To appear in the Journal of Graphics Tools, Volume 16, Issue 1, 2012

Tightening the Precision of Perspective Rendering

Paul Upchurch and Mathieu Desbrun

Abstract

Precise depth calculation is of crucial importance in graphics rendering. Improving precision raises the quality of all downstream graphical techniques that rely on computed depth (e.g., depth buffers, soft and hard shadow maps, screen space ambient occlusion, and 3D stereo projection). In addition, the domain of correctly renderable scenes is expanded by allowing larger far-to-near plane ratios and smaller depth separation between mesh elements. Depth precision is an ongoing problem as visible artifacts continue to plague applications from interactive games to scientific visualizations despite advances in graphics hardware.

In this paper we present and analyze two methods that greatly impact visual quality by automatically improving the precision of depth values calculated in a standard perspective divide rendering system such as OpenGL or DirectX. The methods are easy to implement and compatible with 1/Z depth value calculations. The analysis can be applied to any depth projection based on the method of homogeneous coordinates.

1 Introduction

At each stage of the graphics pipeline, floating point precision and integer rounding errors accumulate. These small errors can have a large impact when they are fed into a nonlinear function such as a pass/fail test. In particular, depth buffer [Catmull 74; Straßer 74] errors exhibit a class of hidden surface removal (HSR) artifacts known as Z fighting due to either depth bucket discretization or depth inversion. Precision errors are the direct cause of depth inversion and exacerbate discretization. Similarly, shadow maps [Williams 78] exhibit light bleeding.

Depth artifacts can be eliminated by tuning the system parameters. However, an interactive system where meshes, lights and cameras are free to move in a large spatial environment is very labor intensive to tune. Systematically improving rendering precision for all possible scene configurations can either eliminate the need to perform tuning, or reduce the amount of work needed to reach an equivalent level of quality.

Precise rendering is especially important in applications involving visualizations of high dynamic range data. Examples include flythroughs for solar system missions, or even physically-based simulations [Harmon et al. 08] where great computational efforts to guarantee that objects do not interpenetrate to machine accuracy can be ruined by visual artifacts.

Previous Work. There are three stages of depth buffering that greatly impact HSR quality: the mapping of Z values to normalized depth, the calculation of fragment depth, and the choice of storage format for fragment depth values.

The first published depth mapping is the standard 1/Z depth, which is still the widest deployed mapping today. Lapidous and Jiao [Lapidous and Jiao 99] give a list of well-known alternative mappings and describe the complementary Z depth mapping. Their analysis only considers mapping and storage formats. Akeley and Su [Akeley and Su 06] address depth calculation with a simulated analysis but they specifically exclude arithmetic errors. Unsuprisingly, there has been no attention devoted to improving depth calculation arithmetic since it is merely addition and multiplication followed by a division, the so-called perspective divide.

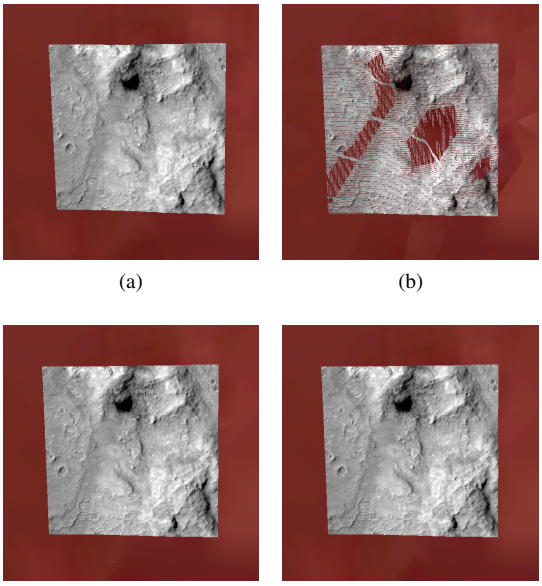

Figure 1: A terrain patch floating above the surface of Mars. An axis aligned camera can correctly distinguish the patch from the surface (a). However, rotating the camera slightly produces obvious HSR artifacts (b). These artifacts cannot be attributed to depth buffer resolution or software/hardware defects. They are the result of introducing inexact floating point numbers into the transform matrix. Rendering the same off-axis viewpoint with an infinite projection (c) or two-step transform (d) produces a markedly improved result. Dataset provided by NASA/JPL/University of Arizona.

A related topic is the infinite projection matrix, which is generally employed for its ability to remove the far clipping plane. Everitt and Kilgard [Everitt and Kilgard 03] describe how this facilitates stencil shadow volumes. Also, interactive games may use an infinite projection to ensure that drawn objects are always visible regardless of the dynamic position of the viewer. Infinite projections are used cautiously since they are assumed to negatively impact precision. This is due to the same number of depth storage bits being distributed across a larger Z range. However, our analysis will show, counterintuitively, that the infinite projection is a *more precise* general purpose projection and the finite projection is only useful when the depth range ratio is very small or extremely high precision is needed at the near plane.

Contributions. In this paper we analyze the floating point precision of the standard 1/Z depth calculation. Then, we describe two methods that drop into a standard graphics pipeline and produce depth values with the standard mapping from Z to buckets. One method does not require any extra computation, the other method guarantees improved precision. The methods may be implemented singly or jointly.

The first method is the infinite projection matrix. It is equivalent to a standard perspective projection matrix where the far plane has been

set to infinity. This causes an element of the matrix to be replaced by 1. By saving one rounding operation and saving one floating point multiply, relative error is reduced. This method has the greatest impact at the back of the depth buffer and does not affect performance.

The second method is to transform vertices by first transforming to view space and then multiplying the view coordinates by the projection matrix instead of transforming by the precomposed matrix. This improves precision by canceling out matching error terms when dividing by Z. There is no performance penalty for rendering engines that already generate view coordinates. Otherwise, an extra matrix multiply is needed in the vertex transform stage.

2 Floating Point Analysis Methodology

Throughout this paper exact real numbers are written $\bar{a}$ and have an associated inexact floating point representation, a. Add, subtract, multiply, and divide floating point operations are written $\oplus, \ominus, \otimes, \oslash$. Precision errors are measured in units of machine epsilon, ϵ.

We follow the method of Goldberg [Goldberg 91], where $\otimes, \oslash$ and rounding operations have a relative error of ϵ and $\oplus, \ominus$ operations with a guard digit have a relative error of 2ϵ. Therefore, we will write

$$\bar{x} = x\,(1+\delta_i), \quad x \oplus y = (x+y)(1+\gamma_i), \quad x \otimes y = xy\,(1+\delta_i),$$

where the errors δ_i and γ_i satisfy $|\delta_i| \le \epsilon$ and $|\gamma_i| \le 2\epsilon$.

3 Projection Matrix Precision

The perspective projection matrix is a sparse matrix that prepares the transformed vertex coordinates for perspective divide while simultaneously scaling the values to match the remainder of the graphics pipeline.

3.1 Standard Projection

For analysis we use a generalized projection matrix that is compatible with both OpenGL and DirectX; i.e., we will write

$$\begin{bmatrix} \bar{c} & & & \\ & \bar{d} & & \\ & & \bar{a} & -\bar{b} \\ & & 1 & \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \text{where} \quad \bar{a} = \frac{f+n}{f-n} \quad \bar{b} = \frac{2fn}{f-n}.$$

The calculated depth value is then

$$\begin{aligned} depth &= \big[(\bar{a} \otimes z) \ominus \bar{b}\,\big] \oslash z \\ &= \big[\big[a\,(1+\delta_1) \otimes z\big] \ominus b\,(1+\delta_2)\big] \oslash z \\ &= \big[az\,(1+\delta_1)(1+\delta_3) \ominus b\,(1+\delta_2)\big] \oslash z \\ &= \big[az\,(1+\delta_1)(1+\delta_3) - b\,(1+\delta_2)\big](1+\gamma_1) \oslash z \\ &= \big[a\,(1+\delta_1)(1+\delta_3) - b/z\,(1+\delta_2)\big](1+\gamma_1)(1+\delta_4) \\ &\approx a\,(1+5\epsilon) - b/z\,(1+4\epsilon). \end{aligned}$$

This expression characterizes the precision error of a single calculated depth value expressed as a relative error. Generally, we are more interested in the *difference* of depth values since that determines the pass/fail result of a depth test. The absolute error of a comparison is double the relative error, so we expect $10\epsilon\,a + 8\epsilon\,b/z$. However, floating point units are actually deterministic, so $\bar{a}$ will always round to the same value. Thus, error terms that depend on values that are invariant between depth fragments will cancel out.

If we note that δ_1 depends on a, δ_2 depends on b, δ_3 depends on a and z, and both δ_4 and γ_1 depend on a, b, and z, we can write

$$\begin{aligned} depth_i - depth_j &= a\,(1+\delta_1)(1+\delta_{3i})(1+\delta_{4i})(1+\gamma_{1i}) \\ &\quad - a\,(1+\delta_1)(1+\delta_{3j})(1+\delta_{4j})(1+\gamma_{1j}) + \cdots \\ &\approx 8\epsilon\,a\,(1+\delta_1) + \cdots. \end{aligned}$$

Therefore, 8ϵ is the contribution of the a term in terms of depth comparison precision.

3.2 Infinite Projection

Observe that as f/n grows, the value of $\bar{a}$ approaches 1. We can eliminate the floating point rounding error by modifying the projection equation to let $\bar{a}$ be exactly 1 and $\bar{b}$ be $2n$ since

$$\lim_{f\to\infty} \frac{f+n}{f-n} = 1 \qquad \lim_{f\to\infty} \frac{2fn}{f-n} = 2n.$$

Thus, the infinite projection matrix is written as

$$\begin{bmatrix} \bar{c} & & & \\ & \bar{d} & & \\ & & 1 & -\overline{2n} \\ & & 1 & \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Following the method of Section 3.1 we have

$$\begin{aligned} depth &= \big[z \ominus \overline{2n}\,\big] \oslash z = \big[\big[z - 2n\,(1+\delta_1)\big](1+\gamma_1)\big] \oslash z \\ &= \big[1 - 2n/z\,(1+\delta_1)\big](1+\gamma_1)(1+\delta_2) \\ &\approx 1\,(1+3\epsilon) - 2n/z\,(1+4\epsilon). \end{aligned}$$

Since, in this case, δ_1 depends only on n, while δ_2 and γ_1 depend on n and z, we get

$$\begin{aligned} depth_i - depth_j &= (1+\delta_{2i})(1+\gamma_{1i}) - (1+\delta_{2j})(1+\gamma_{1j}) + \cdots \\ &\approx 6\epsilon + \cdots. \end{aligned}$$

Infinite Projection versus Standard Projection. The difference between the two methods above is 2ϵ relative error in depth, or 2ϵ absolute error in depth comparison. Since depth is in the range of $[-1, 1]$ a relative (or absolute) error of 2ϵ is equivalent to a relative error of at least 2ϵ in depth.

While the infinite projection method has 2ϵ better precision and covers view space from the near plane to infinity, one might argue that, if there is no interest in rendering objects beyond a far clipping plane, the smaller depth buckets of a standard projection may be better. It is, however, easy to see that the precision of an infinite projection is better over a larger viewing range than the precision of a finite projection with reasonably large values of f/n.

Small depth values benefit more from smaller buckets since integral rounding errors dominate relative error for values near zero. We compute how small the depth must be for this to occur and derive equations to describe this in terms of a view distance, z_λ, and a tolerance, Δ_λ. Letting $r = f/n$,

$$z_\lambda = \frac{2f}{2r - 1 - \sqrt{r}} \qquad \Delta_\lambda = \frac{z_\lambda}{\frac{1}{2}\frac{b}{z_\lambda}2^\beta - 1}.$$

Precision loss is limited to the view volume between n and z_λ. The loss is at most Δ_λ in simulation units. For example, if the near plane is 1 meter and the far plane is 101 meters, the loss is at most 66 nanometers for objects within 5 centimeters of the near clip plane. The remaining 99.95% of the Z range will exhibit improved precision. We provide an in-depth analysis in Appendix A.

4 Vertex Transform Precision

The usual vertex transform is $P \cdot M \cdot \vec{v}$, where P is the projection matrix and M is the transform matrix (commonly called the ModelView matrix). It has been standard practice to precompose the projection and transform matrices before entering the per-vertex transform stage of the graphics pipeline. The main advantage being that this approach saves a few multiply/add operations per vertex. We call the non-precomposed method the *two-step* transform.

4.1 Two-Step Transform

This transform is now evaluated as $P \cdot (M \cdot \vec{v})$: first vertices are transformed to view space by the transform matrix, then the view coordinates are multiplied by the projection matrix. (For brevity, we omit floating point rounding operations.)

$$\begin{bmatrix} c & & & \\ & d & & \\ & & a & -b \\ & & 1 & \end{bmatrix} \left(\begin{bmatrix} m_{11} & m_{12} & m_{13} & x_0 \\ m_{21} & m_{22} & m_{23} & y_0 \\ m_{31} & m_{32} & m_{33} & z_0 \\ & & & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \right)$$
$$= \begin{bmatrix} c & & & \\ & d & & \\ & & a & -b \\ & & 1 & \end{bmatrix} \begin{bmatrix} m_{11}x + m_{12}y + m_{13}z + x_0 \\ m_{21}x + m_{22}y + m_{23}z + y_0 \\ m_{31}x + m_{32}y + m_{33}z + z_0 \\ 1 \end{bmatrix}.$$

Let $\chi = (m_{31} \otimes x) \oplus (m_{32} \otimes y) \oplus (m_{33} \otimes z) \oplus z_0$; we then have

$$\begin{aligned} \mathit{depth} &= \big[(\chi \otimes a) \ominus b\big] \oslash \chi = \big[\chi a\,(1+\delta_1) - b\big](1+\gamma_1) \oslash \chi \\ &= \big[a\,(1+\delta_1) - b/\chi\big](1+\gamma_1)(1+\delta_2). \end{aligned}$$

Note that the perspective divide is nearly exact for the a term. The same can not be said for the precomposed transform.

4.2 Precomposed Transform

Before entering the pipeline, the transform and projection matrix can be precomposed instead. Each vertex is thus transformed by a single matrix multiply as $(P \cdot M) \cdot \vec{v}$, yielding

$$\left(\begin{bmatrix} c & & & \\ & d & & \\ & & a & -b \\ & & 1 & \end{bmatrix} \begin{bmatrix} m_{11} & m_{12} & m_{13} & x_0 \\ m_{21} & m_{22} & m_{23} & y_0 \\ m_{31} & m_{32} & m_{33} & z_0 \\ & & & 1 \end{bmatrix} \right) \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$
$$= \begin{bmatrix} m_{11}c & m_{12}c & m_{13}c & x_0c \\ m_{21}d & m_{22}d & m_{23}d & y_0d \\ m_{31}a & m_{32}a & m_{33}a & z_0a - b \\ m_{31} & m_{32} & m_{33} & z_0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

The depth for this precomposed transform now becomes

$$\begin{aligned} \mathit{depth} = \big[\big[(m_{31} \otimes a) \otimes x\big] &\oplus \big[(m_{32} \otimes a) \otimes y\big] \\ &\oplus \big[(m_{33} \otimes a) \otimes z\big] \oplus \big[(z_0 \otimes a) \ominus b\big]\big] \\ &\oslash \big[(m_{31} \otimes x) \oplus (m_{32} \otimes y) \oplus (m_{33} \otimes z) \oplus z_0\big] \\ = \big[m_{31}a(1+\delta_1)x(1+\delta_2) &\oplus m_{32}a(1+\delta_3)y(1+\delta_4) \\ &\oplus m_{33}a(1+\delta_5)z(1+\delta_6) \oplus \big[z_0a(1+\delta_7) \ominus b\big]\big] \\ &\oslash \big[m_{31}x(1+\delta_8) \oplus m_{32}y(1+\delta_9) \oplus m_{33}z(1+\delta_{10}) \oplus z_0\big]. \end{aligned}$$

At this point it is clear that we will not get a clean divide as in the two-step transform case. Indeed, consider the first terms of the numerator and denominator,

$$\begin{aligned} \mathit{depth} &= \big[m_{31}a(1+\delta_1)x(1+\delta_2) \oplus \cdots\big] \oslash \big[m_{31}x(1+\delta_8) \oplus \cdots\big] \\ &= \big[m_{31}xa(1+\delta_1+\delta_2+\mathcal{O}(\epsilon^2)) \oplus \cdots\big] \oslash \big[m_{31}x(1+\delta_8) \oplus \cdots\big] \\ &\approx \big[\big[m_{31}x(1+\delta_8) + m_{31}x(\delta_1+\delta_2-\delta_8)\big]a \oplus \cdots\big] \\ &\quad \oslash \big[m_{31}x(1+\delta_8) \oplus \cdots\big]. \end{aligned}$$

Arranging terms so that both the numerator and denominator have the expression $m_{31}x\,(1+\delta_8)$ reveals an error term of $m_{31}x\,\mathcal{O}(\epsilon)$. Continuing in this fashion, the final result is of the form

$$\begin{aligned} &a\big[1+\mathcal{O}(\epsilon)\big] - b/\chi\big[1+\mathcal{O}(\epsilon)\big] \\ &\quad + a/\chi\big[m_{31}x\,\mathcal{O}(\epsilon) + m_{32}y\,\mathcal{O}(\epsilon) + m_{33}z\,\mathcal{O}(\epsilon) + z_0\,\mathcal{O}(\epsilon)\big]. \end{aligned}$$

The last term of this expression represents the arithmetic error introduced by the precomposed transform. Furthermore, the error will not cancel out during a depth comparison since it is dependent on x, y and z.

Our analysis clearly shows that the two-step transform will always give better results than the precomposed transform.

5 Implementation and Results

5.1 Implementation

An advantage of the methods presented in this paper is that they do not require extensive software development. The infinite projection method is implemented by substituting the values $1, 2n$ for a, b in the standard perspective projection matrix. In addition, if depth value clipping is desired, the depth buffer clear value must be adjusted. In total, three constant values are modified. The two-step transform method requires making changes at the vertex transform stage of the graphics pipeline. Output vertex values are computed by first multiplying the input vertex by the transform matrix, followed by a second multiply by the projection matrix. The total is two lines of code, provided that the graphics engine has a programmable vertex stage.

The specific values of a and b are implementation dependent. In Listing 1 we apply the infinite projection method to an OpenGL system that is compatible with glFrustum [Segal and Akeley 06] by changing two elements of the matrix. Likewise, for a DirectX system compatible with D3DXMatrixPerspectiveLH, which has $f/(f-n)$ and $-nf/(f-n)$ as the 10th and 14th elements of the matrix, we would set those elements to 1 and $-n$ and set the remaining elements according to the function documentation [Microsoft Corp. 11].

We give a basic implementation of the two-step transform in Listing 2. Several optimizations are possible. Since the projection matrix is typically sparse, floating point operations can be saved and bandwidth can be reduced with value packing. Even more optimization is possible in conjunction with an infinite projection where $n = 1/2$. In addition, the intermediate value, VertexViewPos, which is the vertex position in the frame of the camera relative to the camera origin, may be useful for unrelated graphical techniques.

5.2 Results on a Synthetic Example

Figure 2 shows the results of applying each method singly to a test scene. The test scene is a green plane under a red mesh. The left half of the red mesh is parallel to the green plane. The right half of the red mesh slopes downward until it touches the green plane at the rightmost edge. Correct HSR would display a red square inside

Listing 1: An implementation of infinite projection in an OpenGL system.

```
if(is_infinite_proj) {
  float a,b,c,d,i,j,k,mx[16];
  a=(-1.0f);      /* lim f->inf of (-(f+n)/(f-n)) */
  b=(-(2.0f*n)); /* lim f->inf of (-(2.0f*f*n)/(f-n)) */
  c=(2.0f*n)/(x2-x1);
  d=(2.0f*n)/(y2-y1);
  i=(x2+x1)/(x2-x1);
  j=(y2+y1)/(y2-y1);
  k=(-1.0f);
  mx[0]=c; mx[4]=0; mx[ 8]=i; mx[12]=0;
  mx[1]=0; mx[5]=d; mx[ 9]=j; mx[13]=0;
  mx[2]=0; mx[6]=0; mx[10]=a; mx[14]=b;
  mx[3]=0; mx[7]=0; mx[11]=k; mx[15]=0;
  glMultMatrixf(mx);
}
else glFrustum(x1,x2,y1,y2,n,f);

glDepthRange(0,1);
if(is_infinite_proj && is_depth_clip) {
  glClearDepth(1.0-n/f);
}
else glClearDepth(1.0);
```

Listing 2: An implementation of two-step transform in a GLSL vertex shader.

```
if(is_two_step) {
  vec4 VertexViewPos=gl_ModelViewMatrix*gl_Vertex;
  gl_Position=gl_ProjectionMatrix*VertexViewPos;
}
else gl_Position=gl_ModelViewProjectionMatrix*gl_Vertex;
```

a green square. We expect HSR artifacts to manifest most strongly at the right edge of the red mesh where separation approaches zero.

To recreate the test results it is necesary to store real numbers in the 3x3 upper-left corner of the transform matrix. A default camera initialized with an identity matrix will exhibit greatly reduced precision error. Also, the red plane must be rendered before the green plane so that the presence of green pixels is due to depth inversion (caused by a green fragment having a calculated depth less than a red fragment), not discretization (caused by depth values falling into the same bucket). The test scene was designed to specifically show HSR artifacts and they will always appear on the right-hand side of the red plane, although a narrow field of view may be needed depending on the test environment.

In every screenshot pictured here, the precision errors are completely eliminated on the left-hand side of the red plane. This is because the precision improvement was enough to lower the depth value error below the parallel separation distance. On the right-hand side the errors remain, but typically displaced and with varying intensity relative to the errors shown in the first column. On average the errors will be significantly less, but any two screenshots will vary considerably.

5.3 Results on Real Examples

Figure 3 shows the two-step transform implemented in the Dspace software at the Dynamics And Real-Time Simulation (DARTS) Lab at JPL. The image is of a terrain patch positioned above the Lunar surface. The far clipping plane was adjusted to intersect the terrain patch. The two-step transform method displays a clean, high precision clipped edge. The test environment was a Linux/OpenGL/O-

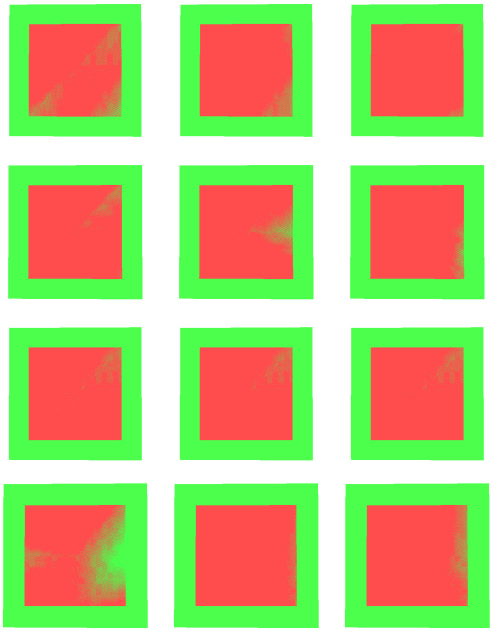

Figure 2: Standard projection (left column), infinite projection (middle column) and two-step transform (right column). Each row represents a comparison of the three methods. Four rows are provided because results vary considerably even when camera angle and/or position changes only slightly.

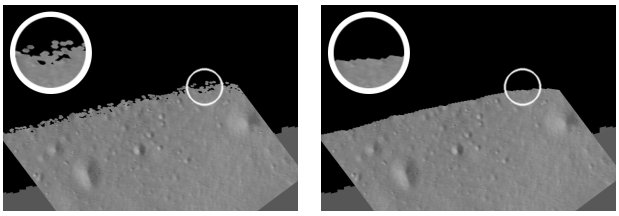

Figure 3: Precomposed transform exhibits a ragged edge where a clean depth clip is expected (left). The two-step transform gives a much better approximation of a clean clip (right). Dataset provided by NASA.

gre3D PC with NVIDIA Quadro hardware.

Figure 1 shows both methods implemented in a high dynamic range terrain engine developed at Caltech for the Visualization Technology Applications & Development group at JPL. The image is of a terrain patch positioned at least 600 meters above the surface of Mars. The camera was located 10 km above the surface. Depth bucket resolution at the terrain patch was less than 7 meters and input vertex coordinate error was less than 1 meter. Despite the low error relative to the 600 meter separation, HSR artifacts are prominent. Both methods, singly or jointly, removed all artifacts. In addition, the underlying surface of two overlapping patches also exhibited HSR artifacts. Improved HSR quality was observed here as well. The test environment was a Windows XP/OpenGL PC with NVIDIA GeForce hardware.

Figure 4 demonstrates an improvement to a non-interactive system. Blender is an open source 3D modeling software package that includes a rendering tool for producing high quality stills and movies. The renderer relies on a depth buffer and thus can benefit from our methods. We downloaded the source code to version 2.49b and

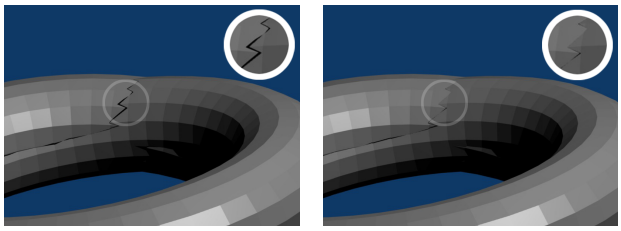

Figure 4: A quality render of two intesecting torii by Blender 2.49b (left). This non-interactive render is improved by implementing an infinite projection (right). The change only requires adding two lines of code.

modified the depth calculation in arithb.c to implement an infinite projection. The two-step transform was not implemented since the software renderer does not have a programmable vertex stage. The test scene of two interpenetrating torii exhibit obvious cracks where the surfaces intersect. With infinite projection the cracks are reduced even though the depth buckets are larger. This clearly shows the arithmetic benefits of infinite projection.

6 Discussion and Conclusion

The methods given in this paper are direct improvements on the standard 1/Z depth mapping with supporting analysis. However, the designer of a graphics pipeline may decide to use an alternative depth mapping and/or depth value storage format. Even so, most of our results are still relevant.

We can generalize the analysis of Section 3 by noting that the key results depend on whether or not the a value has an exact floating point representation, and not on the specific values of a and b. Therefore, we can extend the analysis to any depth mapping. Complementary Z, having $a = n/(f-n)$ and $b = fn/(f-n)$, will suffer the same arithmetic precision loss of 2ϵ as 1/Z. Whereas, inverse W, having $a = 0$ and $b = n$, will enjoy the same high precision as infinite projection.

Likewise, the results of Section 4 are only dependent on the presence of a non-zero a value. Therefore, the two-step method would still be effective in a complementary Z system, but should not have any effect in an inverse W system. The special case of $a = 1$ is noteworthy because it reduces the error term to $2\epsilon\, z_0/\chi$.

It should also be noted that, regardless of the depth mapping, any system can improve precision by carefully choosing values of n and f such that a becomes exact. Our analysis shows exactly why this trick improves precision. However, in the case of 1/Z, this trick is clearly inferior to our methods that do not constrain n and f.

To summarize, the designer of a perspective divide transform matrix should strive to set $a = 0$ as this value maximizes the arithmetic precision of the perspective divide. Failing that, $a = 1$ is preferable to $a = \bar{a}$. An a value without an exact floating point representation should be avoided whenever possible.

Conclusion. In this paper we describe two different methods of improving depth precision with little or no cost. The methods are easy to implement and should be applicable to all modern rasterization engines.

We have restricted our attention to the 1/Z depth calculation; however the same approach may be applied to other depth methods to tighten the bounds on precision. Furthermore, other stages of the graphics pipeline may contain hidden opportunities for optimizing floating point precision, yielding further improvements to rendering quality.

Acknowledgments. We would like to thank Stephen Kulczycki, Kevin Hussey, Patrick Mullen, Marc Pomerantz and Steven Myint for their support. MD wishes to acknowledge the partial support of the NSF (CCF-1011944, CCF-0811373, and CMMI-0757106 grants), Disney & Pixar Animation Studios. This work was funded by JPL's Office of Communications and Education.

A Infinite Projection Depth Bucket Analysis

A depth value is stored with integer rounding, which introduces an absolute error g. The error is related to the number of depth buffer bits, β.

$$\widehat{d} = d + g \;:\; |g| \le \frac{1}{2}\,\frac{1}{2^{\beta}}.$$

This absolute error can be expressed as a relative error that varies depending on the value of d. The relative error is not well defined as d approaches zero.

$$d + g = d\,(1+\delta) \qquad\qquad \delta = \frac{g}{d} \;:\; d > 0.$$

Consider the difference between standard depth, d, and infinite depth, d'.

$$d = \frac{1}{2}\Big(a - \frac{b}{z} + 1\Big) \qquad\qquad d' = \frac{1}{2}\Big(1 - \frac{2n}{z} + 1\Big).$$

Express infinite depth as a function of depth through

$$\begin{aligned} d' &= \frac{1}{2}\Big(\frac{f-n}{f-n} - \frac{2n(f-n)}{f-n}\,\frac{1}{z} + 1\Big) \\ &= d + \frac{1}{2}\Big(\frac{-2n}{f-n} + \frac{2n^2}{f-n}\,\frac{1}{z}\Big) \\ &= d - \Big(\frac{n}{f-n}\Big)\Big(\frac{z-n}{z}\Big) \\ &= d - \theta\,\phi \;:\; 0 < \theta,\; 0 < \phi < 1 \;\text{ for }\; z > n. \end{aligned}$$

Both θ and ϕ are positive, so d' is always less than d. If the increase in relative error is less than ϵ, then the larger buckets do not dominate the precision. Specifically, we want

$$\frac{g}{d'} - \frac{g}{d} < k\epsilon \;:\; k \ge 1.$$

For a 24-bit depth buffer and single precision floating point, we have $\beta = 24$, $\epsilon = 2^{-24}$, thus $\epsilon = 2g$. Therefore, the domain in which $(d-d')/dd' < 2k$ is satisfied for all values of d, d' in some range for given f, n, k is the domain in which $k\epsilon$ is larger than the change in depth bucket size.

> If $d \ge \frac{1}{2}$ and $d' \ge \frac{1}{2}$ then $\frac{d-d'}{dd'} < 2k$ is true for all $k \ge 1$, provided that $f > 2n$.

Indeed, d' is constrained to be less than d, but greater than $1/2$. So, the numerator is maximized by $d = 1$ and $d' = 1/2$, while the denominator is minimized by $d = d' = 1/2$. Note that either way d' is set to the lower bound of $1/2$. The expression $(1 - 1/2)/(1/2)^2 < 2k$ is true for all $k \ge 1$. Consequently, we see that the infinite projection matrix will always improve precision for depth values in the range $[0.5, 1]$.

> Moreover, if $\frac{f}{n} = r$ then d must satisfy $\frac{\frac{1}{r-1}}{d(d-\frac{1}{r-1})} < 2k$.

Following the same strategy of maximizing the numerator and minimizing the denominator, we will set d' equal to the lower bound. Therefore, consider the general case where $d - h < d' < d$ and the expression to satisfy is $(d - (d-h))/(d(d-h)) < 2k$. Note that $1/(r-1) = \theta$ and recall that $d' = d - \theta\,\phi$. We have $d' = d - \phi/(r-1)$. Since ϕ is less than 1, then $d' \ge d - 1/(r-1)$. Thus, let $h = 1/(r-1)$ and $k\epsilon = 2\epsilon$. Solving for the positive root of the quadratic $h/(d^2 - hd) = 4$ gives us a $\lambda < 0.5$ such that depth precision is improved in the range $[\lambda, 1]$.

$$\lambda = \frac{1}{2}\,\frac{n+\sqrt{fn}}{f-n}.$$

To translate this into something meaningful, we need z and bucket size, Δ, expressed in terms of λ:

$$z_\lambda = \frac{b}{a+1-2\lambda}, \qquad \frac{1}{2}\Big(a - \frac{b}{z_\lambda}+1\Big) - \frac{1}{2}\Big(a - \frac{b}{z_\lambda+\Delta_\lambda}+1\Big) = \frac{1}{2^\beta}.$$

From these equations we derive the equations given in Section 3.2.

References

Kurt Akeley and Jonathan Su. "Minimum triangle separation for correct z-buffer occlusion." In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*, pp. 27–30, 2006. Available online (`http://portal.acm.org/citation.cfm?id=1283900.1283904`).

Edwin Catmull. "A subdivision algorithm for computer display of curved surfaces." Ph.D. thesis, The University of Utah, 1974.

Cass W. Everitt and Mark J. Kilgard. "Practical and robust stenciled shadow volumes for hardware-accelerated rendering.", 2003. Available online (`http://arxiv.org/abs/cs.GR/0301002`).

David Goldberg. "What every computer scientist should know about floating-point arithmetic." *ACM Comput. Surv.* 23 (1991), 5–48. Available online (`http://doi.acm.org/10.1145/103162.103163`).

David Harmon, Etienne Vouga, Rasmus Tamstorf, and Eitan Grinspun. "Robust treatment of simultaneous collisions." *ACM Trans. Graph.* 27 (2008), 23:1–23:4. Available online (`http://doi.acm.org/10.1145/1360612.1360622`).

Eugene Lapidous and Guofang Jiao. "Optimal depth buffer for low-cost graphics hardware." In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, pp. 67–73, 1999. Available online (`http://doi.acm.org/10.1145/311534.311579`).

Microsoft Corp. "D3DX Reference.", 2011. Available online (`http://msdn.microsoft.com/en-us/library/windows/desktop/bb172965(v=vs.85).aspx`).

Mark Segal and Kurt Akeley. "The OpenGL Graphics System: A Specification (Version 2.1).", 2006. Available online (`http://www.opengl.org/registry/doc/glspec21.20061201.pdf`).

W. Straßer. "Schnelle Kurven- und Flächendarstellung auf graphischen Sichtgeräten." Ph.D. thesis, Technische Universität Berlin, 1974.

Lance Williams. "Casting curved shadows on curved surfaces." *SIGGRAPH Comput. Graph.* 12 (1978), 270–274. Available online (`http://doi.acm.org/10.1145/965139.807402`).

Web Information:

Paul Upchurch, Caltech, MC 305-16, 1200 East California Boulevard, Pasadena, CA 91125
(paulup@gmail.com)

Mathieu Desbrun, Caltech, MC 305-16, 1200 East California Boulevard, Pasadena, CA 91125
(mathieu@cms.caltech.edu)