Oblique View Frustum Depth Projection and Clipping

Eric Lengyel
Terathon Software
lengyel@terathon.com

Abstract. Several 3D rendering techniques have been developed in which part of the final image is the result of rendering from a virtual camera whose position in the scene differs from that of the primary camera. In these situations, there is usually a planar surface, such as the reflecting plane of a mirror, that can be considered the physical boundary of the recursively rendered image. In order to avoid artifacts that can arise when rendered geometry penetrates the boundary plane from the perspective of the virtual camera, an additional clipping plane must be added to the standard six-sided view frustum. However, many 3D graphics processors cannot support an extra clipping plane natively, or require that vertex and fragment shaders be augmented to explicitly perform the additional clipping operation.

This paper discusses a technique that modifies the projection matrix in such a way that the conventional near plane of the view frustum is repositioned to serve as the generally oblique boundary clipping plane. Doing so avoids the performance penalty and burden of developing multiple shaders associated with user-defined clipping planes by keeping the total number of clipping planes at six. The near plane is moved without affecting the four side planes, but the conventional far plane is inescapably destroyed. We analyze the effect on the far plane as well as the related impact on depth buffer precision and present a method for constructing the optimal oblique view frustum.

1. Introduction

Several techniques have been developed to render 3D images containing elements that are inherently recursive in nature. Some examples are mirrors that reflect their immediate surroundings, portals through which a remote region of the scene can be viewed, and water surfaces through which refractive transparency is applied. Each of these situations requires that part of the scene be rendered from the perspective of some virtual camera whose position and orientation are calculated using certain rules that take into account the position of the primary camera through which the user is looking. For example, the image visible in a mirror is rendered using a camera that is the reflection of the primary camera through the plane of the mirror.

Published in *Journal of Game Development*, Vol. 1, No. 2 (March 2005), pp. 5–16.

Once such a component of an image is rendered through a virtual camera, it is usually treated as a geometrically planar object when rendering from the perspective of the primary camera. The plane chosen to represent the image is simply the plane that naturally separates the image from the rest of the environment, such as the plane of a mirror, portal, or water surface. In the process of rendering from a virtual camera, it is possible that geometry lies closer to the camera than the plane representing the surface of the mirror, portal, etc. If such geometry is rendered, it can lead to unwanted artifacts in the final image.

The simplest solution to this problem is to enable a user-defined clipping plane to truncate all geometry at the surface. Unfortunately, older GPUs do not support user-defined clipping planes in hardware and must resort to a software-based vertex processing path when they are enabled. Later GPUs do support generalized user-defined clipping operations, but using them requires that the vertex or fragment programs be modified—a task that may not be convenient since it necessitates two versions of each program be kept around to render a particular geometry.

This article presents an alternative solution that exploits the clipping planes that already exist for every rendered scene. Normally, every geometric primitive is clipped to the six sides of the view frustum: four side planes, a near plane, and a far plane. Adding a seventh clipping plane that represents the surface through which we are looking almost always results in a redundancy with the near plane, since we are now clipping against a plane that slices through the view frustum further away from the camera. Thus, our strategy is to modify the projection matrix in such a way that the conventional near plane is repositioned to coincide with the additional clipping plane, which is generally oblique to the ordinary view frustum. Since we are still clipping only against six planes, such a modification gives us our desired result at absolutely no performance cost. Furthermore, this technique can be applied to any projection matrix, including the conventional perspective and orthographic projections as well as the infinite projection matrix used by stencil shadow volume algorithms.

Moving the near plane does not affect the four side planes of the view frustum, but such an action generally has an unintuitive and detrimental effect on the conventional far plane which is not addressed by an earlier ad hoc implementation of the technique described herein [1]. The associated effect on the mapping of points to depth values is equally detrimental. Remedies to this undesirable consequence are limited, but we analyze the problem in this paper and provide a method for constructing the optimal oblique view frustum for any given near plane.

2. Background

The technique described in this paper is presented in the context of the OpenGL architecture, and we thus adopt the mathematical representations and coordinate systems used by the OpenGL library. For other graphics systems, such as Direct3D, in which the camera-space and clip-space coordinate systems differ slightly, the derivation in the next section can be reapplied to obtain an equivalent result.

A plane **C** is mathematically represented by a four-dimensional vector of the form

$$\mathbf{C} = \langle N_x, N_y, N_z, -\mathbf{N} \cdot \mathbf{Q} \rangle, \tag{1}$$

where **N** is the normal vector pointing away from the front side of the plane, and **Q** is any point lying in the plane itself. A homogeneous point $\mathbf{P} = \langle P_x, P_y, P_z, P_w \rangle$ lies in the plane if and only if the four-dimensional dot product $\mathbf{C} \cdot \mathbf{P}$ is zero. If the normal vector **N** of a plane **C** is unit length and $P_w = 1$, then the dot product $\mathbf{C} \cdot \mathbf{P}$ measures the signed perpendicular distance from the point **P** to the plane **C**.

A plane is a covariant vector, and therefore must be transformed from one coordinate system to another using the inverse transpose of the matrix that transforms ordinary points (which are contravariant vectors). This is particularly important when transforming planes with the projection matrix since it is generally nonorthogonal. Given a camera-space point **P** and a camera-space plane **C** represented by four-component column vectors, the projection matrix **M** produces a clip-space point **P′** and a clip-space plane **C′** as follows.

$$\begin{aligned} \mathbf{P}' &= \mathbf{M}\mathbf{P} \\ \mathbf{C}' &= \left(\mathbf{M}^{-1}\right)^{\mathrm{T}}\mathbf{C} \end{aligned} \tag{2}$$

Inverting these equations allows us to transform from clip space to camera space.

Recall that in OpenGL camera space, the camera lies at the origin and points in the $-z$ direction, as shown in Figure 1. To complete a right-handed coordinate system, the x-axis points to the right, and the y-axis points upward. Vertices are normally transformed from whatever space in which they are specified into camera space by the model-view matrix. In this paper, we do not worry about the model-view matrix and assume that vertex positions are specified in camera space.

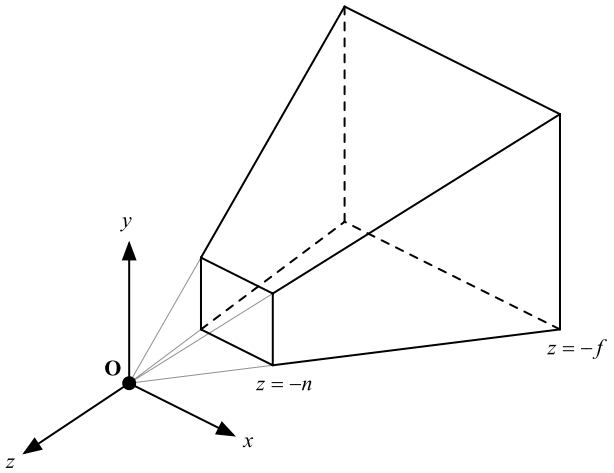

Figure 1. Camera space and the standard view frustum. The near and far planes are perpendicular to the z-axis and lie at the distances n and f from the camera, respectively.

The standard view frustum is the six-sided truncated pyramid that encloses the volume of space visible to the camera. As shown in Figure 1, it is bounded by four side planes representing the four edges of the viewport, a near plane at $z = -n$, and a far plane at $z = -f$. The near and far planes are normally perpendicular to the camera's viewing direction, but our modifications to the projection matrix will move these two planes and change the fundamental shape of the view frustum.

The projection matrix transforms vertices from camera space to homogeneous clip space. In homogeneous clip space, a four-dimensional point $\langle x, y, z, w \rangle$ lies inside the projection of the camera-space view frustum if the following conditions are satisfied.

$$\begin{aligned} -w \le x \le w \\ -w \le y \le w \\ -w \le z \le w \end{aligned} \tag{3}$$

Performing the perspective division by the w-coordinate moves points into normalized device coordinates, where each of the coordinates of a point in the view frustum lies in the interval $[-1, +1]$. Our goal is to modify the projection matrix so that points lying on a given arbitrary plane have a z-coordinate of -1 in normalized device coordinates. We avoid examining any particular form of the projection matrix and require only that it be invertible. This allows our results to be applied to arbitrary projection matrices which may have already undergone some kind of modification from the standard forms.

Figure 2 shows the x and z components of a three-dimensional slice of the four-dimensional homogeneous clip space. Within this slice, the w-coordinate of every point is 1, and the projection of the view frustum described by Equation (3) is bounded by six planes forming a cube. The w-coordinate of each plane is 1, and exactly one of the x-, y-, and z-coordinates is ± 1, as shown in Table 1. Given an arbitrary projection matrix $\mathbf{M}$, the inverse of Equation (2) can be used to map these planes into camera space. This produces the remarkably simple formulas listed in Table 1, in which each camera-space plane is expressed as a sum or difference of two rows of the projection matrix [2].

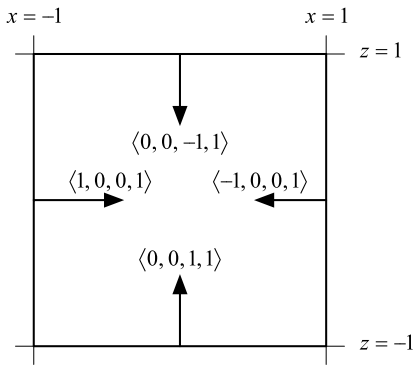

Figure 2. The x and z components of a three-dimensional slice of homogeneous clip space at $w = 1$ and four of the six clipping planes that form a cube in this space.

Table 1. Clip-space and camera-space view frustum planes. The matrix **M** represents the projection matrix that transforms points from camera space to clip space. The notation $\mathbf{M}_i$ represents the i-th row of the matrix **M**.

Frustum Plane	Clip-space Coordinates	Camera-space Coordinates
Near	$\langle 0, 0, 1, 1 \rangle$	$\mathbf{M}_4 + \mathbf{M}_3$
Far	$\langle 0, 0, -1, 1 \rangle$	$\mathbf{M}_4 - \mathbf{M}_3$
Left	$\langle 1, 0, 0, 1 \rangle$	$\mathbf{M}_4 + \mathbf{M}_1$
Right	$\langle -1, 0, 0, 1 \rangle$	$\mathbf{M}_4 - \mathbf{M}_1$
Bottom	$\langle 0, 1, 0, 1 \rangle$	$\mathbf{M}_4 + \mathbf{M}_2$
Top	$\langle 0, -1, 0, 1 \rangle$	$\mathbf{M}_4 - \mathbf{M}_2$

3. Clipping Plane Modification

Let $\mathbf{C} = \langle C_x, C_y, C_z, C_w \rangle$ be the plane shown in Figure 3, having coordinates specified in camera space, to which we would like to clip our geometry. The camera should lie on the negative side of the plane, so we can assume that $C_w < 0$. The plane **C** will replace the ordinary near plane of the view frustum. As shown in Table 1, the camera-space near plane is given by the sum of the last two rows of the projection matrix **M**, so we must somehow satisfy

$$\mathbf{C} = \mathbf{M}_4 + \mathbf{M}_3. \tag{4}$$

We cannot modify the fourth row of the projection matrix because perspective projections use it to move the negation of the z-coordinate into the w-coordinate, and this is necessary for perspective-correct interpolation of vertex attributes such as texture coordinates. Thus, we are left with no choice but to replace the third row of the projection matrix with

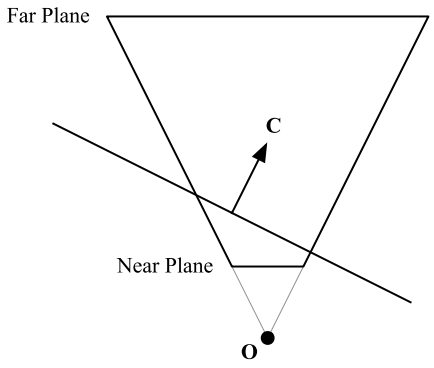

Figure 3. The near plane of the view frustum is replaced with the arbitrary plane **C**.

$$\mathbf{M}_3' = \mathbf{C} - \mathbf{M}_4. \tag{5}$$

After making the replacement shown in Equation (5), the far plane **F** of the view frustum becomes

$$\begin{aligned}\mathbf{F} &= \mathbf{M}_4 - \mathbf{M}_3' \\ &= 2\mathbf{M}_4 - \mathbf{C}.\end{aligned} \tag{6}$$

This fact presents a significant problem for perspective projections. A perspective projection matrix must have a fourth row given by $\mathbf{M}_4 = \langle 0, 0, -1, 0 \rangle$ so that the clip-space w-coordinate receives the negation of the camera-space z-coordinate. As a consequence, the near plane and far plane are no longer parallel if either C_x or C_y is nonzero. This is extremely unintuitive and results in a view frustum having a very undesirable shape. By observing that any point $\mathbf{P} = \langle x, y, 0, w \rangle$ for which $\mathbf{C} \cdot \mathbf{P} = 0$ implies that we also have $\mathbf{F} \cdot \mathbf{P} = 0$, we can conclude that the intersection of the near and far planes occurs in the x-y plane, as shown in Figure 4(a).

Since the maximum projected depth of a point is achieved at the far plane, projected depth no longer represents the distance along the z-axis, but rather a value corresponding to the position between the new near and far planes. This has a severe impact on depth-buffer precision along different directions in the view frustum. Fortunately, we have a recourse for minimizing this effect, and it is to make the angle between the near and far planes as small as possible. The plane **C** possesses an implicit scale factor that we have not yet restricted in any way. Changing the scale of **C** causes the orientation of the far plane **F** to change, so we need to calculate the appropriate scale that minimizes the angle between **C** and **F** without clipping any part of the original view frustum, as shown in Figure 4(b).

Let $\mathbf{C}' = \left(\mathbf{M}^{-1}\right)^{\mathrm{T}}\mathbf{C}$ be the projection of the new near plane into clip space (using the original projection matrix **M**). The corner $\mathbf{Q}'$ of the view frustum lying opposite the plane $\mathbf{C}'$ is given by

$$\mathbf{Q}' = \left\langle \operatorname{sgn}\left(C_x'\right), \operatorname{sgn}\left(C_y'\right), 1, 1 \right\rangle. \tag{7}$$

(For most perspective projections, it is safe to assume that the signs of C_x' and C_y' are the same as C_x and C_y, so the projection of **C** into clip space can be avoided.) Once we have determined the components of $\mathbf{Q}'$, we obtain its camera-space counterpart **Q** by computing $\mathbf{Q} = \mathbf{M}^{-1}\mathbf{Q}'$. For a standard view frustum, **Q** coincides with the point furthest from the plane **C** where two side planes meet the far plane.

To force the far plane to contain the point **Q**, we must require that $\mathbf{F} \cdot \mathbf{Q} = 0$. The only part of Equation (6) that we can modify is the scale of the plane **C**, so we introduce a factor a as follows.

$$\mathbf{F} = 2\mathbf{M}_4 - a\mathbf{C}. \tag{8}$$

Solving the equation $\mathbf{F} \cdot \mathbf{Q} = 0$ for a yields

$$a = \frac{2\mathbf{M}_4 \cdot \mathbf{Q}}{\mathbf{C} \cdot \mathbf{Q}}. \tag{9}$$

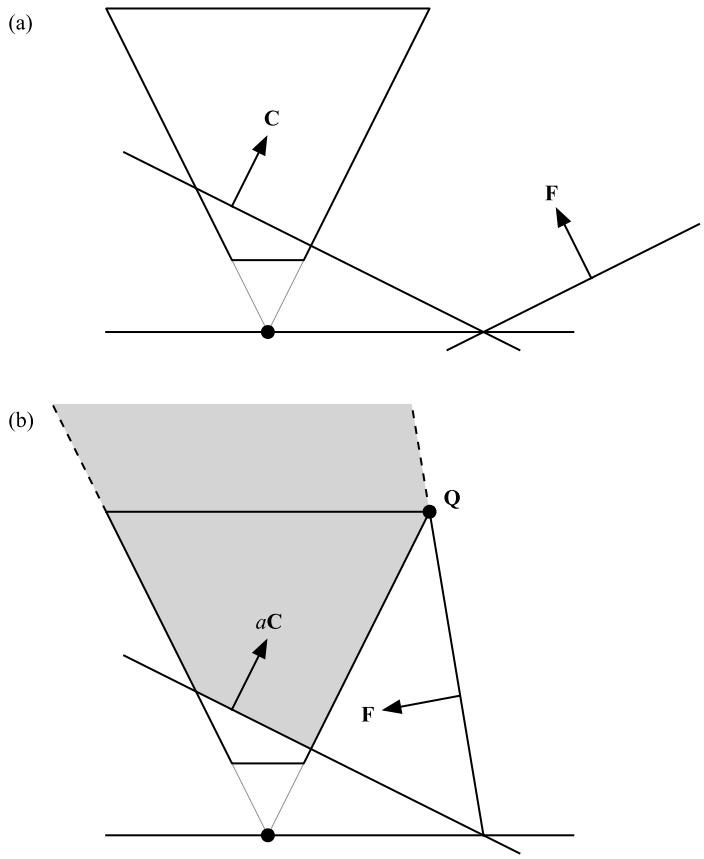

Figure 4. (a) The modified far plane **F** given by Equation (6) intersects the modified near plane **C** in the *x*-*y* plane. (b) Scaling the near plane **C** by the value *a* given by Equation (9) adjusts the far plane so that the angle between the near and far planes is as small as possible without clipping any part of the original view frustum. The shaded area represents the volume of space that is not clipped by the modified view frustum.

Replacing **C** with $a\mathbf{C}$ in Equation (5) gives us

$$\mathbf{M}_3' = a\mathbf{C} - \mathbf{M}_4, \tag{10}$$

and this produces the optimal far plane orientation shown in Figure 4(b). It should be noted that this technique will also work correctly in the case that **M** is an infinite projection matrix (i.e., one that places the conventional far plane at infinity) by forcing the new far plane to be parallel to one of the edges of the view frustum where two side planes meet.

4. Effect on the Standard View Frustum

Given the theory presented in the previous section, we now examine its application to the standard perspective projection matrix. We also analyze the effect that that an oblique near clipping plane has on depth buffer precision. The standard OpenGL perspective projection matrix **M** is given by

$$\mathbf{M} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}, \tag{11}$$

where n is the distance to the near plane, f is the distance to the far plane, and the values l, r, b, and t represent the left, right, bottom, and top edges of the rectangle carved out of the near plane by the four side planes of the view frustum [3]. The inverse of **M** is given by

$$\mathbf{M}^{-1} = \begin{bmatrix} \frac{r-l}{2n} & 0 & 0 & \frac{r+l}{2n} \\ 0 & \frac{t-b}{2n} & 0 & \frac{t+b}{2n} \\ 0 & 0 & 0 & -1 \\ 0 & 0 & -\frac{f-n}{2fn} & \frac{f+n}{2fn} \end{bmatrix}. \tag{12}$$

Given a clipping plane **C**, Equations (9) and (10) give us the following formula for the third row of the modified projection matrix **M′**.

$$\mathbf{M}'_3 = \frac{2\mathbf{M}_4 \cdot \mathbf{Q}}{\mathbf{C} \cdot \mathbf{Q}}\mathbf{C} - \mathbf{M}_4 \tag{13}$$

Since $\mathbf{M}_4 = \langle 0, 0, -1, 0 \rangle$, this simplifies to

$$\mathbf{M}'_3 = \frac{-2Q_z}{\mathbf{C} \cdot \mathbf{Q}}\mathbf{C} + \langle 0, 0, 1, 0 \rangle. \tag{14}$$

Applying the value of $\mathbf{M}^{-1}$ given by Equation (12) to the point $\mathbf{Q}'$ given by Equation (7), we have

$$\mathbf{Q} = \begin{bmatrix} \operatorname{sgn}(C_x)\dfrac{r-l}{2n} + \dfrac{r+l}{2n} \\ \operatorname{sgn}(C_y)\dfrac{t-b}{2n} + \dfrac{t+b}{2n} \\ -1 \\ 1/f \end{bmatrix}. \tag{15}$$

To test our procedure for modifying the projection matrix in the case that the clipping plane $\mathbf{C}$ is perpendicular to the z-axis (i.e., parallel to the conventional near plane), we assign $\mathbf{C} = \langle 0, 0, -1, -d \rangle$ for some positive distance d. The best that we can hope for is a new projection matrix in which the near plane has been moved to a distance d from the camera, but the far plane remains in its original position. Using the value of $\mathbf{Q}$ given by Equation (15), the dot product $\mathbf{C} \cdot \mathbf{Q}$ is simply $1 - d/f$. Evaluating Equation (14) yields the new third row of the projection matrix:

$$\begin{aligned} \mathbf{M}_3' &= \frac{2}{1 - d/f}\langle 0, 0, -1, -d \rangle + \langle 0, 0, 1, 0 \rangle \\ &= \left\langle 0, 0, -\frac{f+d}{f-d}, -\frac{2fd}{f-d} \right\rangle. \end{aligned} \tag{16}$$

We have thus achieved the desired result. If $d = n$, then we recover the original projection matrix shown in Equation (11).

As mentioned earlier, modifying the view frustum to perform clipping against an arbitrary plane impacts depth-buffer precision because the full range of depth values may not be used along different directions in camera space. Given an arbitrary camera-space direction vector $\mathbf{V} = \langle V_x, V_y, V_z, 0 \rangle$ with $V_z < 0$, a camera-space point $\langle 0, 0, 0, 1 \rangle + s\mathbf{V}$, with $s > 0$, produces the normalized device z-coordinate given by

$$\begin{aligned} z(s) &= \frac{\left(\mathbf{M}'\langle sV_x, sV_y, sV_z, 1 \rangle\right)_z}{\left(\mathbf{M}'\langle sV_x, sV_y, sV_z, 1 \rangle\right)_w} \\ &= \frac{(a\mathbf{C} - \mathbf{M}_4) \cdot \langle sV_x, sV_y, sV_z, 1 \rangle}{\mathbf{M}_4 \cdot \langle sV_x, sV_y, sV_z, 1 \rangle}, \end{aligned} \tag{17}$$

where a is the scale factor given by Equation (9). For $\mathbf{M}_4 = \langle 0, 0, -1, 0 \rangle$, Equation (17) becomes

$$\begin{aligned} z(s) &= \frac{asC_xV_x + asC_yV_y + asC_zV_z + sV_z + aC_w}{-sV_z} \\ &= \frac{as(\mathbf{C} \cdot \mathbf{V}) + sV_z + aC_w}{-sV_z}. \end{aligned} \tag{18}$$

We assume that $\mathbf{C} \cdot \mathbf{V} \geq 0$, since otherwise no point $\langle 0, 0, 0, 1 \rangle + s\mathbf{V}$ would fall within the viewable volume of space. Letting s tend to infinity yields the limit

$$\lim_{s\to\infty} z(s) = -\frac{a(\mathbf{C}\cdot\mathbf{V}) + V_z}{V_z}, \tag{19}$$

giving the maximum attainable normalized device z-coordinate in the direction $\mathbf{V}$.

Let us consider the direction $\mathbf{V} = \langle 0, 0, -1, 0\rangle$ that looks straight ahead from the camera position. The limiting value given by Equation (19) is less than one when the condition $aC_z > -2$ is satisfied. In this case, the z-coordinate of the far plane $\mathbf{F}$ given by Equation (8) is less than zero, and the far plane never enters the volume of space bounded by the four side planes of the view frustum. Since the far plane can never be reached along the direction $\mathbf{V}$, the range of normalized depth values that can be attained is smaller than that of the ordinary view frustum. Figure 5 illustrates the effect of the clipping plane's orientation on the utilization of the depth buffer and demonstrates that the loss of precision can be considerable. In general, the precision decreases as the angle between the normal direction of the clipping plane $\mathbf{C}$ and the z-axis increases and as the distance from the camera to the clipping plane increases.

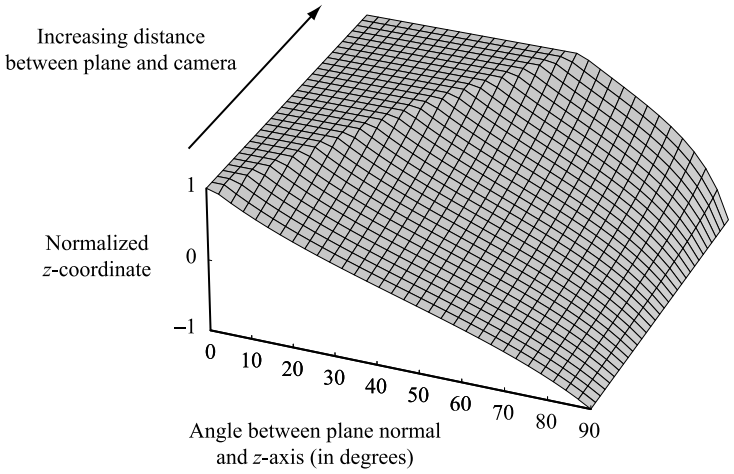

Figure 5. This surface represents the maximum attainable normalized device z-coordinate, as given by Equation (19), in the camera-space direction $\mathbf{V} = \langle 0, 0, -1, 0\rangle$. The surface is plotted as a function of the angle between the near clipping plane's normal direction and the negative z-axis and the distance from the camera to the near plane. The area in which the surface is clamped to one represents plane angles for which the modified far plane intersects the z-axis in front of the camera.

5. Conclusion

An arbitrary invertible projection matrix $\mathbf{M}$ can be modified so that the near plane is replaced by any camera-space clipping plane $\mathbf{C}$, with $C_w < 0$, by constructing a new projection matrix $\mathbf{M}'$ as follows,

$$\mathbf{M}' = \begin{bmatrix} \mathbf{M}_1 \\ \mathbf{M}_2 \\ \dfrac{2\mathbf{M}_4 \cdot \mathbf{Q}}{\mathbf{C} \cdot \mathbf{Q}}\mathbf{C} - \mathbf{M}_4 \\ \mathbf{M}_4 \end{bmatrix}, \tag{20}$$

where $\mathbf{Q} = \mathbf{M}^{-1}\mathbf{Q}'$, and $\mathbf{Q}'$ is given by Equation (7). If $\mathbf{M}$ is a perspective projection, then $\mathbf{M}_4 = \langle 0, 0, -1, 0 \rangle$ and $\mathbf{M}_3'$ can be simplified to Equation (14). This procedure constructs the optimal view frustum by maximizing usage of the available values in the depth buffer. Although depth buffer precision is diminished, we have found that enough discrete depth values are still utilized in a standard 24-bit depth buffer to render images without artifacts in almost all real-world situations.

References

[1] http://developer.nvidia.com/object/oblique_frustum_clipping.html

[2] Eric Lengyel, *Mathematics for 3D Game Programming and Computer Graphics*. Charles River Media, 2002, p. 103.

[3] OpenGL Architecture Review Board, *The OpenGL Graphics System: A Specification*. Silicon Graphics, Inc., 2004, p. 45.

Author contact information

Eric Lengyel
Terathon Software
lengyel@terathon.com

All mathematical expressions in this document were typeset with the Radical Pie equation editor. (Updated in June 2025.)