Comparison of Depth Buffer Techniques for Large and Detailed 3D Scenes

Jonas Gilg*, Sebastian Zander†, Simon Schneegans*, Volker Ahlers†, Andreas Gerndt*⋆

* German Aerospace Center
Institute for Software Technology
Lilienthalplatz 7, 38108 Braunschweig
{jonas.gilg, simon.schneegans}@dlr.de

⋆ University of Bremen
Faculty III - Mathematics and Computer Science
Bibliothekstraße 1, 28359 Bremen
gerndt@uni-bremen.de

† University of Applied Sciences and Arts, Hannover
Faculty IV - Department of Computer Science
Ricklinger Stadtweg 120, 30459 Hannover
sebastian.zander@stud.hs-hannover.de
volker.ahlers@hs-hannover.de

Abstract: Large scale 3D scenes in applications like space simulations are often subject to depth buffer related issues and visual artefacts like Z-fighting and spatial jittering. These issues are primarily a result of indistinguishable depth buffer values. To mitigate these issues, many techniques have been developed over time to better distribute depth values over the clipping range. These techniques range from simple adjustments of the projection matrix to complex solutions like multistage rendering with layered depth buffers. This work presents, compares and evaluates commonly used approaches found in literature and real world applications. An experiment is set up to compare the presented depth buffer techniques using the metric of minimum triangle separation (MTS). The gathered results are presented and evaluated, to give a good overview on which techniques are well suited for the use in applications with large scale 3D scenes.

Keywords: depth buffer, depth mapping, projection matrices, Z-fighting, spatial jittering

1 Introduction

The purpose of this work is to examine the issues of Z-fighting and spatial jittering that arise when rendering large 3D scenes with extensive depth, as well as to discuss different techniques that aim to solve those issues. These issues manifest themselves as visual artefacts, like the flickering of certain pixels. In the worst case, incorrect spatial occlusion in three-dimensional space happens where objects that are behind other objects are drawn in front of them.

The root causes behind Z-fighting and spatial jittering are the limitations of depth buffer precision. Faces with only small distances between them get indistinguishable, or even wrong depth values for some of their fragments. This leads to wrong fragments being drawn on the screen.

The described issues are mainly due to technical limitations in computer hardware. In computer and video games, an area of computer graphics that is at the forefront of realistic and ever-growing virtual worlds, it is required to adapt to technical limitations using appropriate techniques to eliminate artefacts [CR11, Kem12]. However, such approaches almost never come without a drawback, so that involved graphics applications have to choose between either in-memory performance and processing speed or extended accuracy.

This work is going to present and compare different approaches that aim to solve such depth related problems for scenes where large and small scales are present at the same time. Such scenes could be solar system simulations, which have to display objects from small satellites to large planetary systems, or microbiology where individual atoms are visualized to make up large molecular structures.

In section 2 an overview is given over common techniques from science and real world applications. Section 3 presents an experimental setup for comparing the techniques previously introduced. An evaluation of the experimental results is given in section 4 and section 5 gives a brief summary and recommends a fitting solution to the stated problem.

2 Common Techniques

In the following sections, we present a number of solutions to the depth precision problem that have been found both in scientific and entertainment applications.

2.1 Standard Projection

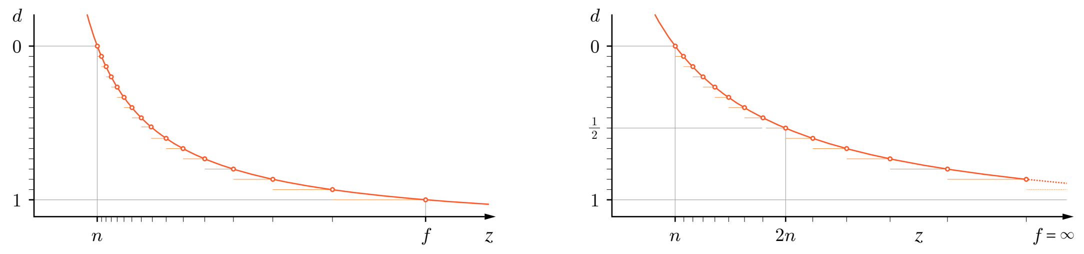
Standard projection is the way of projecting coordinates onto the screen and transforming their corresponding spatial distances through intermediate spaces into the range that a depth buffer can store. Equation 1 shows a standard projection matrix [Len11],

$$\mathbf{P}_{\text{std}} = \begin{bmatrix} s_x & 0 & t_x & 0 \\ 0 & s_y & t_y & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{1}$$

with s_x and s_y as the scale components, t_x and t_y as the translation components in screen space. While all projections define a range of distances that objects need to be in to be drawn, standard projection defines distinct non-trivial values for both the lower and upper limit. Both these limits are defined by near clipping plane n and far clipping plane f respectively.

Figure 1a graphs spatial distance z to its corresponding depth value d, so that distances n and f are mapped to their corresponding limits in the depth range. The non-linear shape of the curve is due to the perspective division that scales d proportional to the reciprocal of z. The graph illustrates that the precision of depth values is unevenly distributed over the entire clipping range and favours the near clipping plane.

The precision of depth through standard projection can be optimized by carefully and efficiently choosing near and far clipping planes. Most importantly for the distribution of precision, however, is the ratio between the two. Choosing a static but optimal clipping



(a) (b)

Figure 1: (a) The distance-depth mapping of a standard projection. (b) The distance-depth mapping of an infinite projection. Both use a hypothetical 4-bit integer buffer. Own illustrations in the style of [Ree15].

range can seldom solve the problem of Z-fighting alone. Dynamically adapting the clipping ranges might be a solution to this problem.

The depth buffer format that is used with standard projection generally only determines the overall depth precision. Integer formats cause depth values to be quantized to discreet levels with constant step heights. Floating-point formats allot precision differently with variable step heights due to their intrinsic quasi-logarithmic value distribution with higher precision for smaller numbers. Standard projections already put more depth precision at the near clipping plane, where depth buffer values are small. Employing a floating-point depth buffer would only intensify the imbalance in precision.

2.2 Infinite Projection

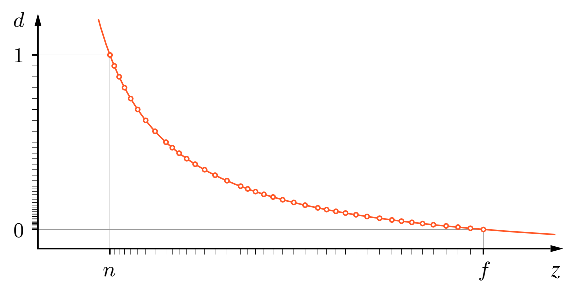
Infinite projection aims to project anything in front of the viewer independent of the distance. Contrary to standard projection, infinite projection thus does not make scaling factor s_z and offset o_z dependent on a far clipping plane. Equation 2 demonstrates an infinite projection matrix and how it is derived as the limit of a standard projection matrix as f approaches infinity [Len11].

$$\mathbf{P}_{\text{inf}} = \lim_{f\to\infty} \begin{bmatrix} s_x & 0 & t_x & 0 \\ 0 & s_y & t_y & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} s_x & 0 & t_x & 0 \\ 0 & s_y & t_y & 0 \\ 0 & 0 & -1 & -2n \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{2}$$

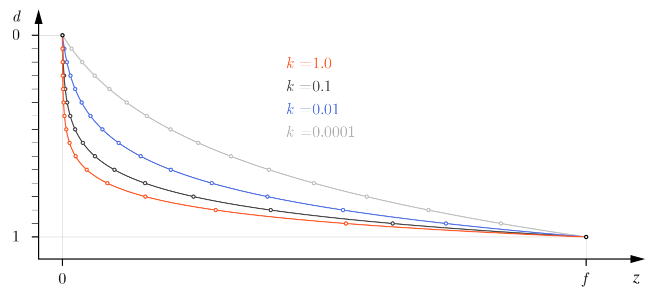
The resulting simplifications in the matrix reduce floating-point rounding errors and thus increase the depth precision [UD12]. While infinite projection allows potentially more to be seen in the distance, the precision of its produced depth values becomes less much sooner as compared to standard projection. Figure 1b demonstrates this.

2.3 Reversed Projection

The intention behind reversed projection (also known as complementary Z-buffer) is to balance the depth precision over the entire clipping range by reversing the depth range, mapping the near clipping plane to zero and the far clipping plane to one [LJ99]. The reversing alone does nothing more than moving depth precision from the near end to the far end. Using



(a)



(b)

Figure 2: (a) The distance-depth mapping of a reversed projection, using a 7-bit floating-point buffer. Notice the flipped direction of the ordinate. (b) Four logarithmic distance-depth mappings with different constants k, using a 4-bit integer buffer. Own illustrations in the style of [Ree15].

it with integer depth buffers would not be suited for reducing Z-fighting, because almost no precision is left at the near clipping plane. This is why reversed projection specifically requires floating-point depth buffers. Figure 2a demonstrates this. Compared to previous graphs, the ordinate is flipped, so that depth values increase bottom-up. The tick marks on the ordinate point out the beneficial property of floating-point encoding that leads to decimal numbers close to zero having the highest precision.

The higher precision near zero counterbalances the lower resolution at the far clipping plane [LJ99, Ree15]. Although the spacing between depth values along the curve is not perfectly constant, it is much more even than before. The corresponding projection matrix in equation 3 is derived from the standard projection matrix by simply reversing the signs of both scaling factor s_z and offset o_z.

$$\mathbf{P}_{\text{rev}} = \begin{bmatrix} s_x & 0 & t_x & 0 \\ 0 & s_y & t_y & 0 \\ 0 & 0 & \frac{f+n}{f-n} & \frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{3}$$

For reversed projection to work however, the graphics API has to be instructed that the depth comparison logic now has to be negated.

2.4 Reversed Infinite Projection

Another common approach is to combine the previously introduced infinite projection with a reversed projection [Ree15]. Just like before, a reversed infinite projection matrix can be derived as the limit of a normal reversed projection matrix as f approaches infinity [PS12, Kem12].

$$\mathbf{P}_{\text{rev inf}} = \lim_{f\to\infty} \begin{bmatrix} s_x & 0 & t_x & 0 \\ 0 & s_y & t_y & 0 \\ 0 & 0 & \frac{f+n}{f-n} & \frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} s_x & 0 & t_x & 0 \\ 0 & s_y & t_y & 0 \\ 0 & 0 & 1 & 2n \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{4}$$

This projection does not clip anything in the distance, making viewing distance infinite. If objects are at infinity, that is, they are at or beyond the distance at which their depth value is always zero, they become indistinguishable in the depth test. To still obtain plausible occlusion for objects at such distances, it requires them to be drawn from back to front.

2.5 Logarithmic Depth Mapping

Logarithmic depth mapping is one of the approaches that require a little more than what projection transformations can supply. Scaling depth values differently than the graphics pipeline does internally demand changes in the shader program. Two variants exist that differ in speed and visual accuracy. Both variants allow to balance the distribution of depth precision through a given logarithmic scaling constant k. Figure 2b illustrates four different logarithmic scales.

The *fast variant* can be integrated partially into the projection matrix and partially in the vertex shader [CR11]. At its core, this variant aims to alter the z coordinate of a vertex right after the projection transformation, as described by equation 5 [CR11, Kem09, Kem12].

$$z_{\text{clip}} = \begin{cases} \left(\frac{2\ln(kw_{\text{clip}}+1)}{\ln(kf+1)} - 1\right)w_{\text{clip}}, & \text{if } z_{\text{ndc}} \in [-1,1] \text{ (e.g., OpenGL)} \\ \frac{\ln(kw_{\text{clip}}+1)}{\ln(kf+1)} w_{\text{clip}}, & \text{if } z_{\text{ndc}} \in [\ \ 0,1] \text{ (e.g., Direct3D)} \end{cases} \tag{5}$$

As $\ln(kf+1)$ is only dependent on constants k and f, this part of the equation can be extracted and put into a standard projection matrix [CR11]. Equation 6 shows how such a projection matrix may look before it is numerically evaluated.

$$\mathbf{P}_{\text{log}} = \begin{bmatrix} s_x & 0 & t_x & 0 \\ 0 & s_y & t_y & 0 \\ 0 & 0 & 0 & \frac{1}{\ln(kf+1)} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{6}$$

After the multiplication with the projection matrix in the vertex shader, the result of $1/\ln(kf+1)$ can be accessed as z_{view} which then has to be multiplied with the rest of equation 5 to get z_{clip} [CR11].

After the fixed part of the graphics pipeline ran and the perspective division was executed, the depth value d is now correctly determined for each vertex position. The graphics pipeline expects d to be in a linear space, where it can be linearly interpolated in perspective correct fashion. This is not the case any more, unfortunately. Depending on the size and orientation of primitives like triangles on the screen, very distinct and unpleasant distortions of incorrectly interpolated vertex attributes may emerge.

A small loss in precision is caused by floating-point rounding errors, due to the preemptive and intended multiplication with w_{clip} and the subsequent imperative perspective division by w_{clip}. These are two arithmetic operations that algebraically cancel each other out, but that cannot be reduced or omitted due to the nature of the fixed part of the graphics pipeline.

The visually more *accurate variant* places the calculation of depth into the fragment shader. This overcomes incorrect interpolation for depth values as depth is calculated per fragment.

Numerous invocations of the logarithmic function makes this variant more accurate. This is also one of the reasons that makes this variant slower.

Depth value d can be calculated with $d = z_{\text{clip}}/w_{\text{clip}}$ [Kem09, Kem12], which would be a slightly simplified version of equation 6.

Manually writing to the depth buffer imposes a performance reduction, because when writing to this variable the graphics pipeline cannot perform the early depth test, which would allow skipping fragment shader invocations [Gro]. Writing to the depth buffer causes other optimizations and techniques, like depth buffer compression, to not work as well.

2.6 Linear Depth Mapping

Linear depth mapping builds upon the method used in the accurate logarithmic approach, as it is implemented through the fragment shader as well. For that reason, it implies the exact same restrictions on performance. This approach can be used when depth precision should be constant, no matter the distance of objects to be drawn. The rather simple calculation of linear depth is given by $d = (z - n)/(f - n)$. As can be seen, it linearly maps the distance represented by z into the clipping range $f - n$.

A theoretical upper bound for precision can be calculated for linear depth mapping with fixed-point depth buffers with $s_{\min} = (f - n)/2^b$, where b is the bit width of the buffer.

Linear depth mapping with a fixed-point depth buffer is likely useless for most applications, since differentiability and step width of depth values is the same for all view distances. This means that Z-fighting is as likely to appear close to the viewer as it is to appear far from the viewer. For that reason, linear depth mapping should only be considered when floating-point depth buffers are available.

2.7 Depth Layering

This approach aims to partition the scene into multiple layers that separate each other through clipping boundaries and thus allow for a more dedicated allotment of depth precision [CR11]. All depth layers are drawn from back to front.

The most basic form of depth layering creates multiple coherent viewing frustums, where the far clipping plane of a previous frustum is the near clipping plane of the next frustum.

Basic depth layering solely aims to eliminate Z-fighting and use the depth buffer for visibility determination only. This is done by drawing the scene layer by layer. Each layer contains only objects that intersect the subfrustum of that layer at some point. Ideally, each object in a collection of objects can be assigned to one depth layer only, while in the worst case an object may have to be drawn in more than one layer or maybe in all layers even. It is crucial to clear the depth buffer before each layer is rendered.

Figure 3a illustrates a configuration of two depth layers with their subfrustums, while the graph in figure 3b shows its corresponding distance-depth mapping.

An overlapping of layer boundaries is recommendable in most cases, as the clipping of primitives that cross those boundaries is subject to spatial jitter as well. If a primitive,

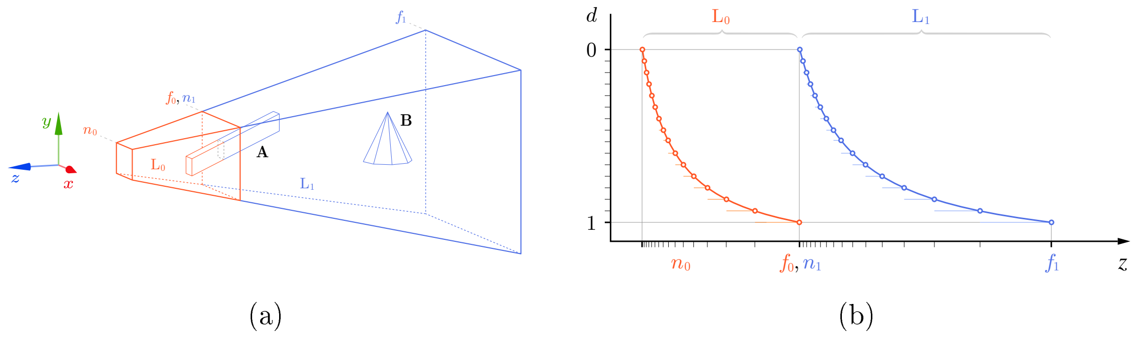


Figure 3: (a) Two connected depth layers with their subfrustums. (b) Corresponding distance-depth mapping. Each layer employs standard projection.

like the upper face of cuboid **A** in figure 3a, intersects the far clipping plane of one layer and the near clipping plane of the following layer, the graphics processor has to split that primitive into multiple smaller primitives. Newly generated smaller primitives subsequently share a face with the near or far plane. The interpolation that is needed to calculate these edges and vertices underlies numeric rounding and quantitation errors. As usual with spatial jitter, arbitrary changes in viewing angle or projection parameters may produce jittering gaps between the two generated edges, leading to the situation that objects appear disconnected and sliced.

Depth layering poses some problems. The overlapping of depth layers works fine for solid objects. It does however cause problems when rendering semi-transparent objects, since the overlapped area in which they are drawn by both layers underlies additive alpha-blending, making that area appear more opaque. Additionally, the drawing order from back to front is contrary to the preferred drawing order for early depth tests.

3 Evaluation Setup

A good measure for the distribution of depth precision is minimum triangle separation s_{min} or MTS in short [AS06]. It describes the minimum distance that two primitives have to be apart in the depth domain for them to be mapped to distinct depth values. Figure 4a illustrates MTS with two planes placed at different distances from the viewer. MTS is a function of view distance and generally increases with distance.

3.1 Measurement Method

Because the final visual appearance is the motivation of this work, MTS is measured using image analysis. In theory an analytical approach is possible, but there are a lot of factors that have to be considered. Every floating point computation in the rendering pipeline is introducing new numerical errors and must be taken into account.

The simple idea behind measuring MTS with image analysis is to extract the content of a frame's colour buffer and test for the occurrence of certain pixel colours. A white plane is

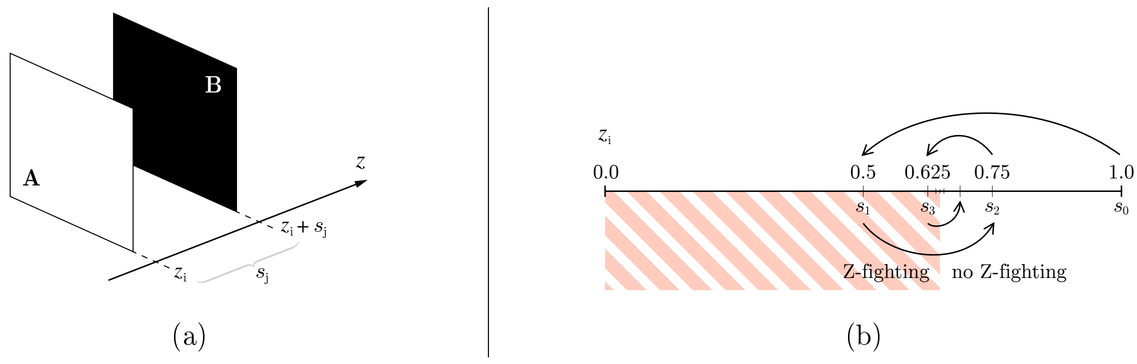


Figure 4: (a) The measurement setup. The camera used for projection and measurement looks along the z-axis such that black plane **B** is covered by white plane **A**. (b) The process used for approximating MTS at distance z_i.

drawn in front of a black plane as shown in figure 4b. Z-fighting can thus be easily detected when black pixels are overdrawn on top of white pixels.

In order to avoid false negatives and bias towards any particular viewing angle, both test objects are rotated in all three dimensions around their geometric centres. The rotation increases numerical complexity, since it ensures the model matrices have rotational components. The rotation animation variable a is incremented by a fixed amount Δa in each frame and is used to control the rotation. For that, a is multiplied by rotation step size r_s and offset by rotation offset r_o with distinct values for each dimension and then input into the sine function. Its result is then scaled by the fixed rotation range r_r to produce the final rotation. This procedure allows to repeat the same object rotations for each distinct set of view distance and object separation — hereafter denoted as a (z, s) tuple. In turn, each distinct step in rotation is hereafter denoted as a (z, s, r) tuple. a is incremented until the animation end a_e is reached, or Z-fighting appeared for the first time in the current tuple (z_i, s_j). If no Z-fighting occurred during the animation, the object separation at the given view distance is assumed high enough to produce no Z-fighting at all.

Measuring takes place frame by frame in a couple of nested pseudo loops. The outermost loop controls the view distance z_i of the test objects. The convention was to put the closer of the two objects at the exact view distance and put the farther object at the view distance plus the currently tested separation distance.

The next control variable is the object separation s. This is what eventually results in MTS through an approximation process. For all measurements, s_0 was set to a value high enough to produce no Z-fighting. Initially the separation step $\Delta s_0 = -\frac{s_0}{2}$. From there on the approximation process starts. The previously described rotation is animated frame by frame, forming a distinct (z_i, s_j, r_k) tuple. As previously stipulated, no Z-fighting occurs for any r_k in (z_0, s_0, r_k).

Generally, the approximation of MTS follows a binary search algorithm. Figure 4b illustrates this process with a number line. The next separation is always given by $s_{j+1} = s_j + \Delta s_j$, where Δs halves each time it crosses the Z-fighting boundary.

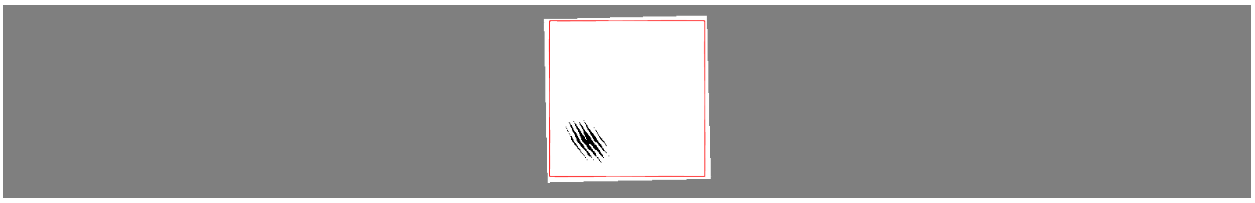


Figure 5: Both test planes with a maximum rotation still enclose the measuring kernel (red square) fully. The black pattern is Z-fighting caused by the black plane showing through.

The approximation process stops, when a given precision in separation was reached. Additionally, the test objects are scaled proportionally to their distance from the camera, so that they always appear the same size. This ensures that both objects cover the exact same screen space area and Z-fighting is allowed to occur on the entirety of the area.

3.2 Measurement Configuration

A measurement *Kernel* was set with a size of 200×200 pixels. The *Clipping Range* was set to $[1.0, 10^9]$. The clipping planes are too separated for standard projection but reasonable for extended depth buffer strategies. The *Camera* was positioned at the origin, to limit the amount of spatial jitter and increased Z-fighting as in the floating origin technique [Tho05].

The *Test Objects* comprise two exact same planes, which are subdivided four times. A plane thus consists of 16 quads (32 triangles) that are made up of 25 vertices evenly distributed in the range $[-0.5, 0.5]$. The *Drawing Order* is set to draw the farther one of the two planes last. Together with the depth function set to *less than or equal to*, it ensures that the farther black plane overdraws the closer white plane whenever the two's depth values are indistinguishable from each other. This in turn makes sure that invalid black pixels are drawn in two cases: 1) if the separation between the planes is zero, 2) if the separation is between zero and the effective value of MTS. Black pixels are only drawn when Z-fighting occurs.

The *Animation End* a_e was set to 10.0. Together with the *Delta Animation* Δa, which was set to 0.1, it ensured that at least 100 frames were measured in order to detect Z-fighting. The *Rotation Step Size* r_s and *Rotation Offset* r_o were arbitrarily set to $(0.50, -0.45, 0.60)$ and $(0.30, 0.12, 0.67)$ to avoid repetition of rotations for potentially long-lasting measurements. The *Rotation Range* r_r was limited to ± 1 degree in order to keep the measurement kernel as big as possible, while the test planes still enclose it at any time (see figure 5).

The *View Distance* starts at the near clipping plane with a value of 1.0 and gets doubled every time MTS was approximated to a sufficient extent for each measured distance z_i. The last tested value is 2^{29} and defines the maximum distance that the closer test object will be placed from the viewer.

The first distance 1.0 was chosen to be exactly at the near clipping plane in order to measure the potentially smallest value for MTS for each approach. This is little problematic though, as the slightest rotation produced by the animation process brings half the plane's vertices and fragments closer to the camera and lets them be clipped. The problem was solved by measuring the first distance independently and setting it marginally higher than 1.0, to $z_0 = 1.005$.

Dependencies and implications could not be determined for all parameters since there are so many. For this reason, some parameters had to be set arbitrarily. Although the entire measurement configuration should produce different absolute measurement results as compared to other configurations, it can be assumed that relative differences between approaches are largely comparable and produce Z-fighting with certain viewing angles equally.

4 Results and Evaluation

Figure 6 illustrates the measured MTS for 32-bit floating-point and 24-bit fixed-point depth buffers respectively. Depth layering was also measured with reversed and linear mappings, but did not achieve better results than their standalone counterparts.

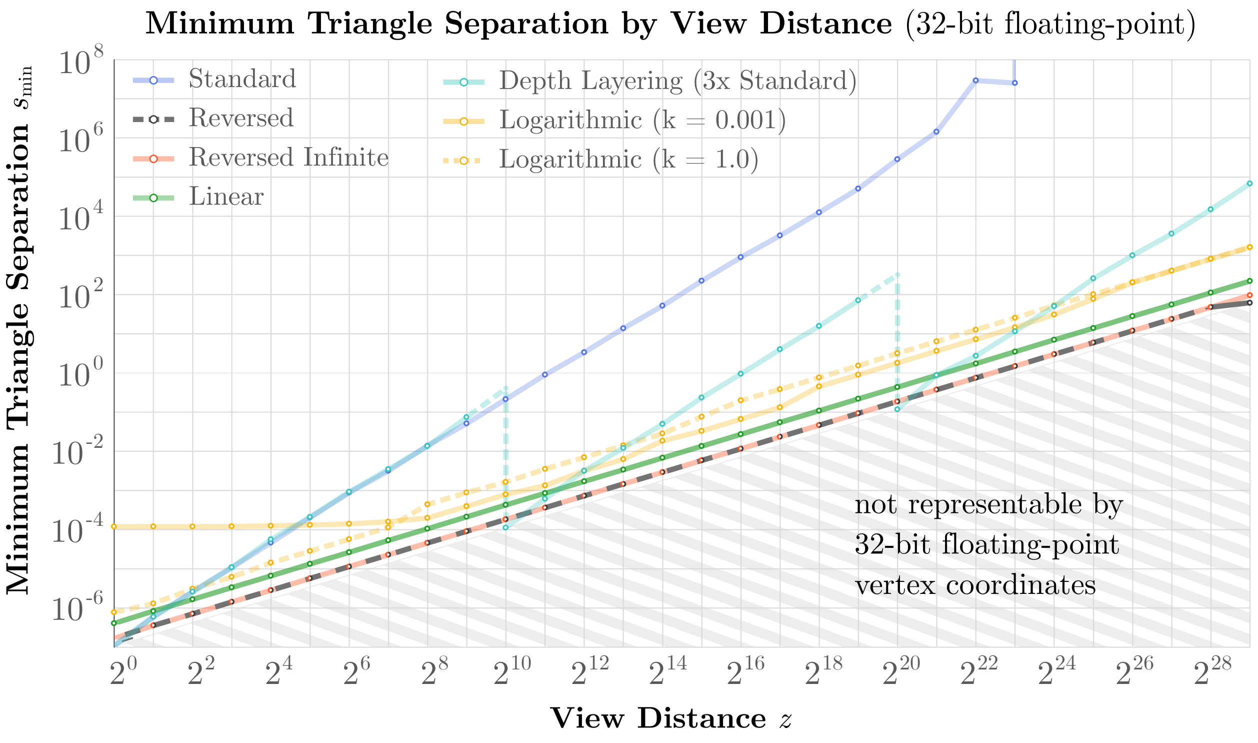
Minimum triangle separation turns out to behave somewhat counter-intuitively, at least for certain measurement configurations. Not all results are easily explained and thus a number of assumptions had to be made which need further work and require more specialized measurement methods to be either verified or falsified. Causes of different results in MTS are a result three factors: 1) the dependency on d and thus the near and far clipping planes, 2) passed on numerical errors and 3) the precision of vertex coordinates that must still be able to separate triangles from each other. Differentiating all three factors and attributing them exact influence in the measured values is challenging.

The cross-hatched areas designated *not representable by 32-bit floating-point vertex coordinates* in both graphs illustrate the minimum numeric value for MTS at any given distance. Lower values cannot be represented using 32-bit floating-point vertex coordinates because z and s cannot be considered individually but only as their sum.

For standard projection, floating-point buffers have more accuracy at the near clipping plane than the fixed-point buffers, and they behave equally towards the far clipping plane. The increased precision of the floating-point buffer seems unnecessary as both buffers have an MTS of $s_{\min} < 10^{-6}$ below a distance of 2. This would be micrometres if we would measure distance in metres. Standard projection also breaks at a distance of greater than 2^{23} where all values become infinite. Generally, standard projection has a bad distribution of depth-buffer values with too much precision at the near clipping plane and bad precision at the far-clipping plane.

Reversed projection also has a bad distribution with a fixed-point buffer and thus suffers from the same problems as standard projection. With a floating-point buffer on the other hand, it has the best overall distribution of precision from all tested methods. It is close to reaching the theoretical lower bound of MTS, which is defined by the limits of vertex coordinate precision.

Reversed infinite projection behaves similar to reversed projection, with slightly worse precision close to the far clipping plane. This is due to the fact that the method *holds back*



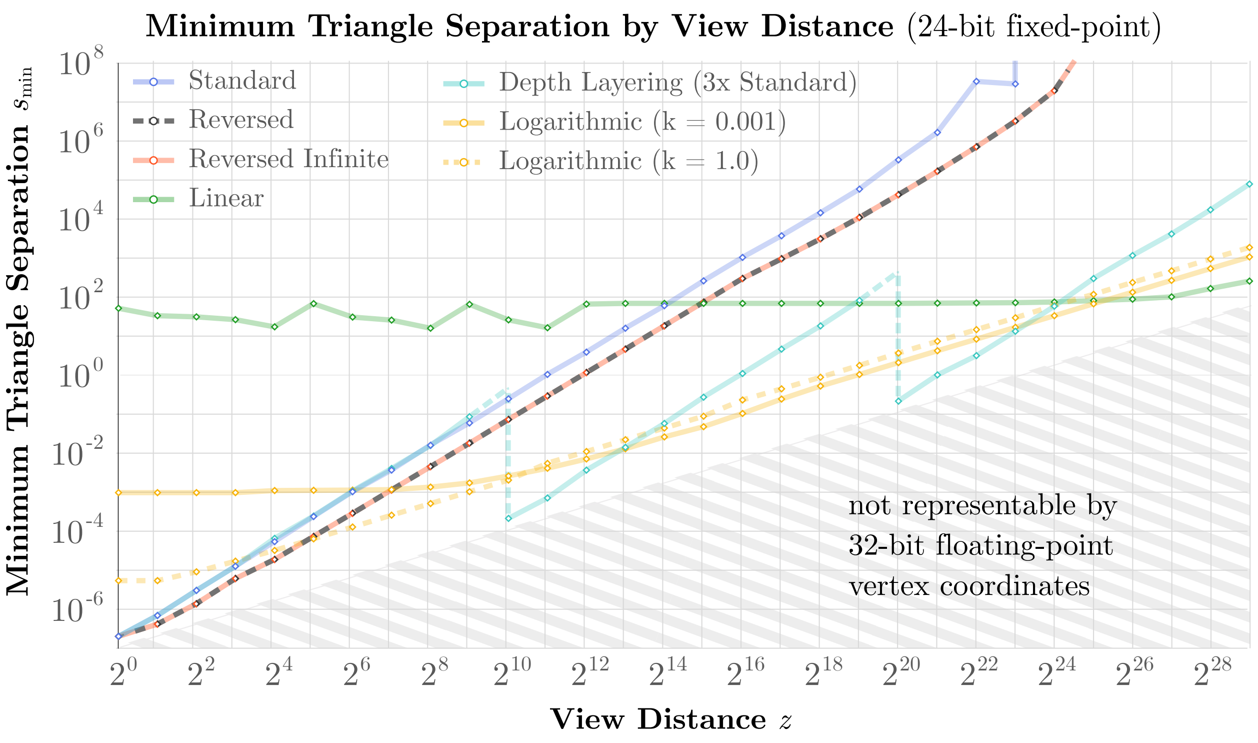


Figure 6: MTS by view distance with 32-bit floating-point (top) and 24-bit fixed-point (bottom) buffers. The x-axis shows the distance between the camera and the front plane. The y-axis shows the minimum measured separation between the planes without artefacts. The cross-hatched area describes the definite lower bound for MTS as vertex coordinate precision underlies similar technical restrictions as depth buffers.

precision for values beyond the far clipping plane, while reversed projection can spend it before the far clipping plane. When no far clipping plane is needed and a 32-bit floating-point buffer is available, this seems to be the best method. In case a far clipping plane can be established, reversed projection should be used instead. To get the best results for objects that receive infinite depth buffer values, the draw order needs to be adjusted, so that objects get drawn from back to front.

Linear depth mapping with fixed-point buffers is close to its theoretical MTS according to the equation given in section 2.6. The uniform distribution makes it unusable at the near clipping plane. With floating-point buffers it comes close to the precision of reversed and reversed infinite projections. Although the precision has a good distribution, it does not have any advantages, making it worth using over reversed or reversed infinite projections. It is less precise over the whole clipping range and worse performance is to be expected due to manually writing to the depth buffer.

Logarithmic depth mapping has similar distributions for floating- and fixed-point buffers. A higher k is resulting in more precision at the near clipping plane and less precision at the far clipping plane. For floating-point buffers the same limitations as the linear depth mapping apply. While the distribution is close to reversed and reversed infinite projections it is never better, and the performance is predicted to be worse. Logarithmic depth mapping is promising for use with fixed-point buffers as it has a good distribution over the whole depth buffer range. The constant k also allows fine-tuning of the precision distribution for specific applications. One must still take into account the performance or accuracy limitations described in section 2.5.

Depth layering does not improve the precision to a level that makes it a perfect solution. It is, as other techniques, limited by vertex coordinate precision. While improving the precision in each layer, it worsens fast again. Because of the additional cost in implementation complexity and performance, it is not worth using with floating-point buffers. It could be a viable solution in use with fixed-point buffers if enough layers are employed. Although, implementation complexity and performance implications still need to be taken into account.

5 Conclusion

This work presented, evaluated and compared different depth buffer techniques. Based on measurement results, it can be concluded that standard projection is unsuitable for use in large scenes, and a different approach should be used.

The best results are given by reversed and reversed infinite projections with 32-bit floating-point buffers. Additional to offering the best depth buffer precision over the whole clipping range, they are also easy to implement by just modifying the projection matrix. The performance, while not measured, is expected to be equal or better than other presented techniques. If only 24-bit fixed-point buffers are available, a solution is not as obvious. Logarithmic depth mapping might be a good solution, but it has to be decided on a case-by-case basis.

References

[AS06] Kurt Akeley and Jonathan Su. Minimum triangle separation for correct z-buffer occlusion. In *Proceedings of the 21st ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*, GH '06, page 27–30, New York, NY, USA, 2006. Association for Computing Machinery.

[CR11] Patrick Cozzi and Kevin Ring. *3D Engine Design for Virtual Globes.* CRC Press, 1st edition, June 2011.

[Gro] Khronos Group. Early fragment test. OpenGL Wiki. `https://www.khronos.org/opengl/wiki/Early_Fragment_Test`. Accessed: 2021-05-12.

[Kem09] Brano Kemen. Logarithmic depth buffer. Outerra Blog, August 2009. `https://outerra.blogspot.com/2009/08/logarithmic-z-buffer.html`. Accessed: 2021-05-12.

[Kem12] Brano Kemen. Maximizing depth buffer range and precision. Outerra Blog, November 2012. `https://outerra.blogspot.com/2012/11/maximizing-depth-buffer-range-and.html`. Accessed: 2021-05-12.

[Len11] Eric Lengyel. *Mathematics for 3D Game Programming and Computer Graphics, Third Edition.* Course Technology Press, Boston, MA, USA, 3rd edition, 2011.

[LJ99] Eugene Lapidous and Guofang Jiao. Optimal depth buffer for low-cost graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, HWWS '99, page 67–73, New York, NY, USA, 1999. Association for Computing Machinery.

[PS12] Emil Persson and Avalanche Studios. Creating vast game worlds: Experiences from avalanche studios. In *ACM SIGGRAPH 2012 Talks*, SIGGRAPH '12, New York, NY, USA, 2012. Association for Computing Machinery.

[Ree15] Nathan Reed. Depth precision visualized. NVIDIA Developer Blog, July 2015. `https://developer.nvidia.com/content/depth-precision-visualized`. Accessed: 2021-05-12.

[Tho05] Chris Thorne. Using a floating origin to improve fidelity and performance of large, distributed virtual worlds. In *2005 International Conference on Cyberworlds (CW'05)*, pages 8–pp. IEEE, 2005.

[UD12] Paul Upchurch and Mathieu Desbrun. Tightening the precision of perspective rendering. *Journal of Graphics Tools*, 16(1):40–56, 2012.