

Antiquation
of the
Back Clipping Plane

Technical Report

David A. Hostetler

EE 598 Spring 1999
New Mexico State University
Submitted to: Dr. E.E. Johnson

Contents

1. Introduction	1
2. Background	1
3. Motivating Problem	2
4. Choose a VSD Algorithm	3
5. Complications	4
6. Aliasing and Z-buffer Size	5
7. Geometry Calculations	7
8. Polygon Densities	8
9. Back-Face Culling	8
10. Counting Polygons	9
11. Variable Level of Detail	10
12. Scaling to Insignificance	12
13. VLOD for Terrain	14
14. Results	15
15. Recommendations	16
16. Conclusions	16
17. References	19

1. Introduction

This technical report describes an investigation during the spring semester of 1999 at New Mexico State University to improve real-time computer graphics. Specifically, the focus is the back clipping plane and the repercussions of its removal. Of primary concern is the task of polygon count reduction (referred to simply as 'polygon reduction'), of which several methods are examined. Also of paramount importance is the relationship between the back clipping plane and the choice of a method for visual surface determination. Discussed in detail is the technique of Z-buffering (also called depth-buffering).

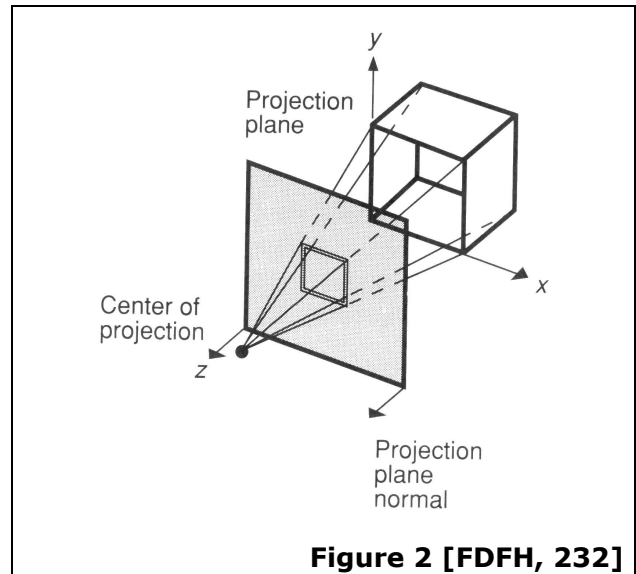
The objectives of the investigation were twofold; first and foremost to discover whether the impetus for a back clipping plane could be eliminated, and second to explore and estimate the effectiveness of different methods of compensating for the absence of the back clipping plane. If the back clipping plane is eliminated from the rendering process, it is crucial to consider how this affects the visual surface determination (VSD) algorithm. The reason is that as scene complexity increases in real-time graphics applications, the VSD algorithm "very rapidly becomes the limiting factor in rendering realistic worlds." [ABRASH, 765]

2. Background

2.1 Graphics Pipeline

The basic goal of computer graphics (at least for the purpose of this discussion) is to accurately represent logically-defined, continuous, three-dimensional scenes on a discrete, two-dimensional medium. In order to understand most computer graphics algorithms, it is best to consider them within the context of the rendering pipeline. The pipeline concept is a very convenient and attractive way to visualize (and ultimately implement) graphics operations. The major stages of a general rendering pipeline are shown in Figure 1. [FDFH, 867]

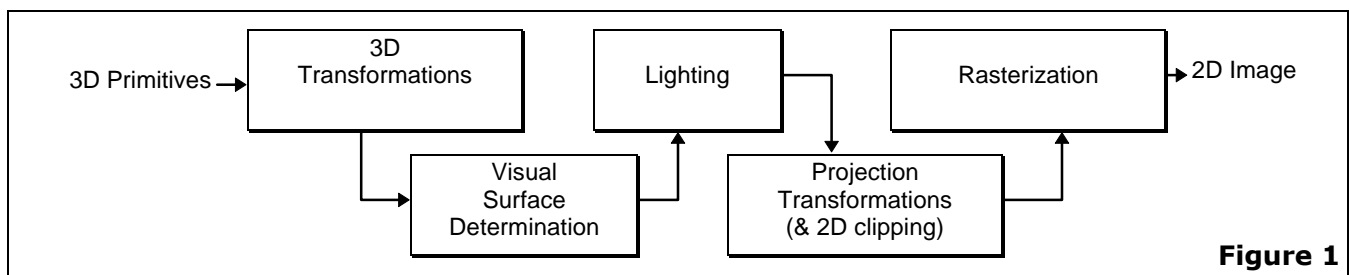
First of all, a *primitive* is simply a constituent part of the scene, defined either mathematically or logically (via an organized



data structure) such that its disposition within the scene can be derived. A common primitive is a convex polygon.

A rudimentary pipeline could consist of as little as the *projection* and *rasterization* stages. These two stages perform the key functions in the creation of an image. The *projection* stage essentially eliminates the third dimension by projecting the scene onto a projection plane positioned in front of the viewpoint, or center of projection (Figure 2). The *rasterization* stage makes the conversion from a continuous mathematical description to a discrete pixel map.

The *transformations* stage allows for the viewpoint to assume different angles and locations within the 3D scene. *Visual Surface Determination* (VSD) is shown as a single stage, when in actuality there are numerous places within a pipeline where VSD related activities occur. In fact, the particular surface determination algorithm that will be discussed (Z-buffering) takes place within the rasterization stage. At the very least, though, three-dimensional volume clipping would occur in the VSD stage as it's shown. Volume clipping is just the process of removing primitives from further consideration because they are located outside the current *field of view* (FOV), which certainly qualifies



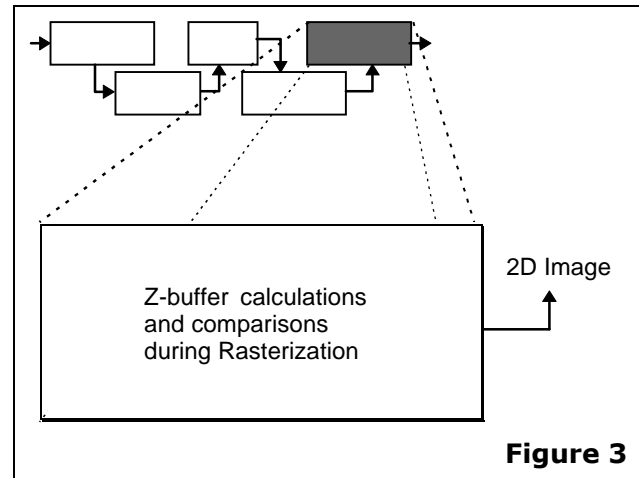
as visual surface determination. Of course, the real difficulty in visual surface determination involves identifying primitives that *are* inside the FOV, but can't be seen due to occlusion. At any rate, pipelining the rendering process is an efficient approach to dealing with the many complicated processes that have to occur to generate a realistic two-dimensional representation of a three-dimensional scene.

2.2 Z-buffer Algorithm

Having covered the bare-bone fundamentals of computer graphics, it is also prudent to review how a basic Z-buffer algorithm works. In short, Z-buffering is a very flexible method with low computational overhead but a steep requirement for physical memory. A buffer is created that is equal in extent to the color buffer (or frame buffer), i.e. large enough to have an entry for each pixel in the final two-dimensional image. For example, an image of dimensions 800 x 600 pixels would require a frame buffer with enough memory for 480,000 color values, and a Z-buffer with enough memory for 480,000 depth values. These buffers are not necessarily equal in memory size, as a color value and depth value are unrelated in size. Stored in this Z-buffer are the depth values of the visible projected primitives.

The method proceeds as follows: The Z-buffer is initialized to the value that corresponds to the furthest possible Z value (a depth beyond which primitives are not rendered). The color buffer is initialized to the background color. As primitives are transformed, projected and clipped in order to be rasterized to the final image, the depth values of the vertices are retained. When a pixel value is ready to be written to the color buffer at a given location, the Z-buffer algorithm is executed (Figure 3):

- 1.) The depth value currently in the Z-buffer at the specified pixel's location is compared to the depth value of the new pixel.
- 2a.) If the old value is closer to the viewer than the new one, then no change is made to either the color or depth buffers.
- 2b.) If the new value is closer, however, then it replaces the old Z value and the new color replaces the old color at the same location in the frame buffer.



Only the actual vertices of the primitives will have a specified Z component, but a Z component is needed for every pixel within the rasterized image in order to do the comparisons. This obstacle is overcome by interpolation using planar coherence, where the difference between the Z values of two vertices are used to calculate an incremental Z for that primitive, which is simply added to each Z value as the projected primitive is traversed in two dimensional space. [ANGEL, 280]

Thus, at any given time, the Z-buffer contains the depth values for the closest element that's been projected up to that time, for every pixel in the image. For this reason, the Z-buffer algorithm is specified as having *image-precision*, as opposed to *object-precision*. [FDFH, 668]

A VSD algorithm with object-precision performs object-to-object comparisons prior to rasterization. Naturally, this requires the use of more complicated techniques to calculate intersections and such, and it is not guaranteed that each *pixel* in the final image represents the primitive that is actually visible.

3. Motivating Problem

3.1 Back Clipping Plane

The impact that a back clipping plane has on the performance of rendering interactive outdoor scenes is very positive. The impact that it has on image fidelity is very negative, and the purpose of this investigation is to determine if that negative impact is avoidable. Ultimately, the idea is to show that there is no longer a need for the back clipping plane, that technologies and methods exist which make the back clipping plane obsolete altogether.

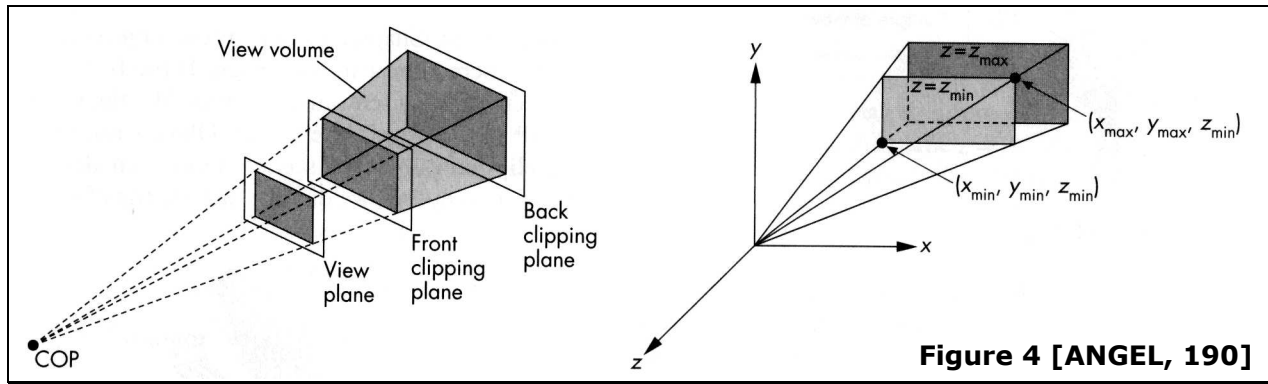


Figure 4 [ANGEL, 190]

A back clipping plane is used to construct a view frustum (volume), which bounds the three-dimensional area that defines the current field of view. This frustum is used to do volume clipping on the set of primitives for a scene (see Figure 4). Any primitive (polygon, for instance) that lies outside this view volume is not included in any successive processing stages. Of course, that's appropriate for primitives that *truly* lie outside the field of view. Anything behind the viewpoint (i.e. in front of the front clipping plane) is considered to be behind the viewer and is trivially rejected. Those primitives that are located too far to the left, right, above, or below the view volume can't be seen either, so no harm is done by eliminating them.

However, when primitives are discarded because they are located behind the back clipping plane, a nasty side effect manifests itself. Consider that such primitives are actually visible (assuming they are not occluded) because they are, in fact, *inside* the field of view. The obvious consequence is that the rendered image does not include objects that ought to be there. That is inconsequential if the viewer doesn't know those objects are missing. Thus, the real problem occurs when the image is rendered within an interactive, or at least real-time animated, application.

3.2 Real-time Applications

The clipping planes of the frustum are bimodal in nature, meaning that whatever isn't on one side of the plane is on the other. Thus, as primitives cross the boundary of the clipping planes, due to movement of either the observer or the object, they simply appear (or disappear) in the next rendered image. This is acceptable for all of the other clipping planes because the viewer isn't aware of what's outside the field of view. But when objects enter or exit the frustum through the back clipping plane, the

effect is extremely unrealistic: they just pop into or out of view, in an area where the viewer was already looking. As a result, the viewer becomes aware that the object was, in fact, missing, either from the previous image or the subsequent one.

For example, consider a terrain scene. In one image a hill or similarly large object is clipped by the back plane, and then in the next image the view volume has moved just enough to include the front of the hill and so there it is, a hill where there was none before. The object does not have to be large for the effect to be undesirable, either. Anything that would cover more than a few pixels if it was rendered is going to be noticed when it suddenly pops into or out of view. This is the problem with back clipping planes. The obvious solution is to simply get rid of the back clipping plane.

4. Choose a VSD Algorithm

After all, there is nothing that absolutely prevents the removal of the back clipping plane. There are several reasons that it might be inadvisable, which will be addressed shortly, but there's no reason it *can't* be done. This is where Z-buffering comes into play. With the back plane gone, the VSD algorithm of choice is definitely Z-buffering, and for several reasons.

The fact that Z-buffering is an image-precision algorithm is one of the things that make it so attractive for this situation. Without a back plane, there will be a much higher polygon (primitive) count per frame, which has two consequences. First, a VSD algorithm with object-precision must do object to object comparisons, and the number of comparisons scales proportionally with an increased polygon count. It might also need to do model to model comparisons, where a model is a collection of polygons treated as a single object. Object-object comparisons are typically much more complex and time

consuming than the elementary pixel to pixel comparison that occurs with Z-buffering. [FDFH, 650]

The second significant advantage of Z-buffering is that primitives can be correctly tested for visibility in any order. The primitive list does not have to be presorted in any way before processing. Presorting is not something an application would want to perform for large outdoor scenes with no back clipping plane. A small change in the view position or angle could result in the inclusion of a large subset of primitives at a depth completely unrelated to the average depth of the previous set of visible primitives. In addition to being fairly complex and imprecise, sorting is at best an $O(N \log N)$ algorithm (for N objects) and thus is generally best avoided.

5. Complications

5.1 Too Many Polygons

The purpose of the back clipping plane, and really all of the clipping planes, is to reduce the number of primitives that must flow through the entire rendering pipeline. Processing primitives is very costly (in terms of clock cycles) and so much of computer graphics involves identifying things that don't have to be done. Pushing a polygon that ultimately will never be seen, all the way through the pipe is something that doesn't have to be done. That's why FOV clipping is done. The incentive for the presence of a *back* clipping plane comes from the fact that many scenes have so many polygons that even just the set of visible ones is too many for a real-time rendering pipe. A back clipping plane reduces this set. In fact, the depth of the back clipping plane is often a variable parameter called 'visibility' which the user can alter to tune performance (a speed vs. fidelity tradeoff). *True visibility* would imply the absence of the back clipping plane.

One very nice consequence of using a back clipping plane is that the standard deviation of the count of polygons that must be fully processed for any given image is bounded. The fixed view volume typically only holds so many polygons, and so the effect of changing the view from an area of low polygon density to one of high polygon density is reduced by the back clipping plane. If the plane is absent, changing the view to a high density area could increase the polygon count by several orders of magnitude. This would be detrimental to a real-time application.

5.2 Z-buffer Aliasing

There are several disadvantages inherent in the use of Z-buffering, disadvantages that could very well foil any attempt to eliminate the back clipping plane. The back plane is there for a reason, and it is imperative to consider the consequences of removing it. The first disadvantage of Z-buffering exists whether or not a back plane is present, but is exacerbated by its absence.

The Z-buffer typically has entries of 16 or 32 bits in length. This bit depth determines the range of possible Z values that can be stored in the buffer. For example, if the values are stored as (unsigned) integers, the range of a 16 bit Z-buffer entry is simply:

$$0 < Z \leq (2^{16} - 1) \quad \text{or} \quad 0 < Z \leq 65,535$$

The implication of this range is that if the Z values of the three-dimensional vertices exceed this range, then aliasing is introduced into the projection process. That is, two points with different Z values in three-dimensional space which project onto the same pixel in the color buffer might map to the same Z-buffer value, preventing correct visibility comparisons.

In other words, the Z-buffer precision is finite, determined by its bit depth, and this precision could very well be less than the precision that results from the scale/granularity ratio in the 3D scene coordinates. This is likely to be the case when the scene is an expansive, outdoor environment, or if "objects defined with millimeter detail are positioned a kilometer apart." [FDFH, 671] This aliasing in the Z-buffer is sometimes called "Z fighting", because two vertices with distinct depth values are fighting for the same Z-buffer value.

The aliasing can perhaps be avoided entirely via the back clipping plane. If for every frame, the scene is translated so that the viewpoint lies at the origin, and rotated so that the view vector lies parallel to the Z axis (as is often done), then the view volume will be entirely in either the positive or negative half Z plane. The advantage of this is that if a back clipping plane is used, then there is a bound on the maximum Z value that need be stored in the Z-buffer. For a 16 bit Z-buffer, the back plane could be placed such that any point with a Z value greater than can be expressed in 16 bits is automatically rejected during volume clipping, and thus no aliasing is possible.

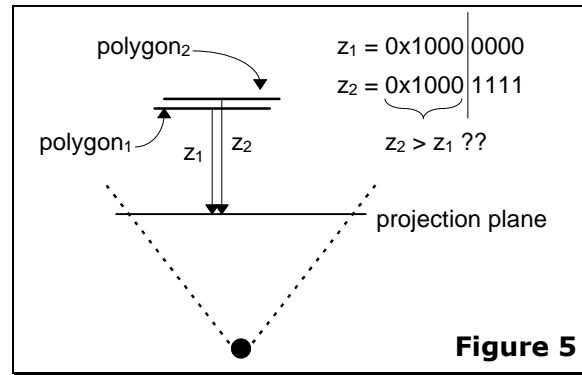
5.3 Z-buffering and Performance Degradation

The second disadvantage of Z-buffering stems from the fact that it is an image-precision algorithm. The visible surface comparisons take place during the rasterization stage, after projection, since the comparisons must be done at the pixel resolution. While this makes life (and rendering) easier in several ways, it also has an unfortunate consequence.

Every primitive that passes the FOV clipping (which is done regardless of which VSD algorithm is used) must be processed by the pipeline up to the rasterization stage. So while Z-buffering avoids the overhead in the early stages that would accompany object-precision algorithms, it causes additional loading in the later stages. In particular, lighting, projection, and 2D clipping must occur for a larger primitive set, since the Z-buffering does not eliminate occluded objects until after these stages. Normally, this is an acceptable consequence of Z-buffering because there is a back bound on the view volume, hence an effective bound on the polygon count for any given frame.

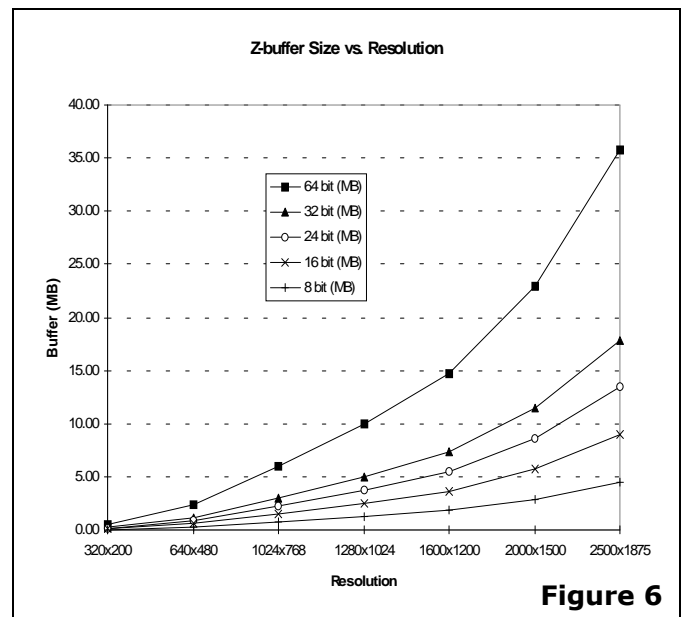
6. Aliasing and Z-buffer Size

As stated above, the finite precision of the Z-buffer introduces aliasing which may or may not be alleviated by the back clipping plane. Certainly, if the plane is removed, the aliasing should be addressed. The amount of aliasing that occurs depends on the proximity of objects to each other. If depth values have 32 bits, and Z-buffer entries are only 16 bits, then the least significant two bytes of the depth value may be truncated when it is stored in the Z-buffer (depending on the mapping used). Thus, aliasing occurs when two points that map to the same pixel have a difference in projection distance of less than 2^{16} . Figure 5 shows an example of 8 bit depth values aliasing in a 4 bit Z-buffer. If the Z-buffer algorithm cannot resolve a comparison because of aliasing, then neither the Z-buffer or the color buffer are updated (to avoid the memory writes). The risk is that polygon₂ (in the example of Figure 5) is processed first, thereby filling the Z-buffer and color buffers with its values. Then, whenever polygon₁ gets processed, its values are discarded because it aliases onto polygon₂, and the incorrect polygon is rendered in the image. So the fact that the primitives are processed in essentially random order is also at the root of aliasing.



The knee-jerk solution to the problem is to increase the bit depth of the Z-buffer until it's sufficient to contain the range of the scene coordinate values themselves. That has been the solution adopted by many applications, even with the presence of the back clipping plane. In fact, for large scenes, the coordinates can be kept in floating point format and the Z-buffer itself holds the depth values in floating point (usually 32 bit, single precision). This solves the aliasing problem, but it seems like an inefficient solution. For one thing, the size requirements of the Z-buffer should not be ignored. The total size of a typical Z-buffer can range anywhere from, roughly, 1MB to 7MB, depending on the bit depth and image resolution. Figure 6 shows the buffer sizes for the full spectrum of image resolutions, for several depths from one to eight bytes.

The last few resolutions are just extrapolated using the common 4/3 ratio. They represent resolutions likely to be used in the near future, and so the accompanying buffer sizes represent the



future requirements for Z-buffering applications. Many real-time rendering applications today are already running at a resolution of 1280x1024. As the chart shows, at the higher resolutions a 32 bit Z-buffer can require as much as 18 MB. A 64 bit Z-buffer could conceivably require over 35 MB! At these sizes, just the act of initializing the Z-buffer between frames becomes a performance concern.

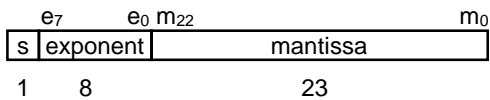
6.1 Ideas about Antialiasing

Instead of merely matching the Z-buffer precision to the scene coordinate precision, perhaps a benefit could be had from utilizing a non-linear mapping from scene depth to Z-buffer depth. This involves adjusting the concept of Z-buffer space.

Consider a 16 bit Z-buffer. Each entry is capable of containing any of 2^{16} distinct values. Is that enough to differentiate between every projected point in the visible scene? The temptation is to say 'no', particularly when those points use a 32 bit floating-point representation. However, if the 2^{16} values are not allocated linearly according to depth, then they might very well be enough to maintain the desired level of visual fidelity. The question becomes whether or not there are more than 2^{16} distinct points that get projected onto a given pixel in the color buffer (if no two points ever projected onto the same pixel, then there'd be no need for even a single bit of depth precision!).

The average number of polygons that project onto a pixel is referred to as the image's *depth complexity*. [DFH, 871] Even if the depth complexity is more than 2^{16} points, is there an *acceptable* level of aliasing, if it came with a major reduction in the space requirements for the algorithm? How would an algorithm assign the Z-buffer values, if not linearly, to minimize aliasing artifacts?

One possible method involves masking the 32 bit floating point representation of a depth value. The single precision IEEE floating point format is as follows:



A 16 bit value could be had by simply combining the 8 bits of exponent with the most significant 8 bits of the mantissa. This would involve one multiple bit right-shift operation and one masking bitwise AND operation. [This algorithm would probably

have to be implemented in hardware or handled by a graphics processor to be feasible.]

What are the implications of using such a mapping method? The least significant 15 bits of the mantissa are discarded, which reduces the granularity of the depth. The nature of floating point representation causes less granularity anyway. As the exponent increases, the effective resolution of the mantissa is repeatedly decreased by a power of 2. At very large exponents, the resolution of the mantissa is extremely low; at a range of 2^{127} the least significant digit represents 2^{105} .

With only 8 bits of significance, at what distance does the depth value no longer have sufficient resolution to do accurate depth comparisons? If the units are assumed to be virtual meters, then the scheme loses the ability to resolve at the one meter resolution for any depth beyond 511 virtual meters.

This value is obtained by first assuming a mantissa value with the most significant 8 bits set to one. The least significant digit of the mantissa that is retained in the Z-buffer is m_{15} . To maintain a one meter resolution, the exponent must be less than or equal to 8 to ensure that m_{15} never represents anything greater than 2^0 . Thus, the maximum depth value is 1.11111111×2^8 , or 511 (the IEEE format uses an implicit leading bit, set to one, for normalized numbers).

The implication is that any object beyond 511 meters cannot be correctly tested for visibility against any other object within one meter. A more likely requirement for visual quality is a resolution of 1/8 or 1/16 of a meter. This brings the limit of the 16 bit scheme to as close as 63 or even 31 meters .

A software model of this algorithm was implemented and executed multiple times with 100 million comparisons of random depths. The average percentage of incorrect comparisons (due to aliasing) was 0.0015 %. This would effect less than 30 pixels on a 1600x1200 image. Of course, all it would take is several large, parallel, overlapping polygons with small depth differences to expose the risk of severe visibility errors.

It is worth noting, however, that while this scheme does not seem feasible for outdoor environments, it could be very well suited for indoor scenes. Such scenes might have enough inherent visibility limits that 63 or 31 meters would be sufficient. If so, the use of Z-buffer entries made of 8 bits of exponent and 8 bits of

mantissa would result in a 50% decrease in the size of the Z-buffer.

6.2 Increasing Speed

Attempts to control aliasing are not without merit, and if a way could be found to reduce the memory requirements for the Z-buffer, that would be nice; but the biggest concern, of course, is speed. To be fair, there are residual side effects of reducing the size of Z-buffer entries that can increase performance. Comparing shorter values is going to be inherently faster than comparing longer values, particularly when the Z-buffer algorithm is implemented in accelerated hardware devoted to 3D, which is the case more often than not. Also, if the Z-buffer takes less memory, that frees up more of the faster, premium, video memory for other uses, as well as reducing the time required to initialize the Z-buffer after each frame is drawn. Unfortunately, these increases are minimal. Remember that the ultimate objective is to solve the problems that result from the removal of the back clipping plane. The biggest problem is the tremendous increase in size of the view volume, and thus in the number of primitives within the FOV for any given frame. Just how big *is* that increase? The answer requires a few calculations, obviously, and is the most important step in determining the feasibility of eliminating the back plane.

7. Geometry Calculations

The critical first step is to calculate the volume of the view frustum, with and without a back plane. The equation for the volume of the view frustum is derived from the fact that the volume of *any* cone, pyramid or otherwise, is determined by the formula $V = \frac{1}{3}hB$, where h is the height of the cone and B the area of its base. [LH, 391]

$$V_f = \frac{4}{3}(d^3 - p^3)\tan^2\left(\frac{\Theta}{2}\right) \quad \text{eq. 1}$$

Where Θ is the field of view angle, d is the distance of the back plane from the viewpoint, and p is the distance of the projection window from the viewpoint (see Figure 7). Looking ahead, it will also be necessary to calculate the area of the 'ground' cross section of the frustum (the shaded area in Figure 7):

$$A_g = (d^2 - p^2)\tan^2\left(\frac{\Theta}{2}\right) \quad \text{eq. 2}$$

As the back clipping plane is moved further away, the value of d increases. The idea is to remove the back plane entirely, which

would theoretically make d approach infinity. This is not the case, however, as there is a logical upper bound on d . One cannot really completely remove the back plane. There will, in essence, always be a back plane located at the maximum Z value. Consider that even with no back plane explicitly imposed, a depth value can't be greater than the upper limit of the data type used for the coordinate system. In other words, if coordinates are represented as 32 bit floating point numbers, then there effectively is a clipping plane at 3×10^{38} , which is the largest value that can be expressed in 32 bit floating point notation. If the coordinates are interpreted as being in meters, then the largest floating point distance is roughly three trillion times the size of the known universe. For most applications, this is excessive.

There is, fortunately, a more pragmatic effective bound on the depth, which stems from the fact that most 3D scenes do not encompass the known universe. An ambitious terrain scene might contain as much as 100 km^2 ($10 \text{ km} \times 10 \text{ km}$). Thus, a maximum depth might be 10 km , or 10^4 m (for 1 m resolution). This scene-dependent depth limit will be denoted as d_{\max} . Now, if the back clipping plane is removed, the two previous equations for the volume and area become:

$$V_{f_{\max}} = \frac{4}{3}(d_{\max}^3 - p^3)\tan^2\left(\frac{\Theta}{2}\right) \quad \text{eq. 3}$$

$$A_{g_{\max}} = (d_{\max}^2 - p^2)\tan^2\left(\frac{\Theta}{2}\right) \quad \text{eq. 4}$$

Now expressions for the increase in both frustum volume and ground area that results from removing the back clipping plane can be derived (equations 5, 6).

$$\Delta V_f = \frac{4}{3}(d_{\max}^3 - d^3)\tan^2\left(\frac{\Theta}{2}\right) \quad \text{eq. 5}$$

$$\Delta A_g = (d_{\max}^2 - d^2)\tan^2\left(\frac{\Theta}{2}\right) \quad \text{eq. 6}$$

The inclusion of the ground (or virtual-ground) area in this discussion may be confusing. It is necessary in order to estimate

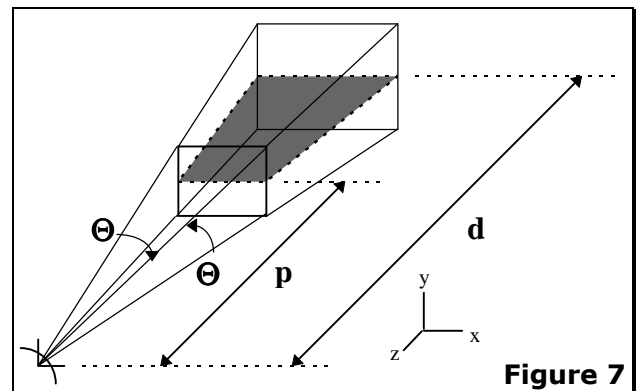


Figure 7

the actual number of polygons, which is much more interesting than just the amount of area and volume (the above equations can be thought of as being expressed in virtual-meters cubed and squared, or vm^3 and vm^2).

In order to get an actual count of polygons, it is necessary to make some assumptions about the density of polygons per scene area and volume. Obviously, these parameters are going to be extremely scene dependent. Imagine a scene located in the vacuum of outer space. The polygon density is likely to be extremely low. A 3D scene in outer space actually represents a scenario very conducive to the removal of the back clipping plane. The relative distances are large enough, to be sure. Also, there is no excuse for using fog or other atmospheric tricks to dampen the effect of the clipping plane. There are no buildings or mountains or other terrain to occlude large amounts of the visible primitive set. Under such circumstances, the back plane could perhaps be eliminated without having to seriously address the pitfalls that come with it. For that very reason, the focus of the investigation is on more earthly scenes: expansive outdoor worlds with lots of open terrain.

8. Polygon Densities

The presence of a terrain mesh, or grid, will account for a large percentage of the polygons within the scene. This is the reason for calculating the ground area within the viewing volume. If the ground area is known, and the resolution of the terrain mesh is known, then it's possible to estimate a value for the number of terrain polygons inside the view volume.

Assume that a terrain mesh will have a resolution of x coordinate units (vm). For every vm^2 , there will be $(2/x^2)$ polygons (triangles). This terrain density will be referred to as δ_{terr} , and has units of polygons per vm^2 . The number of terrain polygons in a frustum with no back plane is expressed in equation 7.

$$\delta_{terr} A_g = \delta_{terr} \left(d_{max}^2 - p^2 \right) \tan\left(\frac{\theta}{2}\right) \text{ polygons} \quad \text{eq. 7}$$

The other source of polygons within a scene is from the models. These might be buildings, bridges, vehicles, trees, animals, soldiers, etc.. Unfortunately, there is a tremendous amount of variability in both the number and detail of the models. While that variability makes it more difficult to estimate the polygon count, it's extremely advantageous to developers because

it allows for scalability and flexibility according to a given system's performance capabilities.

Models can range anywhere from extremely low detail of a dozen or so polygons, to excellent detail of over 1000 or even 2000 polygons. The first step is to address the number of models in the scene. This will be referred to as the model density, δ_{mod} , and has units of models per vm^3 . Next, the detail of the average model is designated as β_{max} , which has units of polygons per model. The polygons contributed to the view volume from models is then:

$$\begin{aligned} & \delta_{mod} \beta_{max} V_f \\ &= \delta_{mod} \beta_{max} \left(\frac{4}{3} (d_{max}^3 - p^3) \tan^2\left(\frac{\theta}{2}\right) \right) \text{ polygons} \quad \text{eq. 8} \end{aligned}$$

Equation 8 does not yet represent the true polygon count that comes with removing the back clipping plane. Instead, it is an expression simply for the number of polygons inside the view volume based on the volumetric polygon density of the scene. There are several other factors that must be accounted for before the real effect of eliminating the back plane is apparent. To begin with, the number of polygons expressed in equation 8 should be divided by two. The reason is that if a terrain mesh is presumed to be present, there will likely only be models on or above the terrain, within the top half volume of the frustum. Instead of dividing by 2, however, it will be assumed that the model density δ_{mod} is really based on the effective volume of the frustum.

9. Back-Face Culling

Next, there is a technique of reducing the number of polygons in the view volume. This technique is called back-face culling, and it exploits the assumption that everything in a scene is an enclosed volumetric object.

If every model is a solid polyhedron, and all of its faces have been defined so that their normals are directed outward from the polyhedron, then it follows that only the faces whose normals point towards the viewpoint are visible. All other faces (those whose normals point away from the viewpoint) are completely occluded by the rest of the model (see Figure 8). The sides of the object that are represented with dotted lines are back-facing polygons and would be rejected.

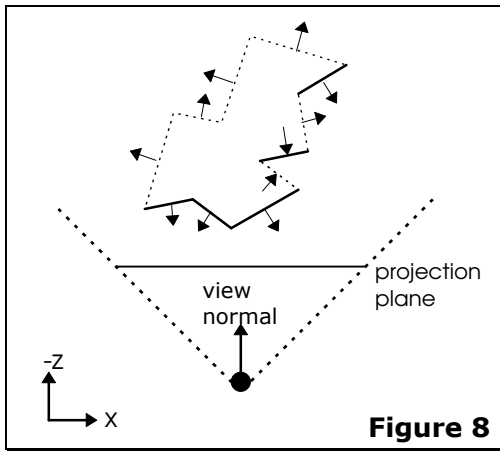


Figure 8

If the scene has been transformed as described previously, so that the viewpoint is at the origin and the view vector is parallel to the negative Z axis, then the test for back-facing polygons reduces to a sign test for the Z coordinate of each polygon's normal. This results in an excellent tradeoff. So much so, in fact, that it is well worth the effort to add faces to a model where necessary to insure that they can only ever be visible from one side.

For the sake of simplicity, it will be assumed that the models in the hypothetical scene are basically symmetric about all three axes. This implies that regardless of the orientation of the model to the viewpoint, only half of the model's polygons will be visible. [FDFH, 664] That reduces the contribution of model polygons by 50%, although its effect will be parameterized as c_{mod} .

The effect of back-face culling on the amount of visible terrain polygons is not so dramatic, only because the assumption of symmetry is not imposed on the terrain as it is on the average model. While terrain is not actually a solid polyhedron, it can be interpreted as a giant solid polyhedron, as long as the viewpoint is not allowed below the terrain mesh. Terrain with lots of elevation differences would benefit from back-face culling much more than terrain that is relatively flat. A conservative estimate of the effect of back-face culling on terrain might be 10% reduction, but for the

sake of the calculations it will be parameterized as c_{terr} ($c_{terr} = 0.9$ for a 10% reduction).

Back-face culling is in essence an algorithm for hidden surface removal more than it is a VSD algorithm, in much the same way as is FOV clipping. Both of these methods ease the burden on the true VSD algorithm, as well as on the rest of the pipeline. Anything that shortens the list of polygons that must flow through the pipe is a boon to performance.

The expressions for the number of both terrain and model polygons that result from the removal of the back plane are now as follows:

Terrain: $c_{terr} \delta_{terr} (d_{max}^2 - p^2) \tan(\frac{\Theta}{2})$ polygons **eq. 9**

Models: $c_{mod} \delta_{mod} \beta_{max} (\frac{4}{3} (d_{max}^3 - p^3) \tan^2(\frac{\Theta}{2}))$ polygons **eq. 10**

10. Counting Polygons

Equations 9 and 10 represent the polygons that must be processed, if the back clipping plane is removed and no other steps are taken. An example calculation follows.

- Let:
- Θ = 60°
 - d_{max} = 1×10^4 vm (10 km)
 - d = 1000 vm
 - p = 0.2 vm
 - δ_{terr} = 2×10^{-2} polys/vm² (10 vm grid)
 - δ_{mod} = 5.0×10^{-7} models/vm³ (500 models/km³)
 - β_{max} = 500 polygons/model
 - c_{mod} = 0.50 (50% reduction)
 - c_{terr} = 0.90 (10% reduction)

For the above parameters, the totals are shown in Table 1.

Those numbers bear excellent testimony to the importance of the back clipping plane, and for the daunting problem of how to compensate for its removal. For the given parameter values, removing the back plane means processing over **858** times as many polygons! For this example, the models are accounting for the majority of the polygons. Selecting a higher resolution terrain mesh or lower detail models would alter this, of course. Even with a back plane, the polygon count is almost

Polygon Totals with and without the back plane				
Back Plane present?	Terrain Polygons	Model Polygons	(Models)	Total Polygons
YES	10,392	55,556	(222)	65,948
NO	1,039,230	55,555,556	(222,222)	56,594,786

Table 1

66,000 polygons; a number which would bring many systems to a grinding halt. Of course, if it did, the back plane would just be moved closer, or other parameters would be adjusted, generally resulting in a lower quality image. [Note: current commodity hardware graphics accelerators can process between 6 and 9 million triangles per second, which corresponds to between 200,000 and 300,000 triangles per frame at 30 fps.] Contrasting **66,000** to the polygon count of over **56 million** without a back plane makes the idea of maintaining interactive frame rates for scenes with no back plane seem absolutely ludicrous. However, there are several steps to take, and several characteristics left to exploit, that will narrow that gap between the polygon count with and without a back plane.

11. Variable Level of Detail

Perhaps the most effective step to take is to implement a variable level of detail system. Variable level of detail (VLOD) is based on the fact that the importance of a model's detail is inversely proportional to the model's distance from the viewpoint. In other words, it's fairly inefficient to render an Abrams M1 tank with 1000 polygons if its position is kilometers from the viewpoint.

The idea, then, is to adjust the number of polygons that constitute each model depending on the model's proximity (and other parameters such as speed and priority). A model extremely close to the viewpoint can be rendered in all its glory; a model some distance away can safely reduce its detail, and a model far away but still visible can abandon quite a lot of detail so long as it retains its defining shape and color.

The effect of such a scheme on a view volume with no back plane is *very* beneficial. Just how beneficial is dependent on the way in which the model detail is scaled. Obviously, the biggest benefit will be derived from models that have a very high maximum polygon count to start with. The potential decrease in polygons for such models is much higher. Also, the granularity of the scaling is important. If there are only two possible levels of detail, then the distance at which the detail shift occurs needs to be fairly close to reap the benefit. This could result in an unacceptable degradation of visual quality. The more levels of detail available for the models the better, although a tradeoff will likely exist between the granularity of detail and the

implementation overhead associated with establishing the detail for each frame.

11.1 Continuously Variable Level of Detail

The idea of variable level of detail has been around for awhile, and has seen incarnations using static levels, or discrete resolution. This causes a 'popping effect' when the model switches from one fixed level to the next. Only very recently has the VLOD concept been extended to use a continuous range of resolution, where the detail can be adjusted on a per-vertex basis (vertex-level resolution).

In January, 1998, the California-based Sven Technologies announced its Multi-resolution Geometry (MRG) technology. In March, 1999, Intel Corporation announced its Multi-resolution Mesh (MRM) technology. Examples of both MRG and MRM show models with over 2000 polygons scaling down to fewer than 100 polygons and maintaining excellent visual fidelity. [INTEL][SVEN]

The use of continuous VLOD technology certainly seems appropriate to the problem at hand, but its effect must be measured in terms of a reduction in polygons inside a view volume with no back plane. For the purpose of this investigation, only depth will be used as a factor for detail selection; model speed and priority will not be considered, but it is important to note that their inclusion would further lower the final polygon count. It will be convenient to create a new stage in the graphics pipeline for the VLOD algorithm. An updated pipeline that more specifically represents the pipe as it has been discussed is shown in Figure 9.

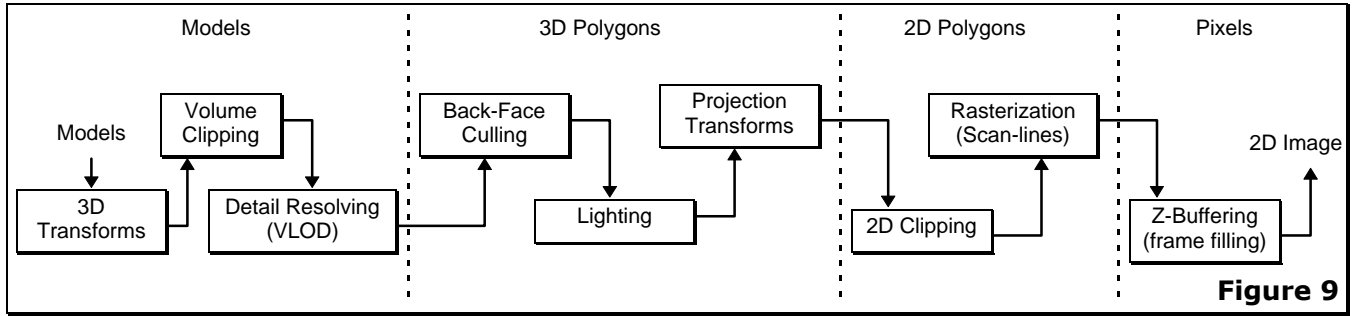


Figure 9

There are only a few parameters that determine the effect of a VLOD algorithm. The first of these is the depth difference between when maximum detail is used and when minimum detail is used. Certainly, models will have maximum detail when they have a very close proximity to the viewer. For the sake of calculations, the depth for maximum detail will be zero. That allows the depth difference to be expressed solely as the depth at which models resolve to the minimum level of detail. The depth of minimum detail will be denoted as \mathbf{d}_{ld} (d_{ld} for *lowest detail*). Any model further away than \mathbf{d}_{ld} will default to its lowest detail, which can save a large amount of model processing time during the detail resolving stage. A single depth comparison could then presumably prevent a huge percentage of the models from having to be tessellated.

This minimum detail depth might be viewed in the same light as the back clipping plane is today: effective but undesirable. A less egalitarian approach would be to scale the minimum depth individually for each model, based on the size of its bounding box, the idea being that a visually larger model would need to retain more of its detail at larger distances than would a smaller model. The algorithm assumed for this discussion, however, will adopt the simpler approach of a fixed \mathbf{d}_{ld} .

A very important thing to realize is that the model detail will truly be *continuously* variable, so that from one frame to the next only small changes in each model will be necessary (excluding scenarios like distance warping). In most interactive 3D applications, there is a tremendous amount of temporal coherence caused by the fact that both the view position and view direction experience very small deltas from one frame to the next. Continuous VLOD will benefit from this temporal coherence, as it will only be necessary to insert or remove relatively small numbers of vertices between frames. Another excellent benefit of implementing a VLOD stage into the pipe is that *all* of the models

will be scaled, not just the ones that would normally be behind the back clipping plane. The use of VLOD will reduce the polygon count for models in the entire range of depth values.

The other important aspect of a VLOD algorithm is the rate of detail reduction. A simple approach, and the one assumed for this discussion is to reduce detail linearly in proportion to distance. Alternatively, an exponential reduction rate would result in a lower average detail for the models, while still reserving the highest levels of detail for close distances. For linear reduction, the rate at which polygons are removed from models is expressed as the maximum polygon count minus the minimum polygon count, divided by the minimum detail distance. Of course, this rate will be different for each model, but as before, the average model can be used to simplify the situation. The equation is:

$$r_{detail} = \frac{(\beta_{max} - \beta_{min})}{d_{ld}} \frac{polygons}{vm} \quad \text{eq. 11}$$

β_{min} is the average polygon count for models at their lowest detail.

The VLOD effect can now be incorporated into equation 10.

$$\text{Model Polygons} = \frac{4}{3} c_{mod} \delta_{mod} \tan^2\left(\frac{\theta}{2}\right) \left(\beta_{avg} (d_{ld}^3 - p^3) + \beta_{min} (d_{max}^3 - d_{ld}^3) \right) \quad \text{eq. 12}$$

where $\beta_{avg} = \frac{1}{2}(\beta_{max} - \beta_{min}) + \beta_{min} = \frac{1}{2}(\beta_{max} + \beta_{min})$

The example from Table 1 is extended with the following new parameters:

$$\begin{aligned} \mathbf{d}_{ld} &= 500 & vm \\ \beta_{min} &= 50 & \text{polygons/model} \end{aligned}$$

The new totals are shown in Table 2.

The previous difference between the two model polygon counts was 55.5 million. The new difference from Table 2 is about 5.5 million polygons. The count difference is a factor of 100, down from 858. That's a remarkable improvement, but the difference is obviously still 5.5 million too much, if equal polygon

Polygon Totals incorporating Model VLOD				
Back Plane present?	Terrain Polygons	Model Polygons	(Models)	Total Polygons
YES	10,392	55,556	(222)	65,948
NO	1,039,230	5,558,681	(222,222)	6,597,911

Table 2

counts are the measure of success (terrain is being ignored for the moment). There is one more way to tighten the grip on the set of model polygons.

12. Scaling to insignificance

The idea behind VLOD is that as models get further and further away from the viewer, their detail becomes less and less important. The reason that the detail becomes less important is that the models cover an increasingly smaller area in the final 2D image as their distance increases. If this idea is extended to its logical conclusion, it becomes obvious that for models beyond a certain distance, not only does their detail become unimportant, but whether or not they are rendered at all is irrelevant. This happens only when perspective projection is used in the rendering process, as opposed to parallel projection.

Perspective projection creates a virtual vanishing point, which makes objects scale inversely proportional to their distance from the viewpoint (Figure 2). Since the final 2D image is a discrete pixelmap, the smallest individual element that can be rendered is a single pixel. Once a model has been moved far enough from the viewpoint that the size of its projected image is less than a single pixel, there is no sense in processing the model. It has been scaled into insignificance. With no back clipping plane, it stands to reason that many model polygons can nevertheless be eliminated from the pipe because their parent model is too far away to be of significance.

The depth at which a model projects to an area less than one pixel obviously depends on the virtual size of the model itself. The major objection to a back clipping plane is that it eliminates models and polygons that did *not* scale into insignificance for a particular frame. In order to avoid this, it is necessary to evaluate each model on an individual basis. This can be accomplished by creating a bounding box for each model.

There is already a need to have a bounding box for each model, for the purpose of 3D volume clipping. For each frame, the square of the distance from the viewpoint to the center of the

bounding box is calculated (to avoid a costly square root) and used with the smallest dimension of the bounding box to see if the projected area would be less than one pixel. Thus, the larger a model is, the further away it can be and still get rendered (with fewer polygons, thanks to VLOD). The smallest dimension of the bounding box is used so that the algorithm errs in favor of performance. If models are used which have drastically different dimensions in each of the three axes, a more complicated approach can be used to select which dimension to scale.

An attractive feature of this test is that it is performed at the model level. The number of models in a scene is much smaller than the number of polygons, and anything that eliminates an entire model with little overhead is almost certainly worth implementing. It is for this very reason that the 3D volume clipping is performed first on models as single entities using their bounding boxes. Only models whose bounding boxes intersect the clipping planes have their constituent polygons evaluated for volume clipping individually. For the same reason, the test for scaling to insignificance should occur very early in the pipe, before the 3D transformations and VLOD stages. This way, only those models that have a high enough size/distance ratio get transformed and tested against the FOV. Then, only those models that are inside the frustum get processed by the VLOD algorithm, and only the polygons exiting the detail resolving stage get tested for back-face culling. After the culling stage, the surviving list of primitives has been drastically reduced, and the only polygons remaining in the pipe that won't ultimately be seen are those that are occluded. The updated pipeline is shown in Figure 10.

12.1 Effect of Scaling to Insignificance

In order to determine how much the new scaling test will reduce the polygon count for the example it is necessary to estimate the different types and sizes of models in the hypothetical scene. The parameters used up to this point have been indicative of a large outdoor environment with a moderate density of high-detail models. For the sake of argument, it will now take on the

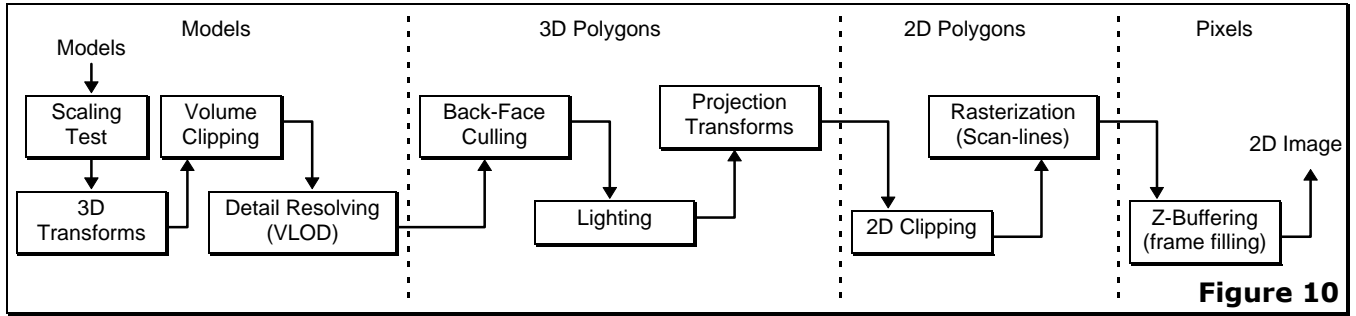


Figure 10

more specific form of a tactical battlefield environment. The types of models within the scene are assumed to be military in nature. The two major classes of models are vehicles and personnel. This division lends itself well to the task of incorporating the effects of the scaling test into equation 12. The percentage of models that are vehicles is designated as \mathbf{m}_{veh} , and the percentage that are infantry is \mathbf{m}_{inf} . Since only two types of models are considered, the sum of those percentages should equal one. The two separate parameters will be used instead of a ratio to facilitate the inclusion of additional model types if desired. Typical values for \mathbf{m}_{veh} and \mathbf{m}_{inf} are about 10% and 90%, respectively. [MCCOY] The vehicles and infantry units will each have a bounding box (a cube for simplicity) with dimensions of \mathbf{S}_{veh} and \mathbf{S}_{inf} .

In order to calculate the distance at which the projection of a model is less than one pixel, the resolution of the final viewport window on the screen must be known. Again for simplicity, a square viewport window will be assumed. Its dimension (in pixels) is denoted as \mathbf{S}_{vp} . The dimension (in 3D coordinate units) of the projection port (that part of the projection plane formed by its intersections with the clipping planes) must also be known. It is expressed as \mathbf{S}_{pp} and has the following equation:

$$\mathbf{S}_{pp} = 2p \tan\left(\frac{\Theta}{2}\right) \quad \text{eq. 13}$$

Again, \mathbf{p} is the distance from the center of projection to the projection plane, and Θ is the angle of the field of view. The equation for determining when a model projects less than a pixel is shown in equation 14:

$$1 > \left(\frac{p \mathbf{S}_{mod}}{d_{mod}}\right) \left(\frac{\mathbf{S}_{vp}}{\mathbf{S}_{pp}}\right) \Rightarrow \\ (d_{mod})^2 > (\mathbf{S}_{mod})^2 \left(\frac{p^2 \mathbf{S}_{vp}^2}{\mathbf{S}_{pp}^2}\right) \quad \text{eq. 14}$$

\mathbf{S}_{mod} is the dimension of a model's bounding cube, and \mathbf{d}_{mod} is the distance from the viewpoint to the center of the bounding cube. If

equation 14 evaluates true, then \mathbf{d}_{mod} is large enough that the model will project an area that covers less than one pixel in the viewport window. This minimum distance for a model will be expressed as \mathbf{d}_{min} .

The next step is to calculate how many of the models would fail this test, and thus how much the polygon count can be reduced. Obviously, this is very scene dependent, and extremely variable within the scene itself. The number of models that fail could be all or none. Keeping with the previous calculations, though, the average scenario is used, meaning that the models are assumed to be evenly distributed within the scene. The straightforward solution is to add up the number of model polygons in the frustum volume for each type of model. Instead of a single model density, there is now one for each model type.

$$c_{mod} \sum_{i=1}^n \left(V_{d_{ld_i}} \delta_{mod_i} \beta_{avg_i} + V_{d_{min_i}} \delta_{mod_i} \beta_{min_i} \right) \quad \text{eq. 15}$$

Where:

n = the number of different model types;

$$V_{d_{ld}} = \frac{4}{3} \left(d_{ld}^3 - p^3 \right) \tan^2\left(\frac{\Theta}{2}\right);$$

$$V_{d_{min}} = \frac{4}{3} \left(d_{min}^3 - d_{ld}^3 \right) \tan^2\left(\frac{\Theta}{2}\right);$$

$$d_{min} = \mathbf{S}_{mod} \left(\frac{p \mathbf{S}_{vp}}{\mathbf{S}_{pp}} \right);$$

$$\beta_{avg} = \frac{1}{2} (\beta_{max} + \beta_{min});$$

The above equations assume that $\mathbf{d}_{min} > \mathbf{d}_{ld}$ and that $\mathbf{d}_{max} > \mathbf{d}_{min}$. Note that it is very easy to allow \mathbf{d}_{ld} , β_{max} , and β_{min} to vary for each model type, although that won't be done in the following calculations. Another iteration of the example can be executed, incorporating the new parameters.

$$\mathbf{m}_{veh} = 10\%$$

$$\mathbf{m}_{inf} = 90\%$$

$$\begin{aligned} S_{veh} &= 5 \quad \text{vm} \\ S_{inf} &= 1 \quad \text{vm} \\ S_{vp} &= 1000 \quad \text{pixels} \end{aligned}$$

Using these parameter values, along with the previous values for all other parameters, the distance at which infantry models scale into insignificance is **866** vm. For vehicle models, $d_{min} = 4330$ vm. Notice that d_{min} for the infantry models is actually less than the distance to the clipping plane ($866 < 1000$). This means that fewer infantry models will be processed when the scaling test is incorporated. This benefits performance. Conversely, d_{min} for the vehicle models is well beyond the clipping plane. This verifies that vehicles which should be visible in the image are not being rendered when the back plane is used. This demonstrates the compromise associated with setting the distance of the back plane. As the distance increases, more of the smaller models get included than is necessary, degrading performance. As the distance decreases, more of the larger models get excluded than is necessary, degrading fidelity. The updated polygon counts are shown in Table 3.

The number of model polygons that must be processed when the back clipping plane is absent is now *less* than the number when the plane is present. There is a **7.3%** decrease. This is especially impressive considering that **8.7** times as many models are being rendered (1,934 vs. 222). All of these are models that are inside the FOV and visible, except for occlusion.

That concludes the effort to reduce the number of model polygons. Up to this point, the terrain polygons have been ignored. Can similar reductions be achieved in their numbers as well?

13. VLOD for Terrain

The first step to reduce the terrain polygons is to apply a VLOD algorithm to the terrain mesh, in much the same fashion as was done for the models. Terrain tessellation is a topic that seems to have garnered much less attention from researchers than has model tessellation. The reason is, presumably, that essentially

every 3D application uses models, while only a small subset uses a terrain mesh. Also, the back clipping plane is used almost indiscriminately to lower the polygon count, and if that isn't sufficient, the resolution of the terrain grid can just be lowered.

Lowering the terrain grid resolution (increasing the effective grid increments) actually resamples the terrain data, which obviously results in the loss of terrain detail; a less than ideal solution. A more intelligent approach is to realize that, as with models, certain circumstances make *some* of the terrain samples unimportant. A variable level of detail algorithm can assess the relevance of each sample and either eliminate or retain the sample, on a frame by frame basis. This would lower the grid resolution only over the areas of the mesh where it was acceptable, where acceptability was derived from scene parameters and threshold variables set by the user or application.

It is likely that the original resolution of the terrain grid would represent the maximum detail level, and samples would only be either removed or retained. However, a more ambitious VLOD algorithm might also have the ability to interpolate between samples for the purpose of inserting additional points in the mesh, effectively increasing the detail beyond the original level for certain areas. This would involve extra computational overhead, but it might be necessary where the original terrain detail is too coarse.

One method of applying VLOD to a terrain is to use a simple depth-based reduction, where the resolution is scaled purely as a function of the distance to the viewpoint. A more rigorous method is the one used by Lindstrom et al., which is the VLOD algorithm upon which the example will be based. [LKRHFT] This algorithm takes into account the angle of incidence between the view vector and the terrain polygons, as well as the slope of adjacent polygons to determine if they can be merged into larger polygons. The terrain mesh is treated hierarchically, and the reduction process is recursive, maximizing the polygon reduction for each frame. As with the model VLOD algorithms, the state of mesh reduction from each previous frame

Polygon Totals incorporating Scaling Test (d_{min})				
Back Plane present?	Terrain Polygons	Model Polygons	(Models)	Total Polygons
YES	10,392	55,556	(222)	65,948
NO	1,039,230	51,478	(1934)	1,090,709

Table 3

is used as the basis for the next frame, so the process is incremental in nature, greatly reducing the computation time for each frame.

The effectiveness of the terrain VLOD algorithm described by Lindstrom et al. is somewhat dependent on the characteristics of the terrain being rendered. Valleys, ridges, mountains, plains, etc. all affect the amount of polygon reduction that occurs. Results from [LKRHFT] are derived from a terrain scene that encompasses a wide variety of those geographical features in order to amortize their effect. Their scene has a 60° FOV, a 2x2 meter terrain resolution, and roughly 13 million polygons inside the view frustum. In order to adopt the reduction ratio obtained under those conditions, the hypothetical terrain used for the example thus far will be assumed to have similar geographic features. It is also necessary to increase the terrain polygon count to approximately 13 million. This is accomplished by changing the grid resolution to 3x3 meters, which results in 12.8 million terrain polygons. The difference in grid resolutions is inconsequential as long as the polygon counts are roughly equivalent. The 10x10 meter resolution will be retained for the control case with a back clipping plane at 1000m.

The VLOD algorithm is parameterized by an error threshold value, λ , which affects the fidelity of the final image. The threshold represents the maximum visible geometric (linear) error (in pixels) allowed between any original projected polygon pair and the merged projected polygon. [LKRHFT, 4] To illustrate, a value of $\lambda=0.0$ results in only coplanar polygons being merged, $\lambda=1.0$ results in reduction with virtually no perceptible changes from frame to frame, and $\lambda=4.0$ would cause excellent reductions while still being extremely faithful to terrain features. [LKRHFT, 8] For this example, $\lambda=2.0$ is used in order to make the example further coincide with the conditions of the results described by Lindstrom et. al. For $\lambda=2.0$, the reduction ratio is approximately 1300:1 (averaged over 3,000 frames with varying viewpoints). [LKRHFT, 9] Applying this reduction to the terrain

count with no back clipping plane (1,039,230) results in a new terrain polygon count of **8,882** polygons! The new totals are summarized in Table 4. After all the dust has settled, the effect of eliminating the back clipping plane is an **8.5% decrease** in the number of polygons. That decrease even comes after changing the terrain resolution from 10x10 to 3x3m for the one case!

14. Results

Of course, the **8.5%** decrease in polygons isn't the whole story. The benefits of eliminating the back plane are numerous! First of all (based on the example) there is a **771%** increase in the number of models that are processed. The most dramatic advantage, though, is the fact that the effective viewing distance can be increased by at least a factor of **10** (d_{max}/d). The overall effect of these benefits is that when something is supposed to be visible in a rendered image, it is.

The disadvantages of the steps taken to allow this vast improvement in visual fidelity come from computational overhead associated with each of the new methods introduced. The scaling test can be done with only 5 additions, 5 multiplications, and one comparison per model (3 additions, 3 multiplications for the distance formula, and 2 more multiplications in order to do the comparison). The information on the model VLOD algorithms is scarce, but both the MRM and MRG technologies imply a negligible overall impact on rendering times, considering the time savings gained from *not* processing the polygons that are removed. The same can be said for the terrain VLOD algorithm developed by Lindstrom et al.

The important thing to note is that all of the algorithms discussed are implemented as stages at the front end of the graphics pipeline. Their benefits are then also measured in terms of reduced load on the very costly stages at the other end of the pipe. Typically, lighting, rasterization, and Z-buffering are the potential bottlenecks in the pipe. These stages deal on the pixel level (except lighting which is just plain complex), and so the extra overhead at the model and polygon levels is acceptable.

Polygon Totals incorporating Everything				
Back Plane present?	Terrain Polygons	Model Polygons	(Models)	Total Polygons
YES	10,392	55,556	(222)	65,948
NO	8,882	51,478	(1934)	60,360

Table 4

Recall that Z-buffering was identified as the only appropriate visible surface determination algorithm for a scene with no back clipping plane. It bears repeating that the reason Z-buffering is the best choice is because it operates at the pixel level. It is an image precision algorithm, and so is placed near the end of the pipeline, once all the objects have been resolved to pixels. Stages at that end of the pipe can basically be oblivious to the fact that the back clipping plane is gone, and that the early stages are dealing with significantly more objects. Any other VSD algorithm (any object-precision algorithm), must also deal with the increased number of objects. From the example, that can amount to 8.7 times as many models. (A terrain grid is inherently depth sorted, and optionally can be drawn using a derivative of the painter's algorithm if it's cheaper to do so.) So it is the fact that these polygon reduction methods are applied early in the pipeline that makes Z-buffering not only feasible under such circumstances, but almost unavoidable.

15. Recommendations

This investigation has centered around a mathematical model which seems to do a fair job of determining the effect of removing the back clipping plane, offset by the application of several polygon reduction methods. It would be most interesting to actually measure the visual quality, effectiveness, and performance changes of the scenario within the context of an actual 3D application. To do so would require an application that is designed to handle large outdoor scenes, uses Z-buffering as the primary VSD algorithm, and for which the source code is freely available, well documented, and modularized.

The algorithm for scaling to insignificance would be simple to implement, requiring only a single parameter for each model (the size of the bounding cube). The terrain VLOD algorithm presented in [LKRHFT] is described in enough detail to allow implementation. The model VLOD algorithms discussed here are both privately licensed technologies, preventing an easy and low cost implementation for the purpose of experimentation. If it could be managed, however, a graphics engine with full implementations of these methods would undoubtedly reveal additional advantages and disadvantages of not only eliminating the back plane, but of the polygon reduction methods as well.

16. Conclusions

The use of a back clipping plane for dynamic 3D scenes has been, and continues to be, a very simple, efficient, effective, and common technique for improving performance. Its use imposes a restriction on the realism for rendered scenes, however. True visibility is sacrificed as an accepted consequence for the level of performance needed to maintain real-time 3D animation. Image fidelity has had many other features sacrificed in the name of performance. These includes things such as a true third dimension, colored lighting, localized lighting effects, true (32 bit) color, multiple resolution texture mapping, transparency, etc.. The last few years, however, have seen each of these features finally incorporated into real-time 3D graphics.

The purpose of this investigation was to determine whether or not true visibility could be added to that list of reclaimed features. The answer seems to be a resounding 'probably'. There now exist several clever but intuitive methods for offsetting the penalty associated with eliminating the back clipping plane. The offset can easily be 100%, although not always, and so situations where true visibility is a priority must be evaluated on an individual basis. The example calculations provided were derived from very specific parameter values and resulted in an 8.5% decrease in the primitive list. These values may or may not be representative of those for any given application.

Moderate changes in the values used to calculate the polygon counts can produce a decrease or increase in the polygon total of greater magnitude that was obtained in the example. The intent of the investigation was to be faithful to the derived model and let the chips fall where they may. The scene parameter values that were used were chosen because they are thought to be typical. Figures 11-13 demonstrate the relationships of some of these parameters to the polygon counts.

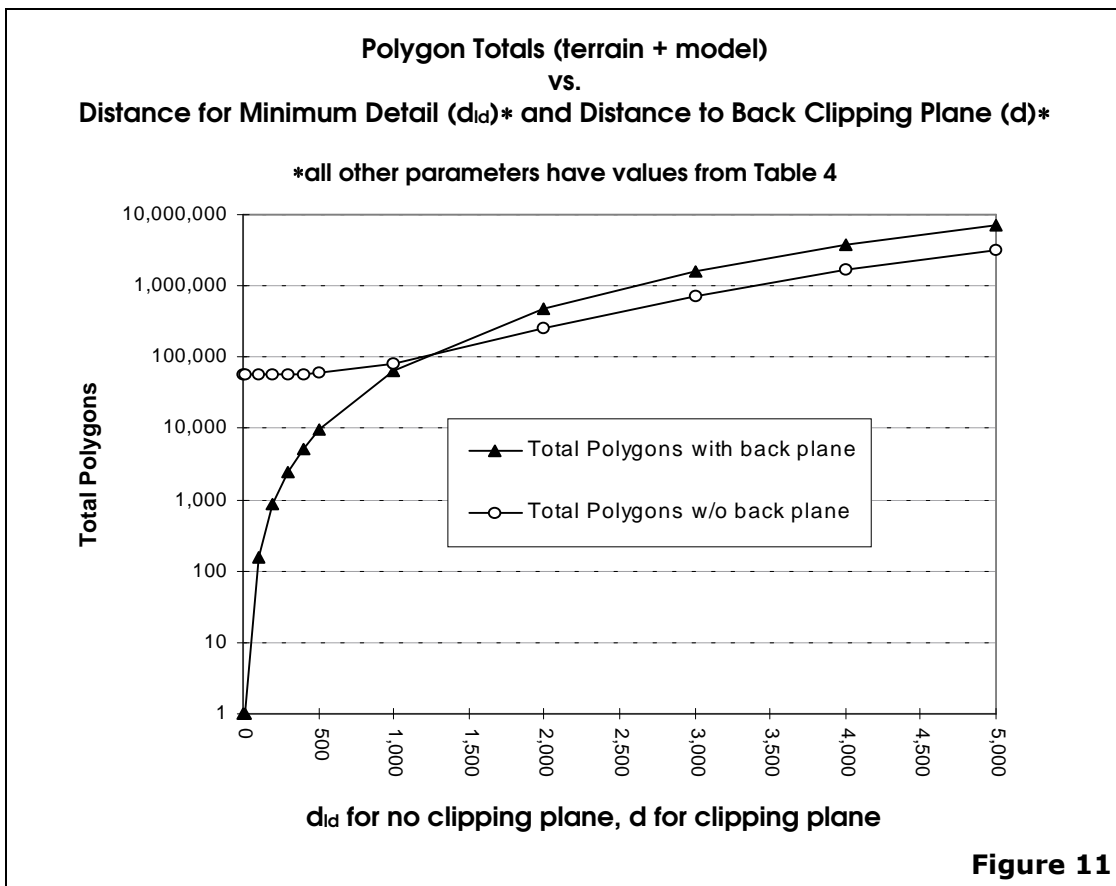
- Figure 11 shows the total polygon count when the distance to the back clipping plane is increased along with the distance for minimum detail.
- Figure 12 shows the model polygon count as β_{\max} is increased, for different methods of choosing β_{\min} .
- Lastly, Figure 13 shows terrain polygon counts for different combinations of terrain reduction ratio and grid

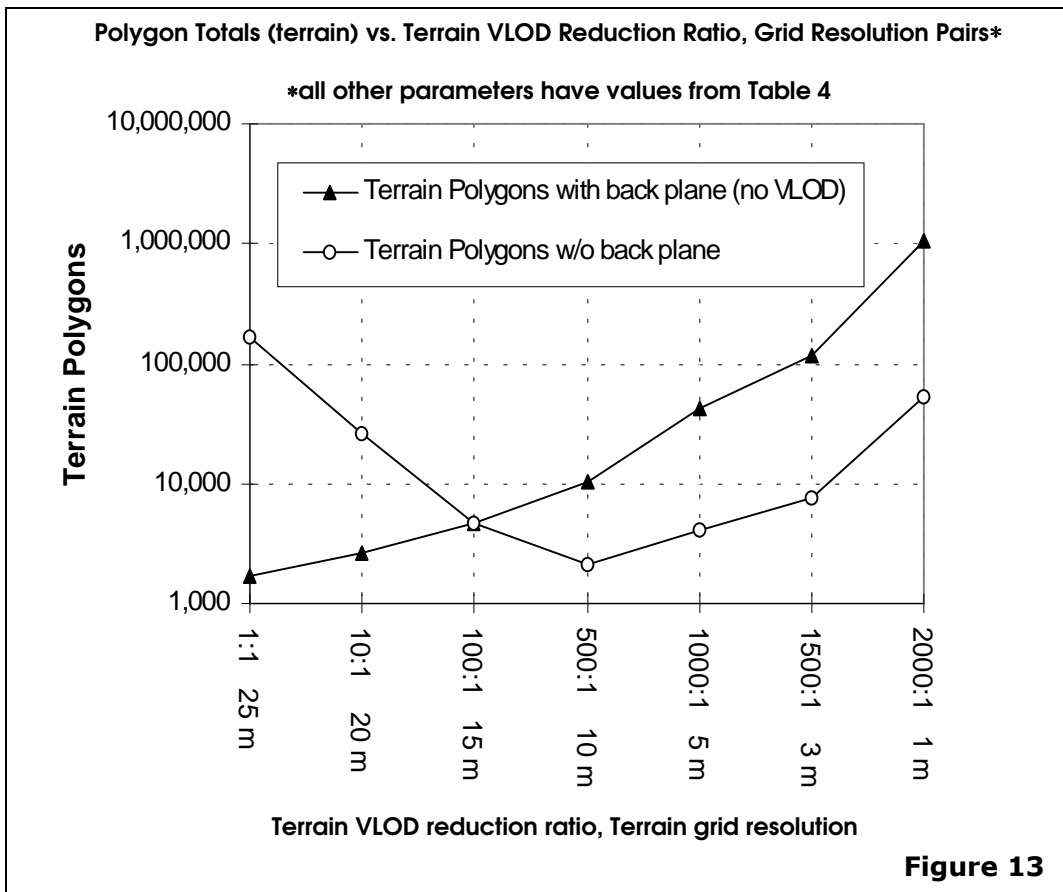
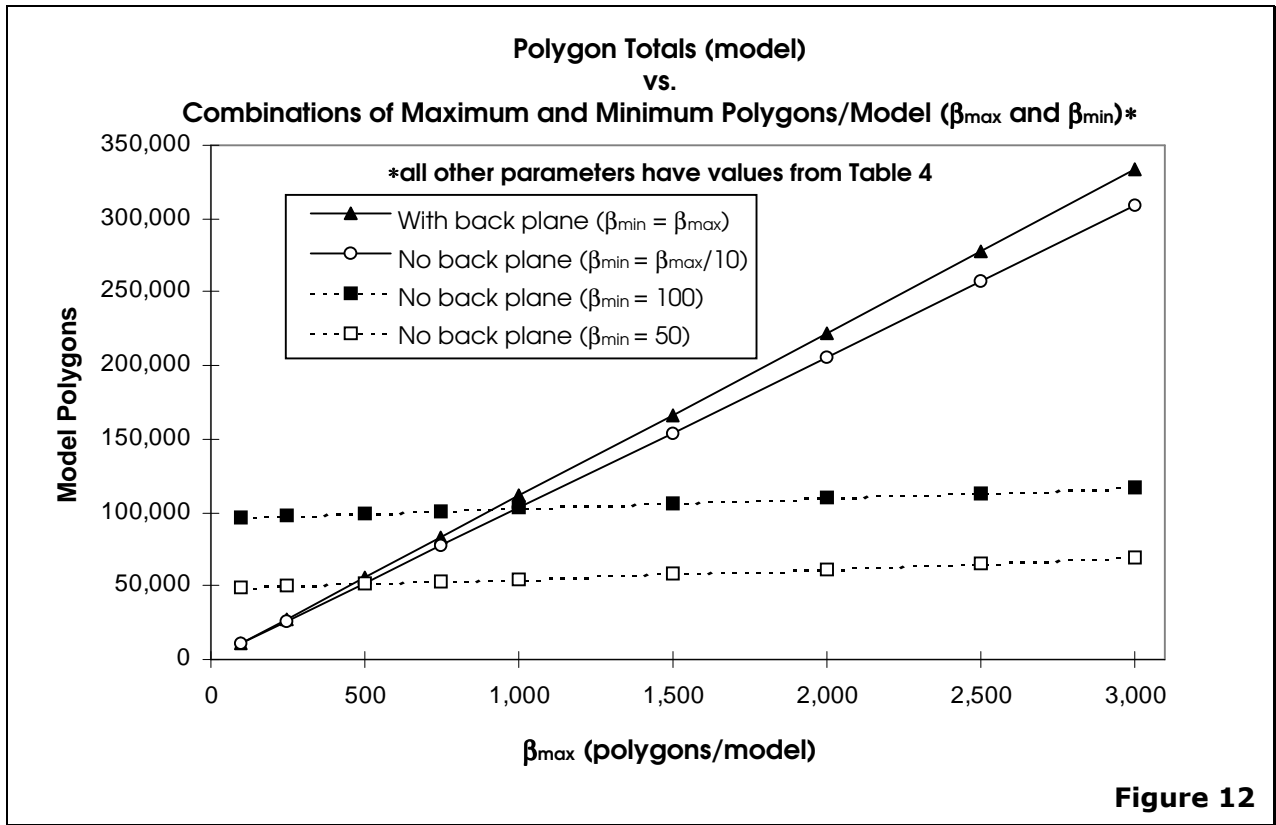
resolution (approximate extrapolations from [LKRHFT]).

Suffice to say that the procedure and the model are very susceptible to scene dependencies. Like most endeavors in computer graphics analysis, a certain amount of risk is assumed when addressing anything as 'average'.

Ultimately, though, it appears that the back clipping plane is just about obsolete. The advent of affordable, extremely powerful, commodity 3D graphics processors has ushered in a new era of computer graphics. Today's graphics accelerator

hardware is slowly absorbing the responsibilities for one stage after another of the rendering pipeline. This has freed up system resources and processor time that were normally committed to low level activities like rasterizing and double-buffering. These liberated resources facilitate the addition of newer, high level algorithms such as the polygon reductions methods discussed here. The door seems to be open for applications to forego the back clipping plane and achieve true visibility rendering.





17. References & Suggested Reading

- [ABRASH] *Zen of Graphics Programming*, Michael Abrash; The Coriolis Group, Inc., 1996; Scottsdale, AZ; **ISBN 1-883577-89-6**
- [ANGEL] *Interactive Computer Graphics*, Edward Angel; Addison-Wesley Publishing Company, 1997; Berkeley, CA; **ISBN 0-201-85571-2**
- [DNW] *OpenGL Programming Guide, Second Edition*, Tom Davis, Jackie Neider, and Mason Woo, OpenGL Architecture Review Board (ARB); Addison-Wesley Developers Press, 1997; Berkeley, CA; **ISBN 0-201-46138-2**
- [FDFH] *Computer Graphics: Principles and Practice, Second Edition*, James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughe; Addison-Wesley Publishing Company, 1996; Berkeley, CA; **ISBN 0-201-84840-6**
- [FOSNER] *OpenGL Programming for Windows 95 and Windows NT*, Ron Fosner; Addison-Wesley Developers Press, 1997; Berkeley, CA; **ISBN 0-201-40709-4**
- [GLASSNER] *Graphics Gems*, Andrew S. Glassner, Editor, Xerox Palo Alto Research Center; AP Professional, 1990; San Diego, CA; **ISBN 0-12-286166-3**
- [HOFF] *Faster 3D Game Graphics by Not Drawing What is Not Seen*, Kenneth E. Hoff III; ACM Crossroads, 1998
- [INTEL] Multi-resolution Mesh, Intel 3D Software Technologies; Intel Architecture Labs, Intel Inc., 1999; <http://www.intel.com/ial/3dsoftware/mrm.htm>
- [LH] *Calculus with Analytic Geometry*, Roland E. Larson, Robert P. Hostetler; D.C. Heath and Co., 1986; Lexington, MA; **ISBN 0-669-09568-0**
- [LKRHFT] *Real-Time, Continuous Level of Detail Rendering of Height Fields*, Lindstrom, P., Koller, D., Ribarsky, W., Hodges, L. F., Faust, N., and Turner, G. A.; Proceedings of SIGGRAPH 96, August 1996, pp. 109-118
- [MCCOY] Discussions about distributed system simulations, Donald Hue McCoy; New Mexico State University, 1999; Las Cruces, NM
- [PETZOLD] *Programming Windows 95*, Charles Petzold; Microsoft Press, 1996; Redmond, WA; **ISBN 1-55615-676-6**
- [SVEN] Multi-resolution Geometry; Sven Technology, Inc., 1999; <http://www.sven-tech.com/products/mrg/>
- [ZMHH] *Visibility Culling using Hierarchical Occlusion Maps*, Hansong Zhang, Dinesh Manocha, Kenneth E. Hoff III, and Tom Hudson; Proceedings of ACM SIGGRAPH, 1997