsets which are then used to reconstruct/resample the output image. Offset reconstruction has good static and temporal antialiasing properties. We present two forward rasterization algorithms, one that renders quadrilaterals and is suitable for scenes modeled with depth images like in image-based rendering by 3D warping, and one that renders triangles and is suitable for scenes modeled conventionally. When compared to conventional rasterization, forward rasterization is more efficient for small primitives and has better temporal antialiasing properties.

Categories and Subject Descriptors: I.3.3 [**Computer Graphics**]: Picture/Image Generation—*Display algorithms*

General Terms: Theory, Performance

Additional Key Words and Phrases: 3D warping, point-based modeling and rendering, rendering pipeline, rasterization, antialiasing

## 1. INTRODUCTION

In raster graphics, rendering algorithms take as input a scene description and a desired view and produce an image by computing the color of each pixel on a 2D grid. In order to compute the color at a given pixel, the traditional approach is to establish an inverse mapping from the image plane to the scene primitives (from output to input). The ray tracing pipeline potentially computes an inverse mapping from every pixel to every scene primitive. The method is inefficient since only a few pixel/primitive pairs yield a color sample. In spite of acceleration schemes that consider only plausible pixel/primitive pairs, ray tracing is not the method of choice in interactive rendering.

Most interactive graphics applications rely on the feed-forward pipeline. The primitives are first forward mapped to the image plane by vertex projection. Then an inverse mapping from the image to the primitive is computed at rasterization setup. The mapping is used during rasterization to fill in the pixels covered by the primitive. This approach is efficient for primitives with sizeable image projections: the rasterization setup cost is amortized over a large number of interior pixels.

For small primitives, like the ones encountered in complex scenes or in image-based rendering (IBR), the approach is inefficient since the inverse mapping is used for only a few pixels. Researchers in IBR

This research was supported by NSF, DARPA, and the Computer Science Departments of the University of North Carolina at Chapel Hill and of Purdue University.
Authors' address: Computer Sciences Department, Purdue University, 250 N University Street, West Lafayette IN 47907-2066; email: popescu@cs.purdue.edu and rosen@purdue.edu.

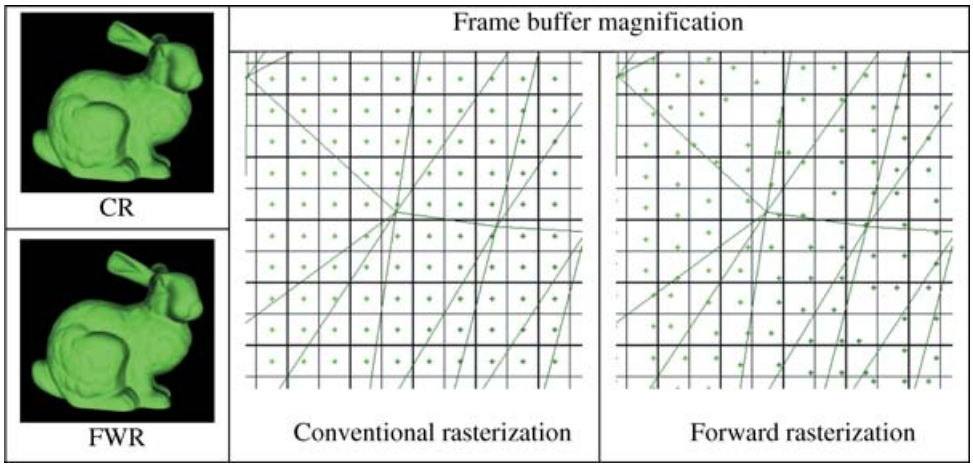


Fig. 1. Images rendered using conventional and forward rasterization. Forward rasterization does not sample at the center of the image pixels.

and point-based rendering have developed an alternative approach where no mapping from pixels to primitives is needed. The approach approximates the output image footprint of the projected primitive with a contiguous set of pixels called a *splat*. Splatting has the potential of being very efficient: the primitives are fragments of output image that are arranged in the desired view by projection. However, computing accurate splats efficiently has proven to be elusive. Splats often have a coarse shape which lowers the quality of the output image. Moreover, no splatting method guarantees that no holes remain between neighboring primitives.

We describe *forward rasterization*, a class of algorithms designed to efficiently and accurately render small polygonal primitives (see Figure 1). A forward rasterization algorithm has the following defining characteristics.

—Samples are generated by interpolation between the vertices of the primitive.
—Sufficient samples are generated to guarantee that each pixel covered by the primitive receives at least one sample.
—The position on the desired image plane of each sample is recorded with subpixel accuracy using a pair of offsets.
—After all primitives are rasterized and z-buffered, the final image is reconstructed/resampled using the offsets stored with each sample.

Forward rasterization is efficient for the following fundamental reasons. First, rasterization is independent of output pixels, avoiding the rasterization setup cost. Second, the inverse mapping needed to compute output pixels is evaluated after visibility, once per output pixel, avoiding unnecessary computation for hidden samples. Third, the inverse mapping implemented using offsets is inexpensive, while sufficiently accurate. In splatting, the input samples are forced to map at the centers of output image pixels; offsets avoid this truncation error, greatly improving the quality of the output image. Moreover, since the samples generated by interpolation move with the primitive in the image, the technique has good temporal antialiasing properties.

Forward rasterization is a class of algorithms because it can be applied to any type of polygonal primitive and because there are many ways of interpolating the vertices of a primitive to generate samples. We present in detail two algorithms, one that renders quadrilaterals and is suitable for scenes

modeled with depth images like in IBR or with tessellated higher-order primitives, and one that renders triangles and is suitable for scenes modeled conventionally with triangle meshes.

The article is organized as follows. The next section reviews prior work. Section 3 gives an overview of the forward rasterization approach. Section 4 describes forward rasterization of quads and its use in the context of IBR by 3D warping (IBRW). Section 5 describes forward rasterization of triangles. Section 6 discusses offset reconstruction. Section 7 discusses our results and sketches directions for future work.

## 2. PRIOR WORK

Levoy and Whitted [1985] were the first to question the efficiency of the classic rendering pipeline for primitives with small screen-space projection as encountered in scenes with high complexity. In what is one of the most frequently cited technical reports in graphics, they point out that it is desirable to separate modeling from rendering and advocate converting primitives to points prior to rendering. They identify maintaining the opacity of point-rendered surfaces as an important and challenging problem. They propose a solution based on locally approximating the surface with its tangent plane at each point. The method does not guarantee surface opacity since the tangent planes sometimes underestimate the actual point density needed in the output image.

The issue of efficiently rendering small primitives resurfaced 10 years later in IBRW. McMillan and Bishop [1995] propose modeling and rendering a 3D scene with images enhanced with per pixel depth. The depth-and-color samples are 3D warped (reprojected) efficiently to novel views. In order to maintain the opacity of front surfaces, two approaches are investigated.

### 2.1 IBR Mesh Method

A possible reconstruction technique for IBRW is the mesh method which connects four neighboring samples of a depth image with two triangles. The depth image becomes a 3D mesh with the nodes defined by the depth samples. The mesh is disconnected when neighboring samples do not belong to the same surface. The mesh is rendered in hardware. The mesh method maintains the opacity of front surfaces since the triangles stretch to adapt to the novel view. However, the method is inefficient because it renders every depth-and-color sample by conventionally rasterizing two triangles that cover at most a few pixels.

### 2.2 IBR Splatting

In splatting [McMillan 1997; Mark et al. 1997; Mark 1999] the shape and size of the projected depth-and-color sample is approximated by splats, a term borrowed from volume rendering [Westover 1990]. Splatting was used in numerous IBRW applications [Rafferty et al. 1998; Popescu et al. 1998; Shade et al. 1998; Aliaga and Lastra 2003] since it is relatively inexpensive. For this reason, rendering with splats was also considered for emerging graphics platforms which have limited transformation, fill-rate, and display resolutions [Duguet and Drettakis 2004].

In splatting, the quality of the reconstruction relies heavily on the precision with which the splats approximate the projected samples. If the splats are too small, neighboring samples of the same surface will leave holes. Consequently, the splats are typically approximated to excess. Splats that are too large incorrectly erase visible samples.

Attempts have been made to improve the quality of splatting reconstruction by filtering the samples as they are warped [Shade et al. 1998]. However, blending before the visible samples are selected and the invisible ones erased, produces artifacts. Another improvement on splatting with a hierarchical scene representation for IBRW is proposed by Chang et al. [1999]. The depth images are stored at several levels of detail. At rendering time, the appropriate level of detail is used such that the projected

samples cover about one pixel of the desired image. Since splats are never bigger than one pixel, the reference images must always have at least the resolution of the desired view. The method suffers from lengthy preprocessing times, large memory requirements, and reduced frame rates.

McAllister et al. [1999] used the programmable-primitives feature of PixelFlow [Molnar et al. 1992; Eyles et al. 1997] in order to reconstruct the warped image. Each sample is rasterized as a disk with a quadratic falloff in depth. This is similar to the technique of rendering Voronoi regions by rasterizing cones [Haeberli 1990] which was used before for reconstruction in image-based rendering [Larson 1998]. The method relied on PixelFlow's unique ability of efficiently rendering quadratics.

Researchers developed splatting techniques that produce high-quality images but that was only achieved by considerably increasing the complexity of the splats. The emphasis shifted from achieving efficient rendering to developing a versatile modeling paradigm.

### 2.3 Point-Based Graphics

The *QSplat* system [Rusinkiewicz and Levoy 2000] stores the scene as a hierarchy of bounding spheres which provides visibility-culling and level-of-detail adaptation during rendering. As with the previous splatting methods, determining the shape and size of the splat is difficult. The system guarantees the opacity of the front surface if circular or square splats are used. However, coercing the splats to be symmetrical degrades the reconstruction. The authors report aliasing at the silhouette edges.

The *surfel* method [Pfister et al. 2000] is another variant of the splatting technique. Surfels are data structures that contain sufficient information for reconstructing a fragment of a surface. Rendering with surfels is done in two stages, visibility and then reconstruction. Visibility eliminates the hidden surfels and is done by splatting model-space disks onto the desired-image plane and then scan-converting their bounding parallelogram into the z-buffer. The cost of rendering a splat is that of rendering two triangles. There is no guarantee that no holes remain. The output image is reconstructed from surfels that survive the visibility stage.

*Elliptical Weighted Average* (*EWA*) surface splatting [Zwicker et al. 2002; Ren et al. 2002] builds on the surfels work to improve the quality of the reconstruction by adapting Heckbert's work [1989] on filtering irregular samples to point-based rendering. During a preprocessing stage, the points with color are used to create a continuous texture which is used at run time to color the visible surface elements. The method enables high-quality anisotropic filtering within the surface and antialiases edges using coverage masks, bringing to point-based rendering what was previously possible only for polygonal rendering. However, porting these techniques to points comes at a higher cost since one has to overcome the lack of connectivity and make do with approximate connectivity inferred from the distances to the neighboring samples. These techniques are readily available to IBRW when the reconstruction is based on the triangle mesh approach which is the case of forward rasterization.

In recent work, Whitted and Kajiya [2005] investigate a programmable pipeline where geometry and shading are represented procedurally. The pipeline can be seen as a generalization of point-based rendering. Programmable geometric primitives have been discussed before in the context of offline rendering [Whitted and Weimer 1982; Cook et al. 1987], and interactive rendering with special hardware [Olano 1998]. The novelty of the fully procedural pipeline is that the rasterizer is replaced with a sample generator which generates samples until hole-free coverage is obtained. Samples are generated by Kajiya invoking the geometric primitive procedure with different parameters. Forward rasterization can be seen as treating quads and triangles procedurally. Although Whitted and Kajiya's and [2005] work is just a preliminary study with many aspects of the pipeline remaining to be refined, the work points out the potential advantage of forward rasterization, and it identifies which challenges have to be overcome to concretize this advantage.

### 2.4 Triangle Rasterization

Since this article proposes a forward rasterization for triangles, we also briefly review rasterization and antialiasing approaches for polygonal rendering. Several techniques were proposed for filling in projected triangles. An early technique referred to as *Digital Differential Analyzer* (DDA) walks on the edges of the triangle and splits the triangle in horizontal spans of pixels. The DDA approach is used by the *RealityEngine* [Akeley 1993].

The Pixel-Planes architecture [Fuchs et al. 1985] has the ability to quickly evaluate linear expressions for many pixels in parallel and introduces the edge equations approach to rasterization. For a given projected triangle, edge equations and rasterization parameter planes are 2D linear expressions in the pixel coordinates $(u, v)$. Given a pixel, deciding whether it is interior to the triangle and computing its rasterization parameter values is done in a unified manner by evaluating 2D linear expressions. What remains is the question of which pixels to consider for a given triangle.

One approach is to use a rectangular screen-aligned bounding box. The advantage of the method is its simplicity. Other approaches have been proposed. Pineda [1988] describes a traversal approach driven by a center line. McCormack and McNamara [2000] describe traversal orders that generate all samples belonging to one rectangular tile of the screen before proceeding to the next tile which is advantageous in the case of partitioned framebuffers. McCool et al. [2001] propose traversing the primitive's bounding box on hierarchical Hilbert curves for improved spatial coherence. These methods reduce the number of exterior pixels considered but come at the price of increased per triangle and per scan-line cost.

*InfiniteReality*, the second SGI graphics supercomputer [Montrym et al. 1997], adopts the edge equations approach to reduce the setup cost of DDA. *PixelFlow* [Molnar et al. 1992; Eyles et al. 1997], the sort-last successor of *Pixel-Planes*, continues to rasterize using linear expressions. In the mid nineties, the almost vertical progress of PC graphics accelerators begins. In a few years, add-in cards catch up from behind and render the graphics supercomputers obsolete [NVIDIA; ATI]. The specifics of the architectures and of the algorithms they implement are not published but a study of patent applications and whitepapers indicate that graphics cards use variants of the edge equation rasterization.

The graphics architecture literature describes several other rasterization approaches. Good overviews of earlier variants of rasterization are given in the survey paper [Garachorloo et al. 1989], in Kaufman's book [1993], and in Ellsworth's Ph.D. thesis [1996].

Greene [1996] describes a hierarchical rasterization (tiling) algorithm that integrates occlusion culling, rasterization, and antialiasing. The subpixel screen coverage of a triangle is determined recursively, using precomputed triage coverage masks for each of its 3 edges. Such a $k \times k$ mask stores which $k \times k$ subregions are inside, outside, or crossing the edge. The rasterization is essentially looked up rather than computed. The method outperforms conventional software rasterization for static, high-depth complexity scenes. Dynamic scenes pose a problem since polygons cannot be presorted in front-to-back order, a requirement for efficient occlusion culling. The approach does not offer a viable stand-alone rasterization solution. The coverage masks avoid the computational cost of determining coverage, but replace it with an increased bandwidth requirement to look up the masks. Moreover, the coverage computation is just a small fraction of the rasterization cost which is dominated by computing the rasterization parameters at each pixel.

Homogeneous coordinates rasterization [Olano and Greer 1997] has the advantage of incorporating clipping into rasterization which avoids the need of actually splitting the triangle. The triangle is enhanced with *clip edges*, one for each clip plane, including the planes defining the view frustum and any arbitrary clip planes specified by the application. The authors state that the algorithm requires a higher number of operations because the hither clip edge requires interpolating all parameters with

perspective correction. Whether rasterization proceeds with homogeneous or projected coordinates is an issue orthogonal to the issue of forward versus inverse rasterization.

Barycentric rasterization [Brown 1999a, 1999b] is an algorithm that limits the setup computations to establishing the linear expressions that give the variation of the barycentric coordinates in the image plane. Then, for every pixel, the barycentric coordinates are used to decide whether the pixel is interior to the triangle, and, if it is, to blend the rasterization parameter values at the vertices. In his study of mapping the graphics pipeline to a programmable stream processor architecture, Owens [2003] proposes using barycentric rasterization to reduce the burden of carrying a large number of interpolants through the rasterization process. McCool et al. [2001] independently develop barycentric interpolation (without explicitly calling it so) for the same purpose of better handling a large number of interpolants.

There are numerous variants of inverse rasterization. We have chosen to compare forward rasterization to the edge equation approach because it is widely adopted by hardware implementations and to barycentric rasterization because of its low setup cost.

### 2.5 Antialiasing

A fundamental challenge in raster graphics is dealing with the nonzero pixel size. The limited sampling rate limits the maximum frequency that can be displayed and ignoring this limit produces visually disturbing *aliasing* artifacts. Antialiasing techniques are expensive since they require rendering with subpixel accuracy. Most techniques compute several color samples per pixel and then blend them to obtain the final pixel color. Stochastic sampling [Cook 1986] produces high quality images but is too slow to be used in interactive graphics applications. See Glassner's ray tracing book [1978] for a good list of additional references.

In interactive graphics, one approach is regular supersampling, where the pixel is subdivided in $\sqrt{n} \times \sqrt{n}$ subpixels. A better approach is to avoid collinear sampling locations. One possibility is random sampling. Jittered supersampling perturbs the sampling locations away from the centers of the $\sqrt{n} \times \sqrt{n}$ subpixels. The n-rooks sampling pattern starts with an $n \times n$ subdivision of the pixel, selects one sample randomly in each of the $n$ cells of the main diagonal, and then shuffles the horizontal coordinate $u$ of the samples. All these techniques, generically called jittered supersampling, employ sampling locations that are defined with respect to the pixel grid and are, therefore, equivalent for the purpose of comparing forward to conventional rasterization. Molnar et al. [1991] investigates the number of color samples per output pixel that is required to obtain high quality images. He concludes that good results are obtained with as little as 5 samples. The *SAGE* graphics architecture [Deering and Naegle 2002] offers the option of up to 16 color samples per output pixel and complete flexibility regarding the reconstruction filter.

Carpenter [1984] introduces the a-buffer, a technique that achieves high-quality supersampling by storing at each pixel a list of coverage masks. The algorithm is expensive, but it introduces the important idea that coverage computation is orthogonal to the number of color samples per pixel. Several antialiasing algorithms were developed based on in this idea [Abram and Westover 1985; Schilling 1991], and the idea is used in today's (extended) graphics APIs [OpenGL, DirectX] which differentiate between multisampling and supersampling to imply higher resolution coverage or color sampling, respectively.

Antialiasing has a direct impact on quality so the antialiasing capability of a graphics card is a heavily marketed feature. Moreover, antialiasing always comes at a cost so application developers need to be provided with an understanding, albeit minimal, of the underlying algorithm in order to decide when to use it and with what parameters. For these reasons several whitepapers are available [NVIDIA 2004, 2005a, 2005b] describing the antialiasing capabilities of graphics cards. A review of the various

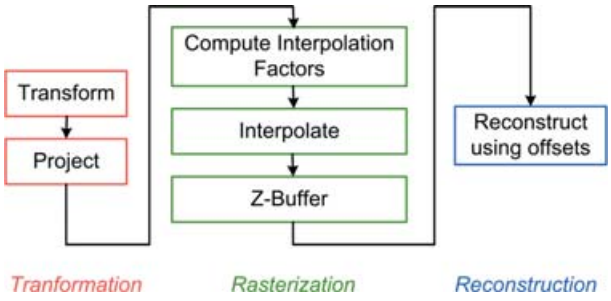


Fig. 2. Forward rasterization pipeline.

antialiasing modes is beyond the scope of this article. The modes differ in the number of samples per pixel, in the resolution at which pixel coverage is computed, and in the kernel footprint and shape used to filter the final image, but they are, in fact, variants of the general jittered supersampling approach.

## 3. FORWARD RASTERIZATION OVERVIEW

We have seen that the splatting approach has not yet produced a method for rendering from samples that is both accurate and efficient. Computing the precise size and shape of a splat is nontrivial. Although it might appear that splatting does not need connectivity, in fact, knowing the neighbors of a sample is indispensable for the footprint approximation. Splats that tile perfectly without overlap and without leaving holes are equivalent to rasterizing the triangle mesh that interconnects the samples. Connecting the samples with triangles is, at the conceptual level, the simplest approximation possible that maintains $C^0$ continuity. Assuming that the surfaces are planar between samples is the approximation of lowest order possible so it is doubtful that a splatting method will ever be simultaneously as accurate as and more efficient than the polygonal mesh method.

Therefore our goal was to improve the mesh method by devising polygon rendering algorithms that take into account the small image footprint of the polygons. Forward rasterization provides infrastructure usable in many contexts, including IBR, point-based graphics, rendering of higher-order surfaces, and rendering of complex polygonal scenes. The work presented here builds upon earlier work [Popescu et al. 2000] where we described the forward rasterization of quads and the offset reconstruction in context of the *WarpEngine*, a hardware architecture for IBRW. We briefly review those concepts here for completeness. The contributions of this article, in the order in which they appear, are:

—interpolation factor selection for forward rasterization of quadrilaterals that guarantees a hole-free reconstruction (with proof);
—forward rasterization of triangles, including interpolation factor selection for guaranteed hole-free reconstruction (with proof);
—cost analysis of forward rasterization of quads and of triangles (number of operations at setup, number of times inner loop is executed, overdraw factor);
—analysis of z-buffering precision in forward mapping rendering;
—temporal antialiasing analysis of forward rasterization.

Forward rasterization proceeds as shown in Figure 2. The primitive is transformed and projected onto the output image plane as usual. Rasterization begins by computing small but sufficient interpolation factors that guarantee that no holes remain. Then samples are generated by interpolation between the vertices of the projected primitive. The actual image plane location of a sample is recorded using a pair of offsets. After visibility, the offsets are used to reconstruct the output image.

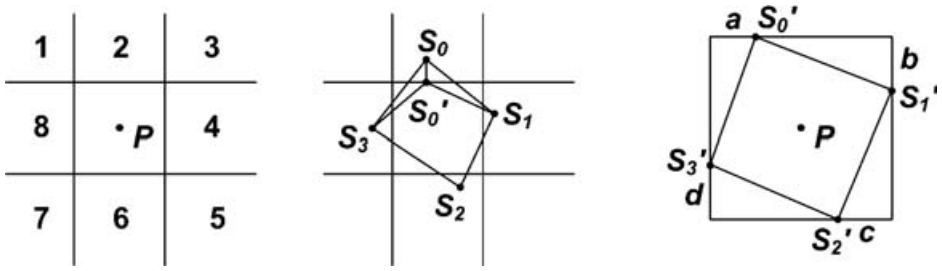


Fig. 3. Proof of rasterization mesh property.

Forward rasterization generates samples that can land anywhere within the boundaries of a pixel. Since no inverse mapping is computed, forward rasterization cannot guarantee that each covered pixel receives exactly one sample. In order to guarantee that each pixel receives at least one sample, for some pixels, forward rasterization algorithms generate more than one sample (overdraw). The success of the approach depends on the ability to compute safe interpolation factors that limit the amount of overdraw. The number and cost of redundant samples depend on the particular forward rasterization algorithm. Estimating and controlling the amount of overdraw is an essential concern in this work.

## 4. FORWARD RASTERIZATION OF QUADS

In some cases, the quadrilateral is the natural modeling primitive. In IBRW, the reference image is a 2D grid of depth and color samples which implicitly defines a mesh of 1 pixel squares between neighboring samples. When the samples are warped to a novel view, the squares become general quads with an image coverage that typically does not exceed a few pixels. Higher-order tensor product surfaces can naturally be tessellated in a mesh of quads using their inherent 2D surface parameterization. Rendering small quads as two triangles is inefficient, and we have developed an algorithm that directly handles quads.

### 4.1 Interpolation

An image-plane quad $V_0V_1V_2V_3$ (see Figure 4) is forward rasterized by bilinear interpolation with interpolation factors $f_0$ x $f_1$ given by Equation (1).

$$f_0 = \left\lceil \sqrt{2}\max(V_0V_1, V_2V_3) \right\rceil$$
$$f_1 = \left\lceil \sqrt{2}\max(V_0V_3, V_1V_2) \right\rceil \quad (1)$$

We now prove that rasterizing an image-plane mesh of quads using the interpolation factors given guarantees that every pixel covered by the mesh will receive at least one sample. The proof has two parts. First, we show that if the quads in a mesh have edges shorter than $1/\sqrt{2}$ pixels, every pixel covered by the mesh contains a mesh node (i.e. a quad vertex). We call such a mesh a *rasterization mesh*. Then we show that Equation (1) generates a rasterization mesh.

CLAIM 1. *Every pixel covered by a rasterization mesh contains at least one mesh node.*

PROOF. Let's assume that there is a pixel $P$ covered by a rasterization mesh that does not contain a quad vertex. Let $S_0S_1S_2S_3$ be the quad that contains the center of $P$. Such a quad exists since $P$, including its center, is covered by the mesh. Vertices $S_0$, $S_1$,$S_2$, and $S_3$ are outside pixel $P$ since $P$ does not contain any vertex. Their possible locations are shown in Figure 3, left. No vertex can belong to

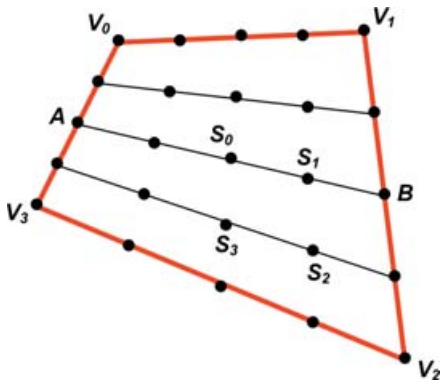


Fig. 4. Bilinear interpolation of quad.

regions 1, 3, 5, or 7. This can be shown by analyzing what happens if, for example, $S_0$ belongs to region 1. $S_1$ and $S_3$ have to belong to regions 2 and 8. $S_2$ has to belong to one of the regions 7, 6, 5, 4, or 3 in order for $S_0S_1S_2S_3$ to contain the center of $P$. This makes at least one of the edges $S_1S_2$ or $S_3S_2$ longer than $1/\sqrt{2}$. It can be argued similarly that none of the regions 2, 4, 6, or 8 can contain two or more of the vertices $S_0$, $S_1$,$S_2$, and $S_3$.

We have established that regions 2, 4, 6, and 8 each contain one of the 4 vertices (Figure 3 middle). Let $S'_0$ be the projection of $S_0$ onto the edge of $P$ shared with region 2. In triangles $S'_0S_0S_3$ and $S'_0S_0S_1$, the angle at vertex $S'_0$ is largest, thus the opposite edge is the longest. This implies that $S'_0S_3 < S_0S_3 < 1/\sqrt{2}$ and $S'_0S_1 < S_0S_1 < 1/\sqrt{2}$. Points $S'_1$, $S'_2$, and $S'_3$ are constructed similarly, yielding the quad $S'_0S'_1S'_2S'_3$ with edges shorter than $1/\sqrt{2}$. Using Pythagoras' theorem in each of the 4 triangles with hypotenuses $S'_0S'_1$, $S'_1S'_2$, $S'_2S'_3$,and $S'_3S'_0$ (Figure 3 right), we reach a contradiction (Equation (2)).

$$
\begin{aligned}
&a^2 + (1-d)^2 < \frac{1}{2}\\
&b^2 + (1-a)^2 < \frac{1}{2}\\
&c^2 + (1-b)^2 < \frac{1}{2}\\
&d^2 + (1-c)^2 < \frac{1}{2}\\
&2a^2 + 2b^2 + 2c^2 + 2d^2 - 2a - 2b - 2c - 2d + 4 < 2\\
&\left(a - \frac{1}{2}\right)^2 + \left(b - \frac{1}{2}\right)^2 + \left(c - \frac{1}{2}\right)^2 + \left(d - \frac{1}{2}\right)^2 < 0
\end{aligned}
\tag{2}
$$

This implies that our assumption that that there is a pixel without any mesh node within its boundaries is false which terminates the proof of Claim 1.

Consequently, to prevent holes, the interpolation factors have to be chosen such that subquads generated by bilinear interpolation have edges shorter than $1/\sqrt{2}$. We next show how to conveniently control the length of the edges of the subquads using the bilinear interpolation factors.

CLAIM 2. *Given a quad $V_0V_1V_2V_3$, let $S_0S_1S_2S_3$ be a subquad formed by four neighboring samples generated by bilinearly interpolating $V_0V_1V_2V_3$ with interpolation factors $f_0$ x $f_1$ (see Figure 4). The*

*following inequality holds*:

$$\max(S_0S_1, S_2S_3) \le \frac{1}{f_0}\max(V_0V_1, V_2V_3)$$
$$\max(S_1S_2, S_3S_0) \le \frac{1}{f_1}\max(V_1V_2, V_3V_0). \tag{3}$$

PROOF. All the segments on a bilinear interpolation row or column have the same length. The same samples are generated no matter whether the interpolation proceeds in row-major order or in column-major order. Consequently, all we have to do is to show that $AB \le \max(V_0V_1, V_2V_3)$, where $A$ and $B$ are corresponding points generated on $V_0V_3$ and $V_1V_2$ (see Figure 4). We derive the proof of the inequality using only the $x$ components of $AB$, $V_0V_1$, and $V_2V_3$; the $y$ expressions are similar because of symmetry.

$$\begin{aligned} V_0V_{1_x}^2 &= \left(x_{V_0} - x_{V_1}\right)^2, V_3V_{2_x}^2 = \left(x_{V_3} - x_{V_2}\right)^2, AB_x^2 = (x_A - x_B)^2 \\ x_A &= x_{V_0} + \left(x_{V_3} - x_{V_0}\right)k \\ x_B &= x_{V_1} + \left(x_{V_2} - x_{V_1}\right)k \\ 0 &\le k \le 1. \end{aligned}$$

Let

$$\begin{aligned} x_{01} &= x_{V_0} - x_{V_1} \\ x_{32} &= x_{V_3} - x_{V_2} \\ V_0V_{1x}^2 &= x_{01}^2 \\ V_3V_{2x}^2 &= x_{32}^2, \end{aligned}$$

then

$$\begin{aligned} AB_x^2 &= (x_{01} + (x_{32} - x_{01})k)^2 = \\ &= x_{01}^2 + \left(x_{32}^2 - x_{01}^2\right)k - \left(x_{32}^2 - x_{01}^2\right)k + (x_{32} - x_{01})^2k^2 + 2x_{01}(x_{32} - x_{01})k = \\ &= V_0V_{1x}^2 + \left(V_3V_{2x}^2 - V_0V_{1x}^2\right)k - k(x_{32} - x_{01})(x_{32} + x_{01} - k(x_{32} - x_{01}) - 2x_{01}) = \\ &= V_0V_{1x}^2 + \left(V_3V_{2x}^2 - V_0V_{1x}^2\right)k - k(1-k)(x_{32} - x_{01})^2 = \\ &= V_0V_{1x}^2 + \left(V_3V_{2x}^2 - V_0V_{1x}^2\right)k - \Delta_x, \Delta_x \ge 0. \end{aligned}$$

Consequently,
$AB^2 \le V_0V_1^2 + (V_3V_2^2 - V_0V_1^2)k \le \max(V_0V_1^2, V_3V_2^2)$,which terminates the proof. □

Figure 5 shows the samples generated by forward rasterization with red diagonal crosses. The samples are independent of the pixel grid. The shaded region shows the quad footprint. All pixels have at least one sample.

### 4.2 Bow Ties

The four depth image samples defining the quadrilateral primitive are not necessarily coplanar. The projection of the quad could be concave. Forward rasterization, which bilinearly interpolates in screen space, generates a primitive footprint equivalent to that obtained by model-space bilinear interpolation. In Figure 6, the left wire frames are obtained by forward rasterizing the projection of the quad with $9 \times 9$ interpolation factors. (The coarse interpolation factors were chosen to show the samples generated at the intersection of the sampling lines.) The right-hand images were obtained by projecting the $9 \times 9$

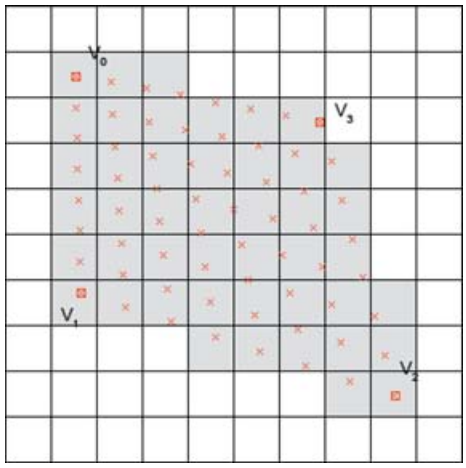


Fig. 5. Forward rasterization of quad. The red crosses show the samples generated and the shaded region shows the quad footprint.

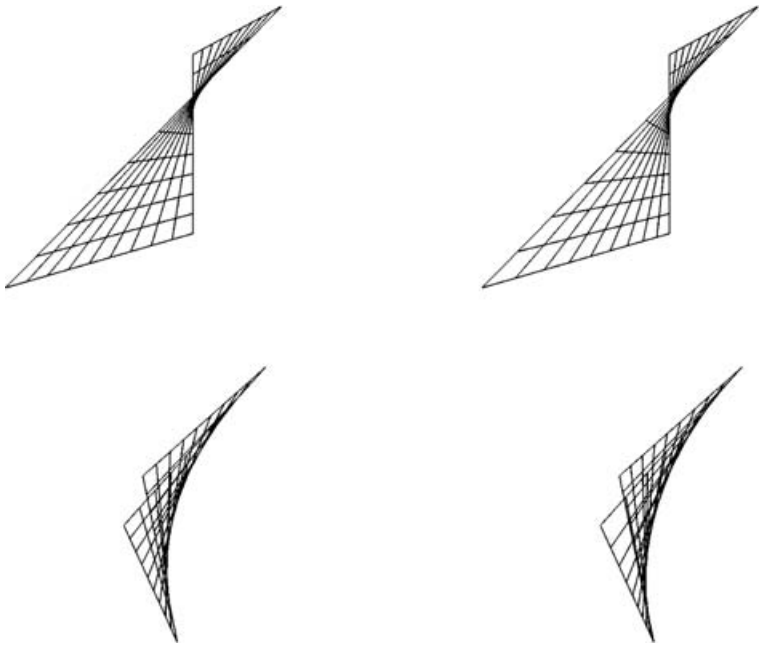

Fig. 6. Screen-space (left) vs. model-space bilinear interpolation of quad with concave projection.

interpolated model space quad. The only difference between the images is the variable image-space sampling rate due to perspective foreshortening in the case of model-space interpolation.

The depth image all by itself approximates the surface between the samples with a bilinear patch. If additional information is available indicating that the surface is in fact better modeled with two triangles obtained by using one of the diagonals, forward rasterization will not produce the correct

footprint. In such cases, the projected vertices could be rearranged to rasterize a conservative convex hull, or the triangles could be rasterized independently.

In practice, bowties are not a problem. Neighboring bilinear patches, even if concave, have $C^0$ continuity. In IBRW, sizeable concave quads occur only between foreground and background objects where the mesh is disconnected anyway to avoid skins. Silhouette concave quads are close to collinear and do not reduce the quality of the silhouette approximation.

### 4.3 Cost Analysis

We compare the cost of forward rasterizing a quad to the cost of conventionally rasterizing two triangles. We first compare it to the widely-used edge equation approach introduced by the *Pixel-Planes* architecture [Fuchs et al. 1985]. We examine the cost of rasterization setup and of the sample generation loop separately. We assume that conventional rasterization executes the inner loop a number of times equal to the area of the screen-aligned bounding rectangle of the triangle. Although techniques for reducing the number of exterior pixels that are tested have been proposed [Pineda 1988], they come with additional setup and scan line costs which make the analysis scene dependent and might offset the benefits in the case of small primitives.

Finally, we compare forward rasterization to barycentric rasterization, a variant of conventional rasterization that uses barycentric coordinates as interpolants to compute the rasterization parameter values at each pixel.

4.3.1 *Rasterization Setup.* Conventional rasterization setup has three parts: bounding box computation, edge equation computation, and computation of the linear expressions that give the rasterization parameter values at a given image plane location. The expressions needed for the first two parts can be reused in the third part so we only count the cost of the third part. In IBRW and whenever the primitives are small in screen space, it is sufficient to screen-space interpolate the colors of the vertices. Therefore, the cost analysis assumes 4 rasterization parameters: $R$, $G$, $B$, and $1/z$ which are linearly interpolated in screen space.

The linear expression $A^i u+B^i v+C^i$ that gives the image plane variation of a rasterization parameter $p^i$ is computed according to Equation (4), where $(u_j, v_j), j = 0$, 1, and 2 are the image plane coordinates of the three vertices.

$$\begin{matrix} A^i \\ B^i \\ C^i \end{matrix} = \begin{bmatrix} u_0 & v_0 & 1 \\ u_1 & v_1 & 1 \\ u_2 & v_2 & 1 \end{bmatrix}^{-1} \begin{bmatrix} p_0^i \\ p_1^i \\ p_2^i \end{bmatrix} \tag{4}$$

The barycentric coefficient matrix is the same for all parameters. Charging $3^3$ multiplications for the matrix inversion and $3^2$ multiplications for the linear expressions of each parameter, the cost of the conventional rasterization setup for two triangles is approximately $2(3^3 + 4(3^2)) = 126$ multiplications.

Forward rasterization setup has two parts, interpolation factor computation and computation of the interpolation increments for the rasterization parameters. The interpolation factors $f_0$ and $f_1$ require computing the square of the length of the four edges, for a total of 8 multiplications. The square root is handled with a look-up table since an integer result is desired and since the interpolation factors are small. The look-up table also provides the quantities $1/f_0$, $1/f_1$ and $1/(f_0 f_1)$.

In forward rasterization, the scan-line interpolation increments are not constant from line to line, but they vary linearly. Consequently, for each rasterization parameter, three increments need to be computed: within line ($incr_c$), from line to line ($incr_l$), and $incr_c$ line to line variation ($incr_{cl}$). This is done with 3 multiplications and the total cost of computing the increments for rasterization parameter interpolation is 18 multiplications, since in addition to $R$, $G$, $B$, $1/z$, forward rasterization also

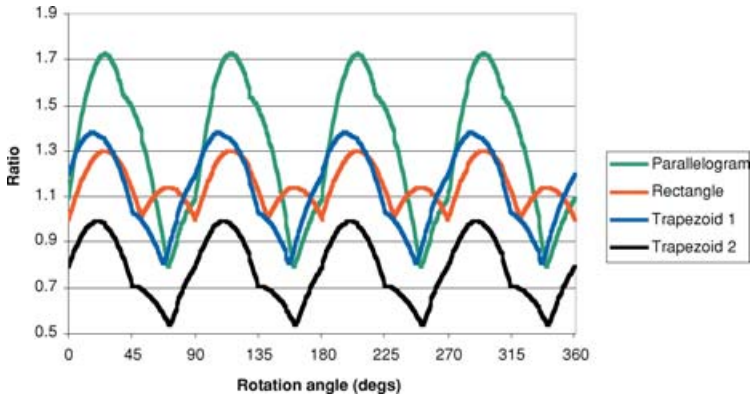


Fig. 7. Ratio between the number of times the inner loop is executed in conventional rasterization and forward rasterization.

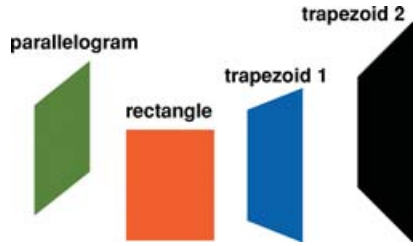


Fig. 8. Test quads used for the graph shown in Figure 7.

interpolates image coordinates $u$ and $v$. The total cost of forward rasterization setup is $18 + 8 = 26$ multiplications, approximately 5 times less than that of conventional rasterization.

We have conducted a detailed analysis that considers all the operations, including additions, at each stage; we do not reproduce it here, but it is available as a technical report [Popescu 2001]. Its results confirm the results of the approximate analysis previously given. Note that conventional rasterization has a larger per-parameter cost and interpolating more parameters increases the cost advantage of forward rasterization.

4.3.2 *Sample Generation Loop.* Both approaches rasterize using two nested for loops. The rasterization parameter values are computed incrementally at an amortized cost of one addition. The same number of additions is performed since the edge expressions needed in conventional rasterization are replaced by $u$ and $v$ interpolation in forward rasterization. For this, we compare the costs of the two approaches by comparing the number of times the inner loop is executed. Conventional rasterization executes the inner loop $A$ times, where $A$ is the sum of the areas of the bounding boxes of the projections of the two triangles. Forward rasterization executes the inner loop $f_0 f_1$ times, where $f_0$ and $f_1$ are the two interpolation factors.

We compared the two quantities for various 2D quads and for actual scenes. For each quad, we tested all possible image orientations, with one-degree increments. The results are given in Figure 7. The test quads are shown in Figure 8. For the rectangle, the classic rasterization algorithm always executes the inner loop more times than the forward rasterization algorithm (ratio greater than one). This is due to the overlap between the bounding boxes of the two triangles, and a formal proof of the result is relatively simple. The minima in the graph correspond to axis-aligned rectangle sides or axis-aligned diagonal (along which the rectangle is split) sides. In such a case, the ratio is exactly one,

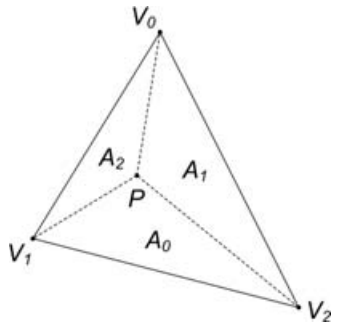


Fig. 9. Barycentric rasterization.

and the total number of times the inner loop is executed is $2ab$, where $a$ and $b$ are the sides of the rectangle.

For the parallelogram, the minima are reached when the diagonal is axis aligned. The minima are below 1 so, for some orientations of the parallelogram, the forward rasterization algorithm executes the inner loop more times than the classic algorithm.

The two algorithms behave similarly in the case of *trapezoid* 1. The worst case for the forward rasterization algorithm corresponds to quads that have opposite sides of very different length as is the case for *trapezoid* 2. For such a quad, numerous redundant iterations occur when interpolating close to the shorter side. Such quads are unlikely since in IBRW or in the tessellation of higher order primitives, the model space aspect ratio of the quads is close to one. This means that the only possible source of imbalance between the lengths of opposite sides is perspective foreshortening. Since the quads are small, perspective foreshortening has negligible effect.

Another important aspect of the comparison between the two methods is the number of samples produced. The classic method generates the minimum number of samples which corresponds to the area of the quad (or polygon, in general). The forward rasterization method can guarantee surface continuity only at the price of some redundant samples. Redundant samples are costly if shading is costly. In the case of IBRW, shading is inexpensive since all that is required is simple color interpolation. Moreover, the average interpolation factor is small, so substantially reducing setup remains a big advantage. In Section 5, we describe techniques for limiting the number of redundant samples created when forward rasterizing triangles based on early discarding of samples that go to the same pixel; such techniques can also be applied to quads.

4.3.3 *Comparison to Barycentric Rasterization.* Brown [1999] proposes using the 2D barycentric coordinates of image plane points inside a triangle as interpolants for rasterization. The method has a reduced rasterization setup cost, but has a larger per pixel cost. The essence of barycentric rasterization is to limit rasterization setup to computing the linear expressions that give the barycentric coordinates of a point in the image plane. These expressions are then used inside the rasterization loop to compute the barycentric coordinates at a pixel which are used in turn to decide whether the pixel is inside the triangle and to compute the rasterization parameter values at the pixel. Since the technical report describing barycentric interpolation [Brown 1999a] is difficult to obtain, we give a brief overview of the technique.

A point P inside an image plane triangle $V_0V_1V_2$ splits the triangle into 3 subtriangles (Figure 9). The barycentric coordinates $b_i$ of $P$ can be expressed as the ratios between the areas $A_i$ of the subtriangles, and the area $A$ of the entire triangle. Area $A_0$ is half the cross product between 2D vectors $V_1P$ and $V_2P$. If the pixel coordinates of $V_i$ are $(u_i, v_i)$ and those of $P$ are $(u, v)$, the barycentric coordinates $b_i$

are given by Equation (5), where the incremented subindices $i+1$ and $i+2$ are computed modulo 3.

$$\begin{aligned}
b_i &= A_i/A \\
2A_i &= \begin{vmatrix} u_{i+1}-u & v_{i+1}-v \\ u_{i+2}-u & v_{i+2}-v \end{vmatrix} = (u_{i+1}-u)(v_{i+2}-v) - (u_{i+2}-u)(v_{i+1}-v) = \\
&= (v_{i+1}-v_{i+2})\,u + (u_{i+2}-u_{i+1})\,v + (u_{i+1}v_{i+2} - u_{i+2}v_{i+1}) \\
2A &= \begin{vmatrix} u_1-u_0 & v_1-v_0 \\ u_2-u_0 & v_2-v_0 \end{vmatrix} = (u_1-u_0)(v_2-v_0) - (u_2-u_0)(v_1-v_0) \\
b_i &= \frac{(v_{i+1}-v_{i+2})}{A}u + \frac{(u_{i+2}-u_{i+1})}{A}v + \frac{(u_{i+1}v_{i+2}-u_{i+2}v_{i+1})}{A} \\
b_i &= K_i^0 u + K_i^1 v + K_i^2
\end{aligned} \tag{5}$$

Each of the 3 barycentric coordinates $b_i$ is given by a linear expression with coefficients $K_i^0$, $K_i^1$, and $K_i^2$. Ignoring the additions, computing each linear expression takes 2 multiplications and 3 divisions, or 5 multiplications since the denominator is the same. The area of the entire triangle is computed with two multiplications. The total rasterization setup cost for the triangle is $5 * 3 + 2 = 17$ multiplications, or 34 multiplications for 2 triangles. This setup cost is comparable to that of forward rasterization (26 multiplications). However, barycentric rasterization has a much higher per-sample cost.

In the sample generation loop, for each pixel considered for the given triangle, the first step is to compute the barycentric coordinates using the linear expressions established at setup. This is done with an amortized cost of 3 additions. If any of the barycentric coordinates is negative, the pixel is outside the triangle. When a pixel is inside a triangle, the rasterization parameters have to be computed as an average of the values at the vertices, weighted with the barycentric coordinates. (See Equation (6), where $b_i$(u, v) and r(u, v) are the barycentric coordinates, and the rasterization parameter r at pixel (u, v)). This implies 3 multiplications per rasterization parameter, or 12 multiplications when $r$, $g$, $b$, and $z$ are desired.

$$r(u,v) = r(u_0,v_0)b_0(u,v) + r(u_1,v_1)b_1(u,v) + r(u_2,v_2)b_2(u,v) \tag{6}$$

In conclusion, in the case of IBRW, barycentric rasterization has a setup cost comparable to that of forward rasterization but deviates significantly from the optimal cost of 1 addition per pixel, per rasterization parameter. This implies that barycentric interpolation is comparable to forward rasterization only for quads that are so small as to not cover any pixels. A quad that covers 5 pixels requires 60 additional multiplications in the case of barycentric rasterization.

## 5. FORWARD RASTERIZATION OF TRIANGLES

### 5.1 Interpolation

Simply bilinearly interpolating the triangle as a degenerate quad produces too many redundant samples as shown in Figure 10, left. Another straightforward interpolation approach is to split each edge of the triangle into $k$ equal segments, and then to connect the points by lines parallel to the edges (Figure 10, right). The disadvantage is that the approach generates the same number of samples on each edge, regardless of its length, which produces numerous redundant samples for triangles that are far from equilateral.

We use a barycentric sampling of the triangle, along lines parallel with two of the edges, $V_1V_0$ and $V_1V_2$, in Figure 11. Four neighboring samples define a *sampling parallelogram*. Choosing the size of the sampling parallelogram such that its sides are shorter than $1/\sqrt{2}$ pixels guarantees that all covered

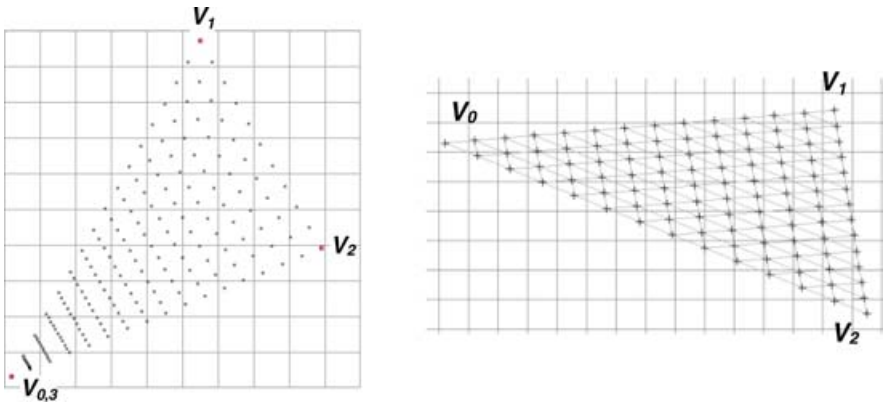


Fig. 10. Possible triangle interpolations. Left: degenerate quad, right: even sampling.

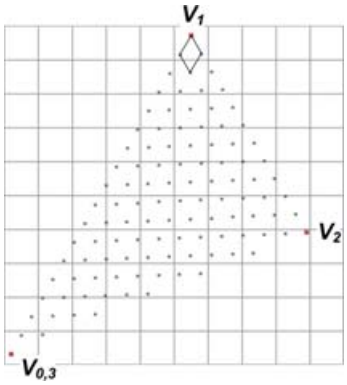


Fig. 11. Forward rasterization of triangle by barycentric sampling.

pixels receive one sample, but is overconservative. When the edges of a pair have slopes that belong to the same quadrant, smaller interpolation factors can be used while still guaranteeing that no holes remain.

5.1.1 *Same-Quadrant Edges.* If two triangle edges that share vertex $V_0$ are in the same quadrant, the following interpolation factors generate at least one sample for each pixel covered.

$$f_0 = \lceil \max(abs(u_0 - u_1), abs(v_0 - v_1)) \rceil$$
$$f_1 = \lceil \max(abs(u_0 - u_2), abs(v_0 - v_2)) \rceil \quad (7)$$

In Equation (7), $u_i$, $v_i$ are the image coordinates of vertex $V_I$; and the interpolation factors ensure that the interpolation step is small enough along both directions to prevent skipping an entire pixel. For edges that are more vertical than horizontal, the vertical span is used, while for edges that are closer to being horizontal, the horizontal span is used. In Figure 12, $AB$ and $AC$ are the sides of the sampling parallelogram, and the interpolation factors make sure that they are shorter than $1/\cos e_1$ and $1/\sin e_2$, respectively. In any new position of the sampling parallelogram obtained by translation that contains the center of the pixel, at least one sample will be on the perimeter or inside the pixel.

The interpolation factors given by Equation (7) are still overconservative, but computing the absolute coarsest interpolation factors that still guarantee that no holes remain in the case of same-quadrant edges is too expensive and defeats the purpose of forward rasterization [Popescu 2001].

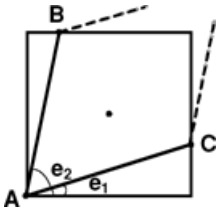


Fig. 12. Sampling parallelogram in same quadrant case.

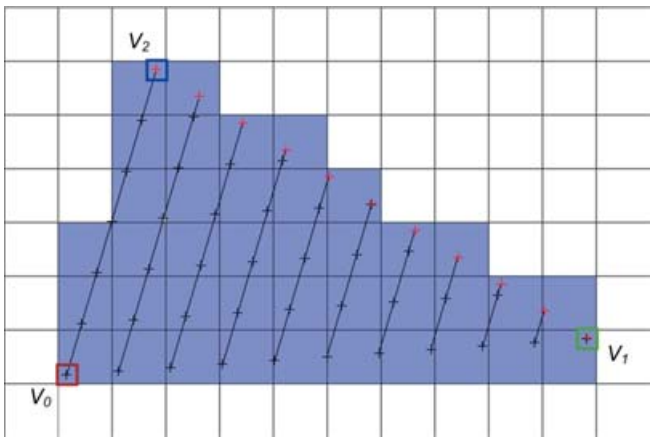


Fig. 13. Forward rasterization of triangle using edges in same-quadrant case.

The same sampling parallelograms are generated regardless of whether the triangle has the acute or obtuse angle of the parallelogram. Using Figure 12 again, the triangle could have the angle $BAC$ or the angle $CAB'$ where $B'$ (not shown) is on the extension of $BA$ beyond $A$. Since every edge crosses two quadrants, every triangle has at least one pair of edges that are in the same quadrant. Although a same-quadrant pair usually provides the best interpolation factors, we do check the other pairs for occasional improvements. Moreover, some triangles have more than one pair of edges in the same-quadrant case.

In conclusion, our algorithm examines all three possible pairs of edges. A pair of edges generates interpolation factors given by Equation (7) if they are in the same quadrant case and by Equation (8), an adaptation of Equation 1, if not. The pair with the smallest $f_0 f_1$ product is chosen.

$$f_0 = \left\lceil \sqrt{2} V_0 V_1 \right\rceil$$
$$f_1 = \left\lceil \sqrt{2} V_0 V_2 \right\rceil \tag{8}$$

Given a triangle that projects at image locations $V_0(u_0, v_0)$, $V_1(u_1, v_1)$, and $V_2(u_2, v_2)$, the forward rasterization algorithm generates samples according to the steps sketched in Appendix 1 and illustrated in Figure 13. Once the pair of edges ($V_0V_1$, $V_0V_2$) that yields the fewest number of samples is determined, samples are generated on lines parallel to edge $V_0V_2$. There are $f_0+1$ lines of samples. The first line is edge $V_0V_2$ and has $f_1+1$ samples, including $V_0$ and $V_2$. The last line has one sample, $V_1$. The number of samples decreases by $f_1/f_0$ from line to line. All samples created are known to be inside the triangle so no edge equation needs to be considered. In order to sample the third edge $V_1V_2$, each line ends with an additional sample on $V_1V_2$. In Figure 13, $f_0 = 10$, $f_1 = 6$, there are 11 sample lines, the first line has 7 samples, the last line has 1 sample. The number of samples decreases at a rate of 10/6 per line. The

samples generated on $V_1V_2$ are shown in red. The pixels that receive at least a sample are highlighted in grey.

## 5.2 Cost Analysis

As in the case of IBRW, we first compare the cost of forward rasterization to the cost of edge equation rasterization and then to the cost of barycentric rasterization.

5.2.1 *Rasterization Setup.* We have seen that the conventional rasterization setup cost for a triangle in the case of the edge equation approach is approximately 63 multiplications. The forward rasterization algorithm described first finds the interpolation factors using Equation (7) and Equation (8). The square of the lengths of the three edges cost 6 multiplications; deciding whether each pair of edges is in the same quadrant case has a total cost of another 6 multiplications. The square root needed by Equation (8) is looked up in a table, along with the values $1/f_0$ $1/f_1$, and $1/(f_0 f_1)$. Then the rasterization parameter increments are computed along each of the three edges, at a total cost of 18 multiplications, assuming that $R$, $G$, $B$, $u$, $v$, and $z$ are interpolated. The total forward rasterization setup cost for a triangle is 30 multiplications, approximately half of that of conventional rasterization.

5.2.2 *Sample Generation Loop.* The amortized per-sample cost is one addition per-rasterization parameter for both conventional and forward rasterization algorithms. We compare the performance of the sample generation loop by analyzing the number of times the inner loop is evaluated *ILN*, the number of samples generated *SGN*, and the number of pixels set *PSN*. The triangle in Figure 13 has a 10 by 6 pixel bounding box and thus, for the conventional rasterization, $ILN = 60$. For the forward rasterization algorithm $ILN = 50$, including the samples on the third edge. *PSN* is 29 and 39 for conventional and forward rasterization, respectively. Conventional rasterization generates a sample only if it is inside the triangle, thus $SGN = PSN = 29$. The algorithm described generates $ILN = 50$ samples which means that $ILN - PSN = 11$ pixels are overwritten. Redundant samples are particularly costly when shading is expensive, for example, involving several texture look ups. It is possible that samples within a pixel are sorted in back-to-front order, in which case deferring shading until after visibility does not help.

We use two methods for reducing the number of redundant samples. Both methods begin by replacing a triangle with one of its vertices if all three vertices map to the same pixel. Note that this simple but effective processing of very small triangles is only possible in forward rasterization: conventional rasterization needs to find the tiny triangle that contains the pixel center, and thus it needs to completely process all small triangles.

The first method discards a sample that maps to the same pixel as the previous sample. This method is called ED (early discarding) in Section 7. In Figure 14, 9 samples are discarded early, and only 2 pixels are overwritten. The shorter the step along the sampling lines, the greater the efficiency of the early discarding scheme. Given a pair of edges, the sampling lines can be chosen in 4 ways: parallel to either of the edges or parallel to either of the diagonals of the sampling parallelogram. We choose the direction that generates the smallest step. In Figure 14, the sampling step along $V_0V_1$ and both diagonals of the sampling parallelogram are longer than the sampling step along $V_0V_2$, which is chosen. Not all redundant samples are discarded since occasionally samples on different sampling lines map to the same pixel.

The second method (called EDL in Section 7) is more costly but avoids all overdraw within a triangle. The coordinates of the pixels set by the previous line of samples are recorded and consulted when a new sample is generated to ensure that none of the three neighbors on the previous line map to the same pixel as the current sample. Since the size and shape of triangles varies considerably within a scene model, we only compare forward and conventional rasterization on actual models (Section 7).

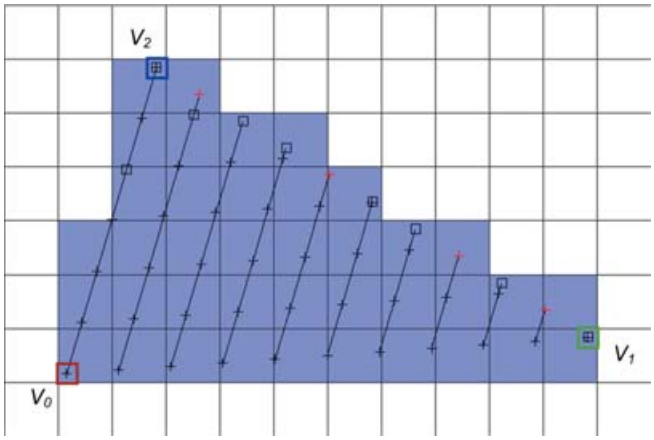


Fig. 14. Early discarded samples shown with black squares.

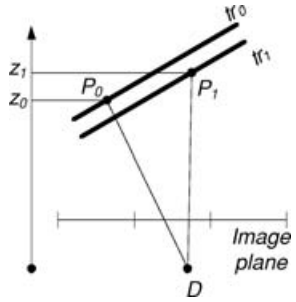


Fig. 15. Z-buffering along different rays in forward rasterization.

5.2.3 *Comparison to Barycentric Rasterization.* As shown earlier, the cost of barycentric rasterization setup is 17 multiplications which is lower than the 30 multiplications required by forward rasterization. However, this comes at the price of 3 multiplications per-rasterization parameter and per pixel, whereas forward rasterization, or edge equation rasterization for that matter, does not require any. In polygonal rendering, the number of rasterization parameters that are used as ingredients for the final pixel color is far larger than the 4 used in IBRW. Two-dimensional texture coordinates and 3D normals bring the tally to 9 rasterization parameters which require 27 multiplications per pixel. Except for very small (subpixel) triangles, the cost of barycentric interpolation makes it unattractive.

## 6. OFFSET RECONSTRUCTION

In forward rasterization, no inverse mapping from the image plane to the primitive is computed, thus one cannot generate a sample at a particular image plane location. The pixel grid is ignored except for determining interpolation factors that are sufficient to cover all pixels. The samples generated by forward rasterization can land anywhere inside the pixel. This has implications in z-buffering and in reconstruction/resampling.

### 6.1 Z-Buffering

Z-buffering samples that land at different locations within the pixel reduces the precision of the $z$ test, since the samples belong to different desired image rays. In Figure 15, the samples $P_0$ and $P_1$ are

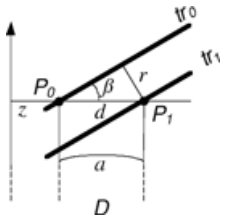


Fig. 16. Z-buffering precision study in forward rasterization.

generated by triangles $tr_0$ and $tr_1$ and map to the same pixel. Since $z_0 < z_1$, sample $P_0$ is incorrectly ruled as visible.

However, the situation depicted in Figure 15 is unlikely to occur. The angle between the two rays $DP_0$ and $DP_1$, where the $D$ is the center of projection of the desired image, is bounded by the angle spanned by a pixel. In the case of actual images with hundreds or thousands of pixels on each row, the angle spanned by a pixel is small.

We study the precision of z-buffering using Figure 16. The angle spanned by a pixel is $\alpha$, and the two triangles are parallel and form an angle $\beta$ with the image plane. The two rays that sample $P_0$ and $P_1$ intersect outside the figure at the desired image center of projection. Since $\alpha$ is small, the distance $d$ between $P_0$ and $P_1$ is comparable to $z\alpha$ when $\alpha$ is expressed in radians. The minimum distance between $tr_0$ and $tr_1$ that can be resolved is $r = d \sin\beta$ (a smaller distance brings $P_0$ closer to $D$ than $P_1$). For an image with 1200 pixels per row and a horizontal field of view of 60 degrees, $\alpha$ is approximately 1 milliradian. This means that $d$ is 1mm at 1m, and $r$ is 0, 0.5, 0.7, and 1mm when $\beta$ is 0, 30, 45, and 90 degrees, respectively. For a good reconstruction we use $2 \times 2$ supersampling which halves $\alpha$ and brings $r$ to 0, 0.25, 0.35, and 0.5mm for the four angle values. These precision estimates were confirmed experimentally.

We discuss the implications of the z-buffering precision loss and the solutions for alleviating the problem separately in the IBRW and polygonal rendering contexts.

In the case of IBRW, the desired image is rendered from a set of reference images acquired from nearby views. Because of this, the reference images do not sample the far surface of a pair of surfaces that are close together. The far surface is culled away which eliminates the risk of z-buffer fighting. During extensive testing in complex scenes (see Figure 24), the z-buffering precision limitation did not create artifacts.

The z-buffering precision loss is not unique to our method, but is rather a fundamental limitation of any forward mapping approach. In splatting, the reference samples project at different locations within a desired image pixel, yet they are z-buffered as if they were captured along the same ray. Moreover, the quality of the surface approximation provided by the splat degrades for the peripheral samples of the splat which implies a further decrease of the z-buffering precision. However, research reports on splatting do not mention visibility artifacts which supports our conclusion that, in the case of IBRW, the reduced z-buffering precision is not a concern.

In the case of polygonal rendering, the geometric model can contain two front facing surfaces that are sufficiently close together to create visibility artifacts. Examples are a sheet of paper on a desk or a painting on a wall. We discuss several approaches for alleviating the problem at the end of this section after we describe our reconstruction/resampling algorithm.

### 6.2 Reconstruction/Resampling

Forward rasterization does not compute the colors at the center of the pixels as expected by existing display technology. The truncation error introduced by assuming that the samples land at the

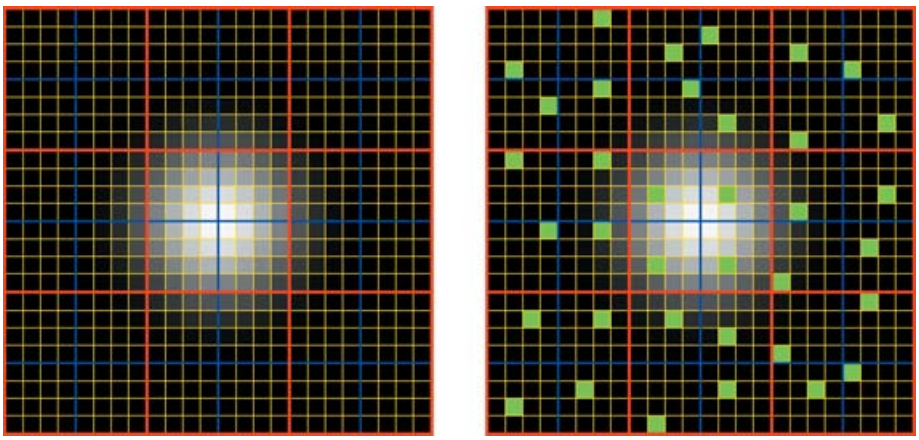

Fig. 17(a). Left: Output image pixels (red), 2 × 2 super-sampled visibility buffer (blue) and 4 × 4 virtual subdivision of the visibility buffer (yellow). The output pixel is computed using a raised cosine filter with a 2 × 2 output pixels base. Right: After all polygons are rasterized, each visibility buffer pixel contains exactly one sample (green squares).

Fig. 17(b). Image obtained without (left) and (right) offsets. The 8 × 8 magnifications in the bottom row show that offsets correctly create intermediate levels of grey which alleviate the jaggedness of the edges of the squares.

center of the pixels produces artifacts similar to the jaggedness caused by aliasing. We avoid this problem by using a pair of small integer offsets that record the actual location of each sample more accurately.

As in the case of conventional rasterization, more than one color sample per output pixel is needed to obtain high-quality images. For this, we render in an intermediate buffer of higher resolution than that of the output image. We call the intermediate buffer the visibility buffer. In Figure 17(a), the visibility buffer has twice the resolution of the output image in each direction. The location of the samples is recorded with a maximum error of $1/(8*\sqrt{2}) < 0.1$ output pixels using a pair of 2 bit offsets. The final pixel color is computed by convolution with a raised cosine filter that has a base of 2 × 2 output pixels, corresponding to 4 × 4 visibility pixels and thus 16 samples. The kernel stores 16 × 16 weights, one weight for each location of the virtual subdivision induced by the offsets. The output pixel is computed as a weighted average of the 16 samples covered by the kernel. The weight of each sample is given by the pair of offsets. Tuning the weight of the sample to its actual location produces a high-quality reconstruction as shown in Figure 17(b). Offset reconstruction is equivalent to reconstructing from an 8 × 8 super sampled buffer that is sparsely populated; the offsets act as pointers and do not require allocating and managing such a buffer of prohibitive cost.

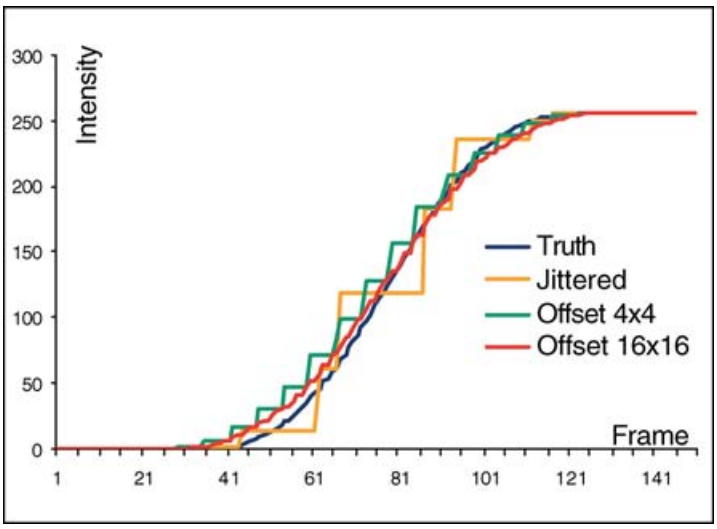


Fig. 18(a). Pixel intensity variation with various reconstruction techniques.

Offset reconstruction implements the inverse mapping from the image plane to the primitive and allows computing the color at the center of the output pixel. When compared to conventional rasterization, one advantage is improved performance. The inverse mapping is computed after visibility, thus only for the visible samples. The inverse mapping is also considerably less expensive than conventional rasterization setup: the main additional cost is storing 4 bits at every visibility buffer pixel.

Another advantage is improved quality for both single frames and frame sequences. When supersampling is used in conventional rasterization, the input samples (reference image color samples for IBRW or triangle vertex colors for triangle meshes), are first blended to create the color samples at the supersampling locations, and then these color samples are blended again to create the final image. This additional resampling is avoided in the case of forward rasterization where the original reference samples are filtered only once to produce the final image pixels. This advantage is important for small primitives when the density of original samples per output pixel is large. Offset reconstruction produces sharper images, an advantage illustrated in Section 7, Figure 25, in the case of an actual model. Offset reconstruction also has good temporal antialiasing properties.

### 6.3 Temporal Antialiasing

In the case of conventional rasterization, the sampling locations are defined with respect to the pixel grid. The best approach is to use an irregular pattern to avoid having three or more collinear sampling locations. Figure 18(a) shows a $3 \times 3$ pixel fragment of the framebuffer. A red and a blue polygon share an edge, shown in magenta. The polygons move slowly upwards. Both methods use 4 color samples per pixel and both compute the final pixel values using a raised cosine filter with a 2-pixel base.

In the case of conventional rasterization, the intensity of the central pixel remains constant while no samples change sidedness with respect to the magenta line. When a sample changes sidedness, the intensity of the blue (or red) channel of the output pixel varies abruptly which produces a noticeable artifact. In the case of forward rasterization, the samples move with the polygon. Their new position is reflected by the offsets which select appropriate weights, and the intensity change is gradual.

Figure 18(b) plots the intensity of the blue channel of the central pixel shown in Figure 20 as the pixel is progressively covered by the blue polygon. The truth graph was obtained using $128 \times 128$

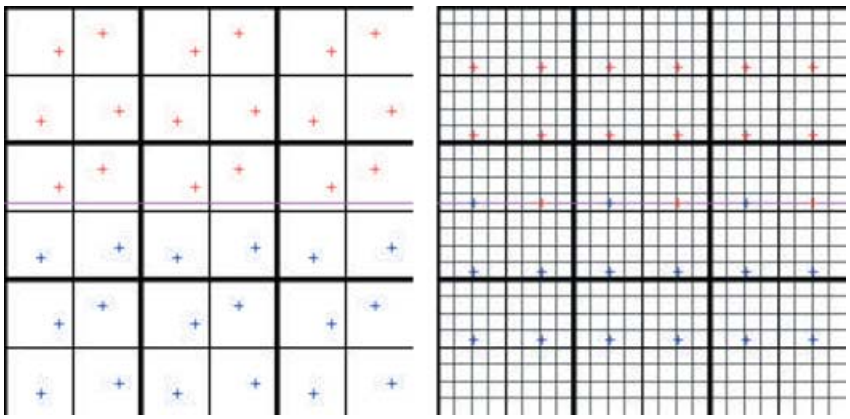

Fig. 18(b). Jittered supersampling (left), and forward rasterization followed by offset reconstruction (right).

regular super-sampling. Conventional rasterization followed by jittered super sampling produces only a few intermediate intensity levels. 2-bit offsets alleviate the intensity jumps. 4-bit offsets (with an aggregate storage cost of 1 byte per pixel) produce a smooth curve.

The case shown here is unfavorable for offseting reconstruction since the edge is parallel to the visibility locations and to the virtual grid induced by the offsets. When the edge is at an angle, even 2-bit offsets produce a smooth curve. The worst case for jittered super-sampling is when the edge is parallel to the line defined by two sampling locations, a case in which the intensity jumps are even more important, and the benefit of offsets is even more salient.

### 6.4 Z-Buffering Precision

As stated earlier, the limited z-buffering precision characteristic to forward rasterization is unlikely to create visibility artifacts in the case of IBRW but could create problems in the case of polygonal rendering when two front facing surfaces are close together.

Alleviating the visibility problems can be done in one of several ways. The limited z-buffering precision is also an issue in the case of conventional rasterization, and the application-level solutions used there are applicable in the context of forward rasterization. One approach is to sort the surfaces that might create problems, draw the far surface first, and then draw the near surface without z-buffering but with stenciling to confine the near surface to the image region of the far surface. Another approach is to draw the near surface with a z-offset.

Several application-transparent solutions are possible. $z$ precision improves with the output resolution, according to Figure 16. For a given output resolution, $z$ precision increases with the resolution of the visibility buffer. A higher level of super-sampling has the positive side effect of increasing the antialiasing quality by increasing the number of color samples per output pixel. The disadvantage of the solution is the high cost associated with the additional supersampling.

Another solution is to treat $z$ $(1/z)$ as a special rasterization parameter, for which the image plane variations are computed during rasterization setup as in the case of inverse rasterization. Knowing the one pixel width $(dz_u)$ and one pixel height $(dz_v)$ $1/z$ increments allows the reduction of the image plane distance between the rays along which the depth comparisons are performed. We describe two variants of this solution which trade $z$ precision with cost. We begin with the costlier but more precise solution.

6.4.1 *Depth Comparison at the Center of the Visibility Pixel.* This approach achieves the same precision as conventional rasterization. The idea is to compute, for every sample, the $1/z$ value at the center

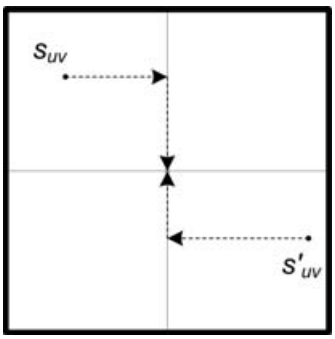


Fig. 19. Depth comparison at the center of the visibility pixel. The new ($s'_{uv}$) and old ($s_{uv}$) samples are moved to the center of the visibility pixel for depth comparison.

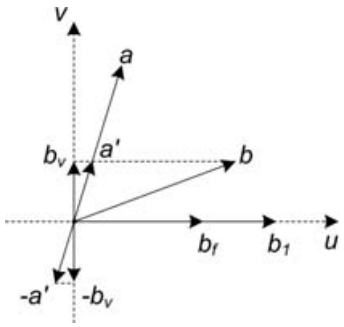


Fig. 20. Geometric computation of 1-pixel $1/z$ increments.

of the visibility pixel. This requires computing $dz_u$ and $dz_v$, and then using them at sample $(u, v)$ to compute $1/z$ at (floor($u$)+ .5, floor($v$)+ .5) by adding the correction ((floor($u$)+.5−$u$)$dz_u$, (floor($v$)+.5−$v$)$dz_v$), see Figure 19.

The increments $dz_u$ and $dz_v$ can be computed more efficiently than in the case of conventional rasterization, since $1/z$ is the only rasterization parameter for which these 1-pixel increments are needed. In Figure 20, let vectors $a$ and $b$ define the two edges of the sampling parallelogram. The rasterization algorithm computes the $1/z$ variations for vectors $a$ and $b$. The $1/z$ variation for vector $b_1$, which spans 1 pixel width, is computed according to Equation (9). Increment $dz_v$ is computed similarly.

The additional setup cost is 2 divisions ($v_b/v_a$, and $dz_f/u_{bf}$) and 2 multiplications ($(v_b/v_a) * u_a$,and ($v_b/v_a$)*$dz_a$) for each of $dz_u$ and $dz_v$, totaling 4 divisions and 4 multiplications. For each, sample, the depth correction costs 2 multiplications. This solution perfectly aligns the rays along which visibility is computed, completely eliminating the precision loss to achieve the same precision as conventional inverse rasterization.

$$\begin{aligned} a' &= \frac{v_b}{v_a}a, b_f = b - a', u_{b_f} = u_b - \frac{v_b}{v_a}u_a, b_1 = \frac{1}{u_{b_f}}b_f \\ dz_f &= dz_b - \frac{v_b}{v_a}dz_a, dz_u = \frac{dz_f}{u_{b_f}} \end{aligned} \tag{9}$$

6.4.2 *Depth Comparison at the Same Offset Location.* This second approach does not completely eliminate the image plane distance $e$ between the rays along which the depth comparison is performed,

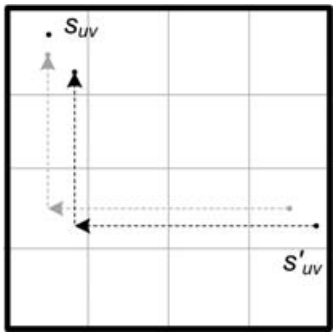


Fig. 21. Depth comparison at same offset location. A single visibility pixel is shown, virtually subdivided $4 \times 4$ by a pair of 2-bit offsets. In the depth comparison between $s$ and $s'$ is translated to the offset location of $s$.

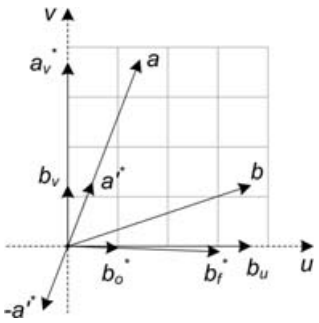


Fig. 22. Approximate geometric computation of one-offset location $1/z$ increments.

rather it reduces it below the size of an offset location. For a $2 \times 2$ visibility buffer super-sampling and a $4 \times 4$ offset virtual super-sampling, $e$ is guaranteed to be less than 1/8 pixels. For a total storage cost of 1 byte, the pair of 4 bit offsets reduces $e$ to less than 1/32 pixels. As in the case of color reconstruction, offsets are used as an effective but inexpensive method for controlling the approximation errors stemming from not enforcing that pixels land at pre-established image plane locations.

The distance $e$ is reduced by applying a depth correction that moves the new sample $s'$ at the offset location where the current sample $s$ resides within the visibility pixel (Figure 21). In this example, the depth correction applied to the $1/z$ value of $s'$ is $(0-3)dz_{ou} + (0-2)dz_{ov}$, where $dz_{ou}$ and $dz_{ov}$ are the $1/z$ variations for a one offset location horizontal and vertical step. The multiplications with the small integers that range from $(n-1)$ to $(n-1)$ can be precomputed in a table with $(n-2)$ entries, where $n$ is the number of possible offset values, 4 in this example. This reduces the per pixel cost to two adds.

The steps $dz_{ou}$ and $dz_{ov}$ are computed approximately to reduce the setup cost as shown in Figure 22. The division with $a_v$ is avoided by discretizing the $a_v$ to the nearest $1/2^k$ pixel. The inverse of the integer is looked up and the division is replaced with a multiplication. Similarly, the final division that normalizes $dz_f$ (see Equation (9)) is replaced by a multiplication, bringing the additional setup cost to 8 multiplications.

Figure 23 shows the results of offset z correction on a test scene. The pairs of squares form a $30^o$ angle with the $z$ plane and their center is situated at a depth of 1m. The location within the field of view does not affect the $z$ precision since the size of the pixel, and thus of the visibility pixel and of the offset locations, is the same. The top-left image shows the squares interpenetrating since they are closer (0.125 mm) than the smallest distance that the $z$-buffer can resolve under these conditions (0.5 mm).

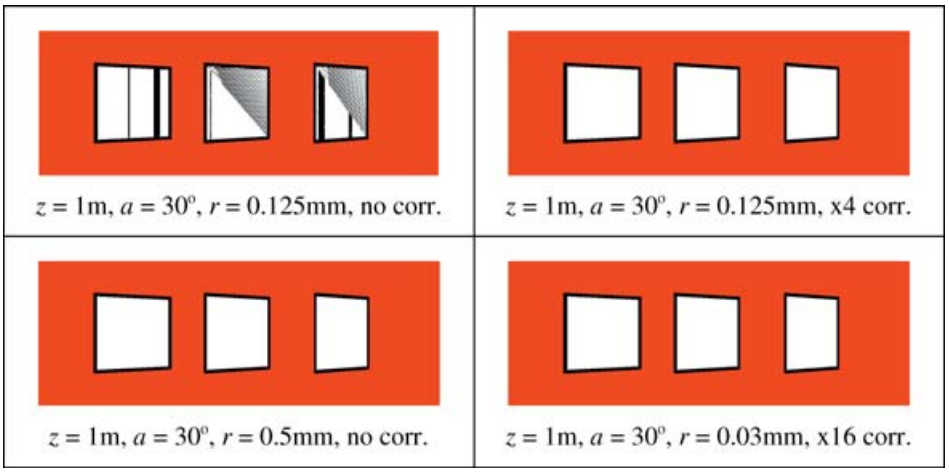


Fig. 23. Study of z-buffer precision. The scene consists of 3 pairs of squares. The squares in a pair are parallel and close together. The white square is slightly smaller and closer than the black square. The correct image is a solid white square with a black frame.

The top-right image shows that the artifacts are eliminated if a depth correction corresponding to 2-bit offsets is applied. The bottom-left image shows that no artifacts occur without correction if the distance $r$ is 0.5 mm, confirming the precision expression derived earlier using Figure 16. The last image shows that the z-precision becomes 30 microns if 4-bit offsets are used.

In conclusion, the forward rasterization algorithm has a z-buffering precision given by the maximum possible distance, measured in pixels, between the rays of two samples that map to the same visibility pixel and thus compete for visibility. Therefore, the z-buffering precision increases with the resolution of the output image, the resolution of the visibility buffer, and the angle between the viewing direction and the surface. For many scenes, the z-precision is sufficient. A higher z-precision can be obtained with various techniques that trade precision with efficiency, including the precision of equivalent inverse rasterization.

Finalizing the z-buffering approach for forward rasterization depends on a careful future study of the various implications on the architecture and the hardware implementation. The approach of using the offset mechanism to reduce the gap between the rays used for visibility testing seems the most promising since it offers good precision at a moderate cost, and a simple mechanism for increasing the precision as desired.

## 7. RESULTS AND DISCUSSION

### 7.1 Quadrilateral Forward Rasterization

We have tested quadrilateral forward rasterization in the context of IBRW. The test scene is shown in Figure 24. Using the geometric model of the town scene, depth panoramas were prerendered from each pink node of the purple wire frame grid. A depth panorama consists of four 1600 $\times$ 1200 depth images with a $90^o$ horizontal field of view, arranged like the four lateral faces of a cube.

Figure 25 shows an image from a novel view rendered by warping nearby reference depth images. The novel view has a resolution of 1600 $\times$ 1200 and a horizontal field of view of $50^o$, thus the sampling rate is about half that of the reference images in each direction. The warped mesh is disconnected to not cross depth discontinuities. Depth discontinuities are detected in each reference image as a preprocess by thresholding the second-order generalized disparity difference [Popescu et al. 2000]. The warped

Fig. 24. *EuroTown* test scene for quad forward rasterization.


Fig. 25. Top: reference images (1600 × 1200). Bottom: novel view (1600 × 1200).

meshes were rasterized with our method (2 × 2 super-sampling, 4 × 4 offsets, 2-pixel raised cosine kernel) and conventionally (2 × 2 super-sampling, 2-pixel raised cosine kernel) for comparison. Forward rasterization produces images of quality comparable to that of images rendered with conventional rasterization which has been the gold standard in forward rendering.

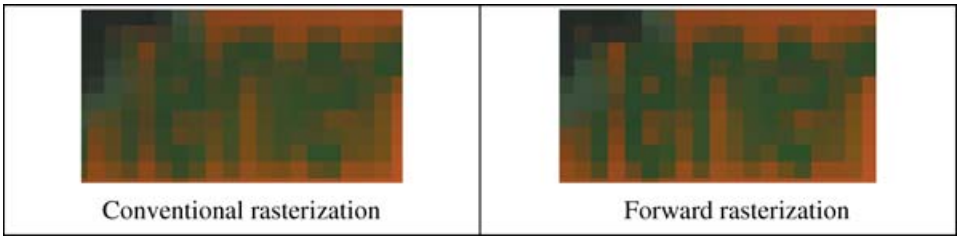


Fig. 26. Comparison of magnified fragments.

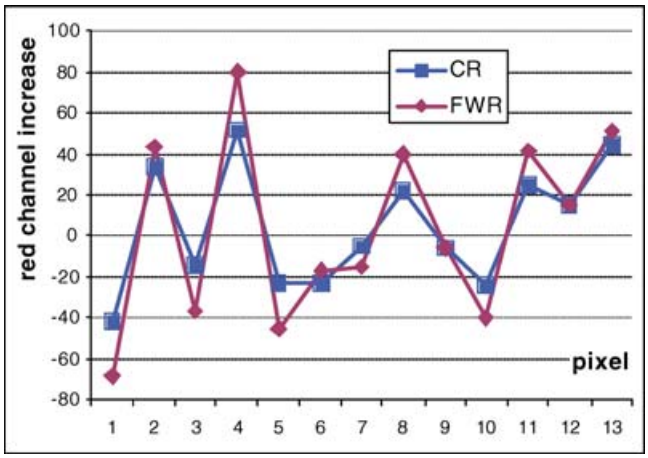


Fig. 27. Pixel to pixel red channel variation, comparison between conventional (CR) and forward (FWR) rasterization.

When the sample density is large, forward rasterization, followed by offset reconstruction, produces higher quality images than conventional rasterization for the same reconstruction kernel because conventional rasterization implies an additional resampling. In conventional rasterization, the color samples of the super-sampled buffer are computed by blending the original samples which are used as texels or as vertex colors. Then the color samples of the super-sampled buffer are blended again to form the final pixel colors. In forward rasterization, when the visibility buffer sampling density is comparable or lower to that of the reference image, the visibility buffer will essentially be populated with reference samples. This is because the forward rasterization of the small primitives is given by the vertex samples, and no interpolation occurs. Consequently, the final pixel colors are directly computed from the input color samples.

We rendered the view shown in Figure 25 at a resolution of $800 \times 600$ which implies a density of reference samples in the visibility buffer of approximately 1. The images in Figure 26 are a $16\times$ magnification of the orange store front and show that the forward rasterized image is less blurry. Figure 27 plots the difference between the red channel of the current pixel and the red channel of the previous pixel for a few consecutive pixels on the same row for each of the two images from Figure 26. The variation has larger amplitude in the case of forward rasterization.

The results presented so far assumed $2 \times 2$ super-sampling. Applications might want to opt for one color sample per pixel to achieve a higher frame rate or output resolution. Forward rasterization supports rendering with one color sample per pixel. Offset reconstruction is still needed to account for the free positioning of the samples within the pixel. Not using offsets would produce worse results than

Fig. 28. Forward rasterization with one color sample per pixel, without (left) and with $4 \times 4$ offsets (right). The bottom half of each image is an $8 \times 8$ magnification of a fragment of the top half. Both images are reconstructed with a $2 \times 2$ raised cosine kernel which for the left image is equivalent to arithmetic averaging.

the equivalent conventional rasterization. Figure 28 shows the results of forward rasterization with a single color sample per output pixel. The offsets create a few intermediate levels of grey which reduce the jaggedness of the edge. Offsets improve single sample rendering substantially at a small additional cost.

When attempting to compare the running times of the two methods, there are several options. The first option is to run the two methods on the CPU, but the relevance of the results is limited by the particulars of the implementations of each algorithm and by the use of general purpose architecture not optimized for rendering. The second option is to conventionally rasterize in hardware and to forward rasterize on the CPU. It comes at no surprise that existing graphics hardware can conventionally rasterize the warped mesh much faster than the CPU can forward rasterize the same mesh of quads.

The third option is to take advantage of the programmability of today's hardware and forward rasterize on the GPU. The sample generation loop can be implemented as a vertex program. Since GPUs do not offer programmability at primitive level, the quad (or triangle) data needs to be replicated for each vertex. The offset reconstruction would be a simple fragment program. However, the GPU is optimized for conventional rasterization and the comparison would not be meaningful. We are looking forward to the upcoming DX geometry shaders which will offer programmability at primitive level and might be appropriate for an efficient implementation of forward rasterization.

To fully take advantage of the potential performance advantage of forward rasterization described in Sections 4, 5, and 6, a dedicated hardware implementation is needed [Popescu et al. 2000]. Therefore, we do not report timing data and limit the performance comparison to the number of times the inner loop is executed and the number of samples generated. We have measured sample generation performance for views along a path through *Eurotown*. For each view and each reference image used, we have computed four quantities: the average length of the quad edges in output pixels (one output pixel corresponds to two visibility pixels), the ratio between the number of times the inner loop is executed (*ILN*, Section 5) in forward versus conventional rasterization, the overdraw as the ratio of the number of samples generated by forward versus conventional rasterization, and the overdraw with early discarding (*ED*, see Section 5) as the ratio of the number of samples that pass early discarding in forward rasterization, and the number of samples generated by conventional rasterization.

The first four rows of the table in Figure 29 report the minima, maxima, averages, and medians for each of these four quantities. The last row reports the four quantities for the view and the reference image that had the median overdraw ED value. The numbers vary little in the columns since

| | Quad size (pix) | Inner loop FWR/CR (%) | Overdraw | Overdraw ED |
|---|---|---|---|---|
| **Min** | 1.35 | 47 | 2.44 | 1.76 |
| **Max** | 4.12 | 88 | 2.97 | 2.11 |
| **Average** | 1.83 | 65 | 2.64 | 1.90 |
| **Median** | 1.72 | 66 | 2.60 | 1.87 |
| **Median overdraw** | 2.24 | 53 | 2.60 | 1.87 |

Fig. 29. Analysis of sample generation for quad forward rasterization.

every view is rendered with nearby reference views. Forward rasterization executes the inner loop less frequently than conventional rasterization, which is expected, since the quads are parallelograms or even rectangles (see Figure 7 and Figure 8). Moreover, forward rasterization only samples the top and left edge of every quad to avoid sampling the edges redundantly, whereas the bounding boxes of triangles of neighboring quads typically overlap in conventional rasterization.

If all the quads were parallelograms with an angle $\theta$, the theoretical overdraw $o_t$ would be $1/(1/\sqrt{2} * 1/\sqrt{2} * \sin\theta)$. For a rectangle $o_t$ is 2. The overdraw measured is larger because the quads are not exactly rectangular and because of the ceiling operation in Equation (1). Early discarding reduces overdraw below 2.

### 7.2 Triangle Forward Rasterization

We have rendered several triangle meshes with conventional and forward rasterization (Figure 30). Both methods rendered in a $2 \times 2$ super-sampled visibility buffer. The final image was reconstructed by convolution using a raised cosine with a 2-pixel base. Forward rasterization used 2 bit offsets.

The second column in the table of Figure 31 gives the average edge size of the triangles projected on the image plane. For each model, the table gives data for two views. The first view is shown for each model in Figure 30. The second view is closer to the model, thus the triangles are bigger. The second column gives the ratio between the number of times the inner loop is executed (*ILN*) for the forward and the conventional rasterization algorithms. *ILN* is usually smaller in the case of forward rasterization. When triangles are very small (1.2 pixels in size), *ILN* is slightly larger in the case of forward rasterization because the ceiling operation introduces a relatively larger increase of the interpolation factors (Equation (7) and Equation (8)).

The fourth column measures the performance of early discarding by looking only at the previous sample (*ED*, see Section 5), compared to conservatively discarding all redundant samples within a triangle (*EDL*). The figures are computed as the ratio between the number of samples discarded by *ED* versus *EDL*. The simple mechanism of not generating a sample if it maps to the same pixel as the previously generated sample avoids a large percentage of the redundant samples. The fifth column reports the average number of times a pixel is overwritten by samples from the same triangle. The figures are computed as the ratio between the number of samples that pass *ED* versus *EDL* and show that a pixel gets on average only about 1.3 samples.

The last column reports the overall overdraw ratio; the figures are computed as the ratio between the number of samples that pass *EDL* versus the number of samples generated by conventional

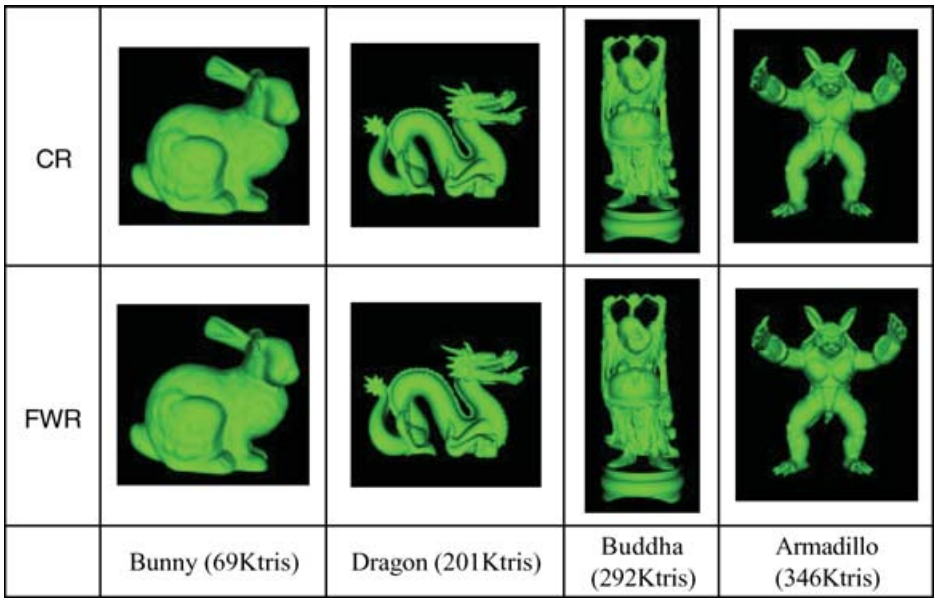


Fig. 30. Images produced by conventional rasterization (top row) and forward rasterization (bottom row).

| | Tri. size (pix) | Inner loop FWR/CR (%) | ED (%) | Overdraw inside tris. | Overdraw overall |
|---|---|---|---|---|---|
| **Bunny** | 3.0 | 75 | 51 | 1.27 | 1.57 |
| | 6.0 | 64 | 31 | 1.26 | 1.32 |
| **Dragon** | 2 | 90 | 55 | 1.34 | 2.01 |
| | 3.5 | 72 | 39 | 1.35 | 1.44 |
| **Buddha** | 1.2 | 119 | 68 | 1.32 | 2.92 |
| | 2.2 | 89 | 52 | 1.36 | 1.88 |
| **Armad.** | 1.2 | 112 | 75 | 1.22 | 2.8 |
| | 2.0 | 86 | 62 | 1.24 | 1.86 |

Fig. 31. Analysis of sample generation for triangle forward rasterization.

rasterization. In addition to the overdraw inside the triangle, forward rasterization overdraws shared edges (and vertices). The relative importance of the triangle edges quickly decreases with the size of triangles as confirmed by the overdraw figures for triangles of 3–6 pixels in size. All overdraw numbers are computed before z-buffering.

We now compare the images produced by the two methods, see Figure 32. The images are very similar. For interior pixels, the difference is 0, 1, or 2 for each channel, and it is distributed randomly. The largest difference between the images is encountered at the silhouette of the bunny.

In conventional rasterization, a foreground object wins the silhouette pixels if the silhouette line encloses the pixel centers (or sampling locations when super-sampling is used). The foreground object is thicker or thinner according to its relative position with respect to the pixel raster, and we have seen that a slowly moving foreground object conquers pixels at a nonuniform rate, producing artifacts.

In forward rasterization, the samples of the foreground object always win every pixel they map to because they are closer. However, the offsets record the peripheral position of such samples and decrease

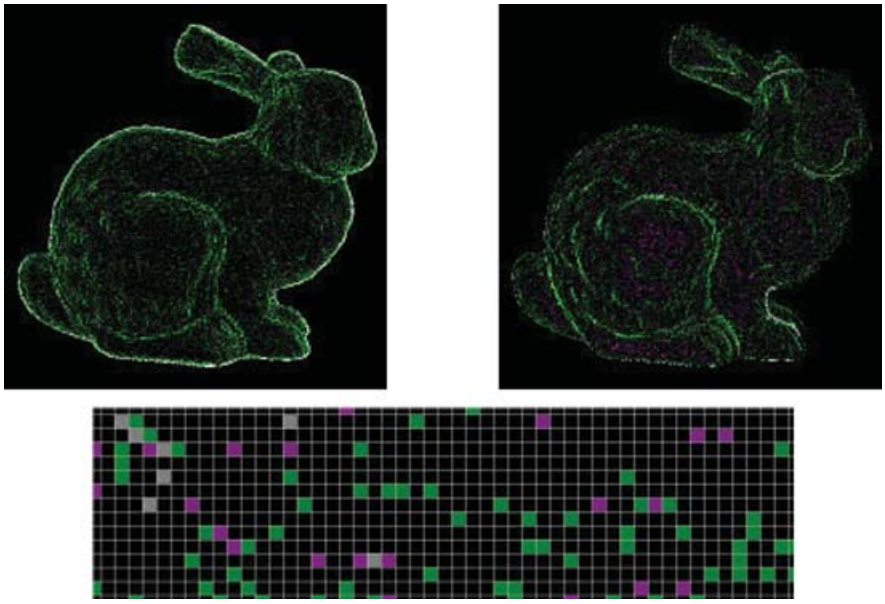

Fig. 32. The top-left image shows the difference between the forward ($I_{FWR}$) and the conventional rasterized ($I_{CR}$) images of the bunny given in Figure 30. The difference is truncated to 0 and multiplied by 127. The top-right image shows the difference between $I_{CR}$ and $I_{FWR}$. The bottom image is a 16-fold magnified fragment of the top left image.

their importance during reconstruction which makes the bias towards the foreground object small. In Figure 32, the average difference between the green channels of the silhouette pixels is 9. This does not account for the fact that, in conventional rasterization, the foreground object is too small for some parts of the silhouette. A comparison to a truth image computed with a very high level of antialiasing would reveal that the intensity difference is even smaller.

## 8. CONCLUSION

We have described forward rasterization, a class of algorithms that decomposes the polygonal primitives in samples by interpolation. Enough samples are generated to guarantee that each pixel covered by the primitive receives one sample. The samples are generated in the input domain and can map anywhere on the pixel grid. A pair of offsets records the location of the sample within its pixel. Offsets effectively decouple visibility from reconstruction: we compute visibility at half pixel resolution ($2 \times 2$ supersampling), and the samples are stored at one eighth pixel resolution. The offsets tie the samples to the output pixel centers and implement an inexpensive but accurate approximate inverse mapping from pixel centers to the input domain. The inverse mapping is computed just in time, only for the visible samples (Figure 33).

When compared to conventional rasterization, forward rasterization has the advantage of less expensive rasterization setup and better static and temporal antialiasing properties. When compared to barycentric rasterization, forward rasterization setup has a comparable cost in the case of IBRW, and a slightly higher cost in the case of triangle rendering. However, barycentric rasterization has a substantially higher per pixel cost (inner loop cost).

Conventional rasterization has the advantage of generating the minimum number of samples required for coverage. Early discarding reduces the number of samples that go to the same pixel in the case of forward rasterization. When compared to prior splatting techniques, forward rasterization

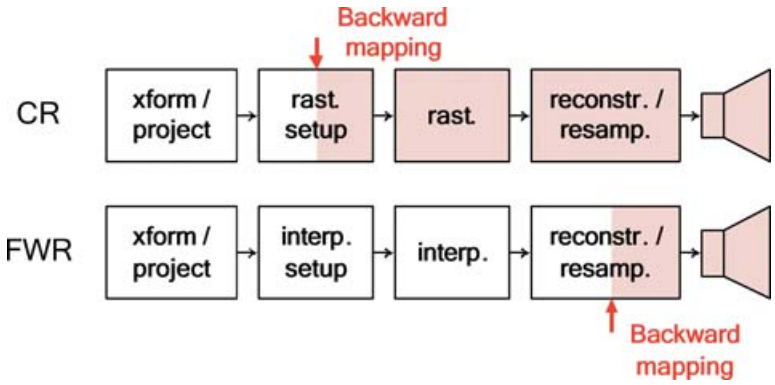


Fig. 33. Conventional (CR) and forward rasterization (FWR) pipelines.

has the advantages of guaranteed surface coverage and high-quality reconstruction. Prior splatting techniques do not report the amount of overdraw. Based on our own experience with splatting, alleviating the problem of holes requires an important overestimate of the splat size which produces overdraw ratios larger than 4 (each pixel ends up being covered by four splats or more) and a low quality reconstruction.

As mentioned in the Section 2, Whitted and Kajiya [2005] described a fully procedural pipeline that is related to forward rasterization. The forward rasterization approach can be seen as handling quads and triangles procedurally. Of course, forward rasterization is not a full blown procedural pipeline. For example, higher-order primitives still need to be tessellated, at least with quads, before they are fed to the forward rasterization pipeline. This implies that forward rasterization does not limit the external (input) bandwidth as the procedural pipeline, which defers the data amplification by directly accepting procedural element, does.

However, forward rasterization overcomes several of the challenges of a fully procedural pipeline identified by Whitted and Kajiya in their report. One of them is the additional computational cost due to projecting each sample individually. Forward rasterization avoids the problem since the samples are generated after projection. Another challenge is the amount of overdraw. In the fully procedural pipeline work, a coarse, overconservative, estimate is used to determine a sampling rate that avoids holes. For forward rasterization we have derived tight, yet conservative, interpolation factors, and the overdraw is reduced by a factor of two. A third challenge is the cost of evaluating the procedure for each sample. Since in forward rasterization the primitives are simple (quads or triangles), samples are generated incrementally with the amortized cost of one add per rasterization parameter.

Efficiency is not the only penalty for overdraw. Some algorithms (e.g. compositing, shadow volumes) produce artifacts if a primitive touches a pixel more than once. Such algorithms can only be supported by devising techniques that avoid all overdraw. The *EDL* technique presented avoids overdraw within the triangle, but we have yet to investigate eliminating the overdraw at the shared edges. Generating samples on a space-filling curve might provide improved *ED* performance. With z-buffer correction, a fuzzy z-buffer test could catch all redundant samples, but this approach increases the cost of these samples since their identification requires consuming z-buffer bandwidth.

A complete analysis of the computer graphics architecture implications of forward rasterization remains to be conducted. The first step is to study the dynamic 3D graphics workload produced by forward rasterization based on similar studies conducted for conventional rasterization [Mitra and Chiueh 1999]. The reduced setup cost advantage vanishes for large triangles. It remains to be seen whether

in this case the static and temporal antialiasing advantages are sufficient to offset the additional cost introduced by the redundant samples. We will investigate subdividing large triangles 50 as to minimize the number of redundant samples. We will design a coarse rasterizer best suited for forward rasterization, similar to the ones used in conventional rasterization, to break up large triangles in rectangular tiles and left-over triangles.

The cost analysis presented here focuses on computational cost. The bandwidth requirement is an important component of the total cost of an algorithm and an important factor when deciding its suitability for hardware implementation. The number of interpolants and the number of samples generated begin to outline the bandwidth requirements for forward rasterization. However, a detailed analysis of these requirements is still in the future as it is tied to actually devising a hardware architecture that implements forward rasterization. Our method will be compared again to other rasterization algorithms under this new angle. Barycentric interpolation, for example, trades computational cost (chip area) for a reduction of the bandwidth between the rasterizers and the fragment units.

Also as future work, we will perform a signal processing analysis of forward rasterization and attempt to design a reconstruction kernel optimized for offset reconstruction that will eliminate the small difference between the $16 \times 16$ offset reconstruction and the truth graphs in Figure 18.

Important future work will adapt texture mip-mapping and level of detail to forward rasterization. An option is to restrict the possible interpolation factors $f_1$ and $f_2$ to values that are power of two, and to store mipmaps based on these fixed values. Another important extension which will increase the usability of forward rasterization in the context of today's Graphics hardware is support for multisampling as a cost effective alternative to super-sampling.

Forward rasterization can readily replace conventional rasterization to provide infrastructure support for higher-level computer graphics algorithms. Offset reconstruction offers independence from the pixel grid. The technique could be used to avoid resampling errors whenever an intermediate sample-based representation is constructed (e.g. LDIs, acceleration by repeated reprojection of prerendered samples). Another possible use that we will investigate is to optimize sampling. For example, geometry images [Gu et al. 1992] augmented with a vertex per pixel that is freely placed within the pixel boundaries have increased modeling power that comes at a small additional cost.

## APPENDIX 1

Sample generation algorithm for forward rasterization of triangles. The triangle vertices project at $V_0$, $V_1$, and $V_2$. The samples generated are illustrated in Figure 13.

**For** each of three possible pairs of edges ($V_0^i V_1^i$, $V_0^i V_2^i$)
  **If** $V_0^i V_1^i$ and $V_0^i V_2^i$ are in same quadrant compute ($f_0^i$, $f_1^i$) with Equation 7
  **Else** compute ($f_0^i$, $f_1^i$)with Equation 8.
**EndFor**
Choose pair of edges with smallest $f_0 f_1$ product; let that be ($V_0V_1$, $V_0V_2$).
$uLine = u_0$; $duLine = (u_1 - u_0)/f_0$; $vLine = v_0$; $dvLine = (v_1 - v_0)/f_0$;
$samplesN = f_1 + 1$; $dsamplesNLine = f_1/f_0$;
$du = (u_2 - u_0)/f_1$; $dv = (v_2 - v_0)/f_1$;
**For** $lineIndex = 0$ **to** $f_0$
  $u = uLine$; $v = vLine$;
  **For** $sampleIndex = 0$ **to** $samplesN$
    UseSample($u$, $v$);
    $u$ +=$du$;
    $v$ +=$dv$;
  **EndFor**
  $u = u_2 + lineIndex/f_0 * (u_1 - u_2)$, $v = v_2 + lineIndex/f_0 * (v_1 - v_2)$;

UseSample($u$, $v$)
*samplesN* = floor*(samplesN – lineIndex*dsamplesNLine);*
*uLine* = *uLine* + *lineIndex*duLine; vLine* = *vLine* + *lineIndex*dvLine;*
**EndFor**

ACKNOWLEDGMENTS

The authors are grateful to John Eyles, Anselmo Lastra, Gary Bishop, Joshua Steinhurst, Nick England, Lars Nyland, John Poulton, and Chris Hoffmann for numerous fruitful discussions. We thank John Owens for promptly providing reviewers and a hard to find technical report. The anonymous reviewers, the associate editor, and the editor in chief have generously donated many hours of their time towards improving this manuscript, for which we thank them sincerely.

REFERENCES

Abram, G. and Westover, L. 1985. Efficient alias-free rendering using bit-masks and look-up tables. In *Proceeding of SIGGRAPH*. 53–59.

Akeley, K. 1993. RealityEngine Graphics. In *Proceedings of SIGGRAPH*. 109–116.

Aliaga, D. and Lastra, A. 1999. Automatic image placement to provide a guaranteed frame rate. In *Proceedings of SIGGRAPH*. 307–316.

Ati. http://www.ati.com.

Brown, R. 1999a. Barycentric coordinates as interpolants. Sun Microsystems Tech. rep.

Brown, R. 1999b. Modeling specular highlights using bezier triangles. Sun Microsystems Tech. rep.

Carpenter, L. 1984. The A-Buffer, an antialiased hidden surface method. In *Proceedings of SIGGRAPH*. 103–108.

Chang, C., Bishop, G., and Lastra, A. 1999. LDI Tree: A hierarchical representation for image-based rendering. In *Proceedings of SIGGRAPH*. 291–298.

Cook, R. 1986. Stochastic sampling in computer graphics. *ACM Trans. Graph. 5*, 1, 51–72.

Cook, R., Carpenter, L., and Catmull, E. 1987. The Reyes image rendering architecture. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. 95–102.

Deering, M. and Naegle, D. 2002. The SAGE graphics architecture. In *Proceedings of SIGGRAPH*. 683–692.

Microsoft DirectX, multimedia API. http://www.microsoft.com/windows/directx/default.aspx

Duguet, F. and Drettakis, G. 2004. Flexible point-based rendering on mobile devices. *Point-Based Comput. Graph.* 57–63.

Ellsworth, D. 1996. Polygon rendering for interactive visualization on multicomputers. Ph.D. Dissertation, Computer Science Department, University of North Carolina at Chapel Hill, NC.

Eyles, J., Molnar, S., Poulton, J., Greer, T., Lastra, A., England, N., and Westover, L. 1997. PixelFlow: The realization. In *Proceedings of the SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*. 57–68.

Fuchs, H., Goldfeather, J., Hultquist, J., Spach, S., Austin, J., Brooks, F., Eyles, J., and Poulton, J. 1985. Fast spheres, shadows, textures, transparencies, and image enhancements in pixel-planes. In *Proceedings of SIGGRAPH*. 111–120.

Fuchs, H., Poulton, J., Eyles, J., Greer, T., Goldfeather, J., Ellsworth, D., Molnar, S., Turk, G., Tebbs, B., and Israel, L. 1989. Pixel-Planes 5: A heterogeneous multiprocessor graphics system using processor-enhanced memories. In *Proceedings of SIGGRAPH*. 79–88.

Garachorloo, N., Sproull, R. F., Gupta, G., and Sutherland, I. 1989. A characterization of ten rasterization techniques. In *Proceedings of SIGGRAPH*. 355–368.

Glassner, A. 1978. *An Introduction to Ray-Tracing*. The Morgan Kauffman Series in Computer Graphics.

Greene, N. 1996. Hierarchical polygon tiling with coverage masks. In *Proceedings of SIGGRAPH*. 65–74.

Gu, X., Gortler, S., and Hoppe, H. 2002. Geometry images. In *Procedings of the SIGGRAPH*. 355–361.

Haeberli, P. 1990. Paint by numbers: Abstract image representations. In *Proceedings of SIGGRAPH*. 207–214.

Heckbert, P. 1989. Fundamentals of texture mapping and image warping. Master's thesis, Department of Electrical Engineering and Computer Science, University of California at Berkeley.

Kaufman, A. 1993. *Rendering, Visualization, and Rasterization Hardware*. Springer-Verlag, New York, NY.

Larson, G. 1998. The Holodeck: A parallel ray-caching system. In *Proceedings of Eurographics Workshop on Parallel Graphics and Visualization*.

LEVOY, M. AND WHITTED, T. 1985. The use of points as a display primitive. UNC Computer Science Tech. rep. TR85-022.

MARK, W., MCMILLAN, L., AND BISHOP, G. 1997. Post-rendering 3D warping. *Symposium on Interactive 3D Graphics*. 7–16.

MARK, W. 1999. Post-rendering 3D image warping: Visibility, reconstruction, and performance for depth-image warping. Ph.D. Thesis, University of North Carolina at Chapel Hill, NC.

MCALLISTER, D., NYLAND, L., POPESCU, V., LASTRA, A., AND MCCUE, C. 1999. Real-time rendering of real-world environments. In *Proceedings of Eurographics Workshop on Rendering*. 145–160.

MCCOOL, M., WALES, C., AND MOULE, K. 2001. Incremental and hierarchical hilbert order edge equation polygon rasterization. In *Proceedings of ACM/Eurographics Symposium on Graphics Hardware*.

MCCORMACK, J. AND MCNAMARA, R. 2000. Tiled polygon traversal using half-plane edge functions. In *Proceedings of the ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware*. 15–21.

MCMILLAN, L. AND BISHOP, G. 1995. Plenoptic modeling: An image-based rendering system. In *Proceedings of SIGGRAPH*. 39–46.

MCMILLAN, L. 1997. An image-based approach to three-dimensional computer graphics. Ph.D. Thesis, University of North Carolina at Chapel Hill, NC.

MITRA, T. AND CHIUEH, T. 1999. Dynamic 3D graphics workload characterization and the architectural implications. In *Proceedings of the 32nd Annual ACM/IEEE International Symposium on Microarchitecture*. 62–71.

MOLNAR, S., EYLES, J., AND POULTON, J. 1992. PixelFlow: High-speed rendering using image composition. In *Proceedings of SIGGRAPH*. 231–240.

MOLNAR, S. 1991. Efficient supersampling antialiasing for high-performance architectures. UNC-CS Tech. rep. TR91-023.

MONTRYM J., BAUM, D., DIGNAM, D., AND MIGDAL, C. 1997. InfiniteReality: A real-time graphics systems. In *Proceedings of SIGGRAPH*. 293–302.

NVIDIA. http://www.nvidia.com.

BURROWS ET AL. 2004. All about antialiasing. NVIDIA whitepaper. http://www.nvidia.com/object/all_about_aa.html.

NVIDIA. 2005A. ACCUVIEW TECHNOLOGY. NVIDIA tech. brief. http://www.nvidia.com/object/feature_accuview.html.

NVIDIA. 2005b. High-Resolution Antialiasing Through Multisampling. NVIDIA tech. brief. http://www.nvidia.com/object/feature_hraa. html.

OLANO, M. 1998. A programmable pipeline for graphics hardware. Ph.D. Thesis, Department of Computer Science, The University of North Carolina at Chapel Hill, NC.

OLANO, M. AND GREER, T. 1997. Triangle scan conversion using 2D homogeneous coordinates. In *Proceedings of ACM/Eurographics Symposium on Graphics Hardware*. 89–95.

OPENGL. Computer graphics API. http://www.opengl.org/.

OWENS, J. 2003. Computer graphics on a stream architecture. Ph.D. thesis, Computer Science Department, Stanford University.

PFISTER, H., ZWICKER, M., VAN BAAR, J., AND GROSS, M. 2000. Surfels: Surface elements as rendering primitives. In *Proceedings of SIGGRAPH*. 335–342.

PINEDA, J. 1988. A parallel algorithm for polygon rasterization. ACM Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques. vol. 22, 4, 17–20.

POPESCU, V., LASTRA, A., ALIAGA, D., AND OLIVEIRA, M. 1998. Efficient warping for architectural walkthroughs using layered depth images. In *Proceedings of IEEE Visualization*. 211–215.

POPESCU, V., EYLES, J., LASTRA, A., STEINHURST, J., ENGLAND, N., AND NYLAND, L. 2000. The WarpEngine: An architecture for the post-polygonal age. In *Proceedings of SIGGRAPH*. 433–442.

POPESCU, V. 2001. Forward Rasterization: A reconstruction algorithm for image-based rendering. Ph.D. thesis, Computer Science Department, University of North Carolina at Chapel Hill, NC.

RAFFERTY, M., ALIAGA, D., AND LASTRA, A. 1998. 3D image warping in architectural walkthroughs. In *Proceedings of VRAIS*. 228–233.

REN, L., PFISTER, H., AND ZWICKER, M. 2002. Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. In *Proceedings EUROGRAPHICS*, 461–470.

RUSINKIEWICZ, S. AND LEVOY, M. 2000. QSplat: A multiresolution point rendering system for large meshes. In *Proceedings of SIGGRAPH*, 343–352.

SCHILLING, A. 1991. A new simple and efficient antialiasing with subpixel masks. In *Proceedings of SIGGRAPH*, 133–141.

SHADE, J., GORTLER, S., HE, L., AND SZELISKI, R. 1998. Layered depth images. In *Proceedings of SIGGRAPH*, 231–242.

WESTOVER, L. 1990. Footprint evaluation for volume rendering. In *Proceedings of SIGGRAPH*, 367–376.

WHITTED, T. AND KAJIYA, J. 2005. Fully procedural graphics. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, 81–90.

WHITTED, T. AND WEIMER, D. 1982. A software testbed for the development of 3D raster graphics systems. *ACM Trans. Graph. 1*, 43–58.

ZWICKER, M., PFISTER, H., VAN BAAR, J., AND GROSS, M. 2002. EWA splatting. *IEEE Trans. Visualiz. Comput. Graph.* 223–238.

Received April 2005; revised October and November 2005; accepted December 2005