noted, however, that if the focal length and aspect ratio are known for a particular view frustum, then the formulas in Table 5.1 provide a significantly more efficient way of calculating normalized frustum planes.

5.6 Reflections and Oblique Clipping

Many scenes contain a reflective surface such as a mirror or a body of water for which a reflection image needs to be rendered. The typical way in which reflections are shown in a scene is to establish a separate image buffer called the *reflection buffer* to hold the result of rendering the objects in the scene that are visible in the reflection. The reflected scene is first rendered into the reflection buffer, and then the main scene is rendered into the main image buffer. When the geometry representing the reflective surface is rendered, colors from the corresponding pixels in the reflection buffer are read and used to contribute to the final image.

The reflected scene is rendered through a virtual camera that is the reflection of the main camera through the plane of the reflection, as shown in Figure 5.19.

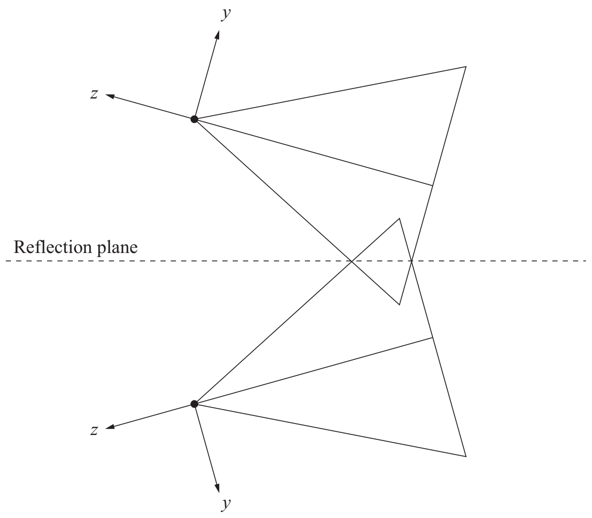

Figure 5.19. The upper view frustum represents the actual camera rendering a scene that contains a reflection plane. The virtual camera used to render the reflection is represented by the lower view frustum, and it is itself a reflection of the upper view frustum. The x axis points out of the page for both view frustums, and consequently, the camera-space coordinate system for the camera rendering the reflection is left-handed.

Since this virtual camera is a reflection, the coordinate system changes from right-handed to left-handed, and some steps need to be taken in order to account for this. In OpenGL, it is convenient to call the `glFrontFace()` function to reverse the winding order of front-facing triangles for culling purposes.

In the process of rendering from a virtual camera, it is possible that geometry lies closer to the camera than the plane representing the reflective surface. This typically happens when an object straddles the reflection plane and parts on the opposite side of the plane are flipped backwards in the reflection. If such geometry is rendered in the reflection, it can lead to unwanted artifacts in the final image, as shown in Figure 5.20.

The simplest solution to this problem is to enable a user-defined clipping plane to truncate all geometry at the reflective surface. Unfortunately, even though most GPUs support generalized user-defined clipping operations, 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. Furthermore, the necessary modifications tend to be slightly different across various GPUs.

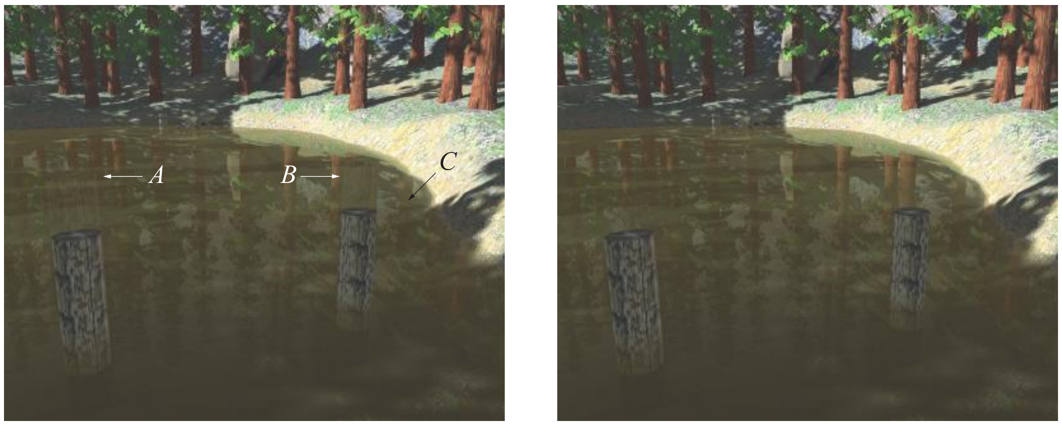

Figure 5.20. In this scene, a reflection is rendered about a plane coincident with the water surface. In the left image, no clipping is performed at the reflection plane, and multiple artifacts appear. At locations *A* and *B*, the portions of the posts that extend below the water surface are seen extending upwards in the reflection. At location *C*, some polygons belonging to submerged terrain are visible in the reflection. As shown in the right image, clipping at the reflection plane removes these unwanted artifacts. (*Image from the C4 Engine, courtesy of Terathon Software LLC.*)

In this section, we describe a trick that exploits the view frustum clipping planes that already exist for every rendered scene.[1] Normally, every geometric primitive is clipped to the six sides of the view frustum by the graphics hardware. Adding a seventh clipping plane that represents the reflective surface 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. Instead, we look for a way to modify the projection matrix so that the conventional near plane is repositioned to coincide with the reflective surface, 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.

Let $\mathbf{C} = \langle C_x, C_y, C_z, C_w \rangle$ be the plane shown in Figure 5.21, having coordinates specified in camera space, to which we would like to clip our geometry. The camera should lie on the negative side of this clipping plane, so we can assume that $C_w < 0$. The plane $\mathbf{C}$ will replace the ordinary near plane of the view frustum. As shown in Table 5.2, the camera-space near plane is given by the sum of the last two rows of the projection matrix $\mathbf{M}$, so we must somehow satisfy

$$\mathbf{C} = \mathbf{M}_4 + \mathbf{M}_3. \tag{5.61}$$

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,

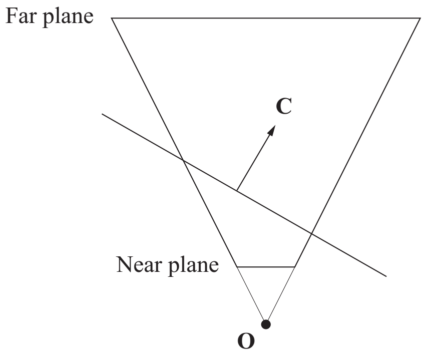

Figure 5.21. The near plane of the view frustum is replaced with the arbitrary plane $\mathbf{C}$.

[1] For a more detailed analysis, see Eric Lengyel, "Oblique Depth Projection and View Frustum Clipping", *Journal of Game Development*, Vol. 1, No. 2 (Mar 2005), pp. 5–16.

and this is necessary for perspective-correct interpolation of vertex attributes. Thus, we are left with no choice but to replace the third row of the projection matrix with

$$\mathbf{M}'_3 = \mathbf{C} - \mathbf{M}_4. \tag{5.62}$$

After making the replacement shown in Equation (5.62), the far plane $\mathbf{F}$ of the view frustum becomes

$$\begin{aligned}\mathbf{F} &= \mathbf{M}_4 - \mathbf{M}'_3 \\ &= 2\mathbf{M}_4 - \mathbf{C}.\end{aligned} \tag{5.63}$$

This fact presents a significant problem for perspective projections because 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 5.22(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 $\mathbf{C}$ possesses an implicit scale factor that we have not yet restricted in any way. Changing the scale of $\mathbf{C}$ causes the orientation of the far plane $\mathbf{F}$ to change, so we need to calculate the appropriate scale that minimizes the angle between $\mathbf{C}$ and $\mathbf{F}$ without clipping any part of the original view frustum, as shown in Figure 5.22(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 $\mathbf{M}$). The corner $\mathbf{Q}'$ of the view frustum lying opposite the plane $\mathbf{C}'$ is given by

$$\mathbf{Q}' = \langle \operatorname{sgn}(C'_x), \operatorname{sgn}(C'_y), 1, 1 \rangle. \tag{5.64}$$

(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 $\mathbf{C}$ into clip space can be avoided.) Once we have determined the components of $\mathbf{Q}'$, we obtain its camera-space counterpart $\mathbf{Q}$ by computing $\mathbf{Q} = \mathbf{M}^{-1}\mathbf{Q}'$. For a standard view frustum, $\mathbf{Q}$ coin-

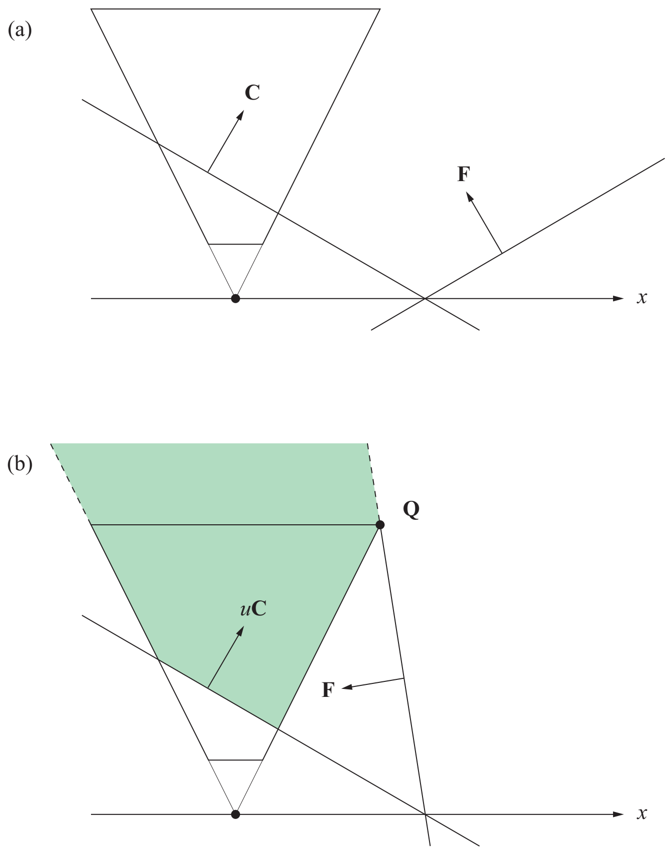

Figure 5.22. (a) The modified far plane **F** given by Equation (5.63) intersects the modified near plane **C** in the *x*-*y* plane. (b) Scaling the near plane **C** by the value *u* given by Equation (5.66) adjusts the far plane so that the angle between the near and far planes is as small as possible 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.

cides 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 (5.63) that we can modify is the scale of the plane **C**, so we introduce a factor *u* as follows:

$$\mathbf{F}=2\mathbf{M}_4-u\mathbf{C}. \tag{5.65}$$

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

$$u = \frac{2\mathbf{M}_4 \cdot \mathbf{Q}}{\mathbf{C} \cdot \mathbf{Q}}. \tag{5.66}$$

Replacing $\mathbf{C}$ with $u\mathbf{C}$ in Equation (5.62) gives us

$$\mathbf{M}'_3 = u\mathbf{C} - \mathbf{M}_4, \tag{5.67}$$

and this produces the optimal far plane orientation shown in Figure 5.22(b). For perspective projection matrices, we have $\mathbf{M}_4 = \langle 0,0,-1,0 \rangle$, so Equation (5.67) simplifies to

$$\mathbf{M}'_3 = \frac{-2Q_z}{\mathbf{C} \cdot \mathbf{Q}} \mathbf{C} + \langle 0,0,1,0 \rangle. \tag{5.68}$$

Equation (5.68) is implemented in Listing 5.1. It should be noted that this technique for optimizing the far plane also works correctly in the case that **M** is the infinite projection matrix given by Equation (5.53) by forcing the new far plane to be parallel to one of the edges of the view frustum where two side planes meet. (See Exercise 7.)

Listing 5.1. The `ModifyProjectionMatrix()` function modifies the standard OpenGL perspective projection matrix so that the near plane of the view frustum is moved to coincide with a given arbitrary plane specified by the `clipPlane` parameter.

```
inline float sgn(float x)
{
    if (x > 0.0F) return (1.0F);
    if (x < 0.0F) return (-1.0F);
    return (0.0F);
}

void ModifyProjectionMatrix(const Vector4D& clipPlane)
{
    float         matrix[16];
    Vector4D      q;

    // Grab the current projection matrix from OpenGL.
    glGetFloatv(GL_PROJECTION_MATRIX, matrix);
    // Calculate the clip-space corner point opposite the clipping plane
    // using Equation (5.64) and transform it into camera space by
    // multiplying it by the inverse of the projection matrix.
    q.x = (sgn(clipPlane.x) + matrix[8]) / matrix[0];
    q.y = (sgn(clipPlane.y) + matrix[9]) / matrix[5];
    q.z = -1.0F;
    q.w = (1.0F + matrix[10]) / matrix[14];

    // Calculate the scaled plane vector using Equation (5.68)
    // and replace the third row of the projection matrix.
    Vector4D c = clipPlane * (2.0F / Dot(clipPlane, q));
    matrix[2] = c.x;
    matrix[6] = c.y;
    matrix[10] = c.z + 1.0F;
    matrix[14] = c.w;

    // Load it back into OpenGL.
    glMatrixMode(GL_PROJECTION);
    glLoadMatrix(matrix);
}
```

Chapter 5 Summary

Lines

A line passing through the point $\mathbf{P}_0$ and running parallel to the direction $\mathbf{V}$ is expressed as

$$\mathbf{P}(t) = \mathbf{P}_0 + t\mathbf{V}.$$

The distance from a point $\mathbf{Q}$ to the line $\mathbf{P}(t)$ is given by

$$d = \sqrt{(\mathbf{Q} - \mathbf{P}_0)^2 - \frac{[(\mathbf{Q} - \mathbf{P}_0) \cdot \mathbf{V}]^2}{V^2}}.$$

Planes

A plane having normal direction $\mathbf{N}$ and containing the point $\mathbf{P}_0$ is expressed as

$$\mathbf{N} \cdot \mathbf{P} + D = 0,$$

where $D = -\mathbf{N} \cdot \mathbf{P}_0$. This can also be expressed as $\mathbf{L} \cdot \mathbf{P} = 0$, where $\mathbf{L}$ is the 4D vector $\langle \mathbf{N}, D \rangle$ and $\mathbf{P}$ is a homogeneous point with a w coordinate of 1. The distance from a point $\mathbf{Q}$ to a plane $\mathbf{L}$ is simply $\mathbf{L} \cdot \mathbf{Q}$.

Planes must be transformed using the inverse transpose of a matrix used to transform points.

Intersection of a Line and a Plane

The parameter t where a line $\mathbf{P}(t) = \mathbf{Q} + t\mathbf{V}$ intersects a plane $\mathbf{L}$ is given by

$$t = -\frac{\mathbf{L} \cdot \mathbf{Q}}{\mathbf{L} \cdot \mathbf{V}}.$$

The View Frustum

The focal length e of a view frustum having a horizontal field of view angle α is given by

$$e = \frac{1}{\tan(\alpha/2)}.$$

For a display having an aspect ratio a, the rectangle carved out of the near plane at a distance n from the camera is bounded by $x = \pm n/e$ and $y = \pm an/e$.

Perspective-Correct Interpolation

In a perspective projection, depth values z_1 and z_2 are correctly interpolated by linearly interpolating their reciprocals:

$$\frac{1}{z_3} = \frac{1}{z_1}(1-t) + \frac{1}{z_2}t.$$

Perspective-correct vertex attribute interpolation uses the similar formula

$$\frac{b_3}{z_3} = \left[\frac{b_1}{z_1}(1-t) + \frac{b_2}{z_2}t\right],$$

where b_1 and b_2 are vertex attribute values.

Perspective Projections

The perspective projection matrix $\mathbf{M}_{\text{frustum}}$ that transforms points from camera space into clip space is given by

$$\mathbf{M}_{\text{frustum}} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2nf}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix},$$

where n and f are the distances from the camera to the near and far planes, and l, r, b, and t are the left, right, bottom, and top edges of the viewing rectangle carved out of the near plane.

An infinite view frustum can be constructed by allowing the far plane distance f to tend to infinity. The corresponding projection matrix $\mathbf{M}_{\text{infinite}}$ is given by

$$\mathbf{M}_{\text{infinite}} = \lim_{f \to \infty} \mathbf{M}_{\text{frustum}} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -1 & -2n \\ 0 & 0 & -1 & 0 \end{bmatrix}.$$

Oblique Near-Plane Clipping

A perspective 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,

$$\mathbf{M}' = \begin{bmatrix} \mathbf{M}_1 \\ \mathbf{M}_2 \\ \frac{-2Q_z}{\mathbf{C} \cdot \mathbf{Q}} \mathbf{C} + \langle 0,0,1,0 \rangle \\ \mathbf{M}_4 \end{bmatrix},$$

where $\mathbf{Q} = \mathbf{M}^{-1}\mathbf{Q}'$, and $\mathbf{Q}'$ is given by Equation (5.64).