Fast Extraction of Viewing Frustum Planes from the World-View-Projection Matrix

Authors (in alphabetical order):

Gil Gribb
(ggribb@ravensoft.com)

Klaus Hartmann
(k_hartmann@osnabrueck.netsurf.de)

1. Introduction

We present an algorithm for extracting the six viewing frustum planes directly from the combined world, view, and projection matrices. The algorithm is fast, accurate, and general. It allows one to quickly determine the frustum planes in camera space, world space, or object space.

The paper is divided into two main chapters. The first chapter "Plane Extraction in Direct3D" shows how to extract the viewing frustum planes in Direct3D, and the second chapter "Plane Extraction in OpenGL" shows how to do the same in OpenGL.

In addition, there are two appendices. Appendix A reviews some of the maths behind plane equations, and Appendix B contains sample implementations of the plane extraction algorithm for both, Direct3D and OpenGL.

We hope you enjoy this short paper.

Gil Gribb
Klaus Hartmann
06/15/2001

2. Plane Extraction in Direct3D

We start off by extracting the frustum planes from the projection matrix only. That is, the world and view matrices are both identity matrices. This means that the camera is located at the origin of the world coordinate system, and we are looking along the positive z-axis.

Let $\mathbf{v} = \begin{pmatrix} x & y & z & w=1 \end{pmatrix}$ be a vertex and let $\mathbf{M} = (m_{ij})$ be a 4×4 projection matrix. Transforming the vertex $\mathbf{v}$ with the matrix $\mathbf{M}$ results in the transformed vertex $\mathbf{v}' = \begin{pmatrix} x' & y' & z' & w' \end{pmatrix}$. This can be written as:

$$\mathbf{v}' = \mathbf{v}\mathbf{M} \Rightarrow \begin{pmatrix} x' & y' & z' & w' \end{pmatrix} = \begin{pmatrix} x \cdot m_{11} + y \cdot m_{21} + z \cdot m_{31} + w \cdot m_{41} \\ x \cdot m_{12} + y \cdot m_{22} + z \cdot m_{32} + w \cdot m_{42} \\ x \cdot m_{13} + y \cdot m_{23} + z \cdot m_{33} + w \cdot m_{43} \\ x \cdot m_{14} + y \cdot m_{24} + z \cdot m_{34} + w \cdot m_{44} \end{pmatrix}^T = \begin{pmatrix} \mathbf{v} \bullet col_1 \\ \mathbf{v} \bullet col_2 \\ \mathbf{v} \bullet col_3 \\ \mathbf{v} \bullet col_4 \end{pmatrix}^T$$

where $\bullet$ denotes the dot product, and $col_j = \begin{pmatrix} m_{1j} & m_{2j} & m_{3j} & m_{4j} \end{pmatrix}$ denotes the vector represented by the j^{th} column of matrix $\mathbf{M}$. For example, $\mathbf{v} \bullet col_2$ denotes the dot product of $\mathbf{v}$ with the vector represented by the 2nd column of matrix $\mathbf{M}$.

After this transformation, the vertex $\mathbf{v}'$ is in homogeneous clipping space. In this space the viewing frustum actually is an axis-aligned box whose size is API-specific. If vertex $\mathbf{v}'$ is inside this box, then the untransformed vertex $\mathbf{v}$ is inside the 'untransformed' viewing frustum. Using Direct3D, the vertex $\mathbf{v}'$ is inside the box, if the following inequalities are all true for the components of $\mathbf{v}'$:

$$-w' < x' < w'$$
$$-w' < y' < w'$$
$$0 < z' < w'$$

There are a couple conclusions we can draw from these inequalities. They are listed in the following table.

If	$-w' < x'$	then	x' is in the inside-halfspace of the left clipping plane
If	$x' < w'$	then	x' is in the inside-halfspace of the right clipping plane
If	$-w' < y'$	then	y' is in the inside-halfspace of the bottom clipping plane
If	$y' < w'$	then	y' is in the inside-halfspace of the top clipping plane
If	$0 < z'$	then	z' is in the inside-halfspace of the near clipping plane
If	$z' < w'$	then	z' is in the inside-halfspace of the far clipping plane

Now suppose that we wanted to test, if x' is in the inside-halfspace of the left clipping plane. This is the case, if the following inequality is true:

$$-w' < x'$$

Using the information from the beginning of this chapter, we can rewrite this inequality as:

$$-(\mathbf{v} \bullet col_4) < (\mathbf{v} \bullet col_1)$$

This in turn is equal to:

$$0 < (\mathbf{v} \bullet col_4) + (\mathbf{v} \bullet col_1)$$

And finally:

$$0 < \mathbf{v} \bullet (col_4 + col_1)$$

This already represents the plane equation for the left clipping plane of the 'untransformed' viewing frustum:

$$x(m_{14} + m_{11}) + y(m_{24} + m_{21}) + z(m_{34} + m_{31}) + w(m_{44} + m_{41}) = 0$$

Since $w = 1$, we can simplify this as follows:

$$x(m_{14} + m_{11}) + y(m_{24} + m_{21}) + z(m_{34} + m_{31}) + (m_{44} + m_{41}) = 0$$

This gives a standard plane equation:

$$ax + by + cz + d = 0,$$

where

$$a = m_{14} + m_{11}, \quad b = m_{24} + m_{21}, \quad c = m_{34} + m_{31}, \quad d = m_{44} + m_{41}$$

We have now shown that the left clipping plane of the viewing frustum can be directly extracted from the projection matrix. It is important to note, that the resulting plane equation is not normalized (i.e., the plane's normal vector is

not a unit vector), and that the normal vector is pointing to the inside halfspace. This means, that $0 < ax + by + cz + d$, if the vertex $\mathbf{v}$ is in the inside halfspace of the left clipping plane.

We can now repeat the above steps for the other inequalities to determine the plane equations for the remaining clipping planes.

Clipping plane	Inequality	Coefficients for the plane equation
left	$-w' < x'$ $-(\mathbf{v} \bullet col_4) < \mathbf{v} \bullet col_1$ $0 < (\mathbf{v} \bullet col_4) + (\mathbf{v} \bullet col_1)$ $0 < \mathbf{v} \bullet (col_4 + col_1)$	$a = m_{14} + m_{11}$ $b = m_{24} + m_{21}$ $c = m_{34} + m_{31}$ $d = m_{44} + m_{41}$
right	$x' < w'$ $\mathbf{v} \bullet col_1 < \mathbf{v} \bullet col_4$ $0 < (\mathbf{v} \bullet col_4) - (\mathbf{v} \bullet col_1)$ $0 < \mathbf{v} \bullet (col_4 - col_1)$	$a = m_{14} - m_{11}$ $b = m_{24} - m_{21}$ $c = m_{34} - m_{31}$ $d = m_{44} - m_{41}$
bottom	$-w' < y'$ $-(\mathbf{v} \bullet col_4) < \mathbf{v} \bullet col_2$ $0 < (\mathbf{v} \bullet col_4) + (\mathbf{v} \bullet col_2)$ $0 < \mathbf{v} \bullet (col_4 + col_2)$	$a = m_{14} + m_{12}$ $b = m_{24} + m_{22}$ $c = m_{34} + m_{32}$ $d = m_{44} + m_{42}$
top	$y' < w'$ $\mathbf{v} \bullet col_2 < \mathbf{v} \bullet col_4$ $0 < (\mathbf{v} \bullet col_4) - (\mathbf{v} \bullet col_2)$ $0 < \mathbf{v} \bullet (col_4 - col_2)$	$a = m_{14} - m_{12}$ $b = m_{24} - m_{22}$ $c = m_{34} - m_{32}$ $d = m_{44} - m_{42}$
near	$0 < z'$ $0 < \mathbf{v} \bullet col_3$	$a = m_{13}$ $b = m_{23}$ $c = m_{33}$ $d = m_{43}$
far	$z' < w'$ $\mathbf{v} \bullet col_3 < \mathbf{v} \bullet col_4$ $0 < (\mathbf{v} \bullet col_4) - (\mathbf{v} \bullet col_3)$ $0 < \mathbf{v} \bullet (col_4 - col_3)$	$a = m_{14} - m_{13}$ $b = m_{24} - m_{23}$ $c = m_{34} - m_{33}$ $d = m_{44} - m_{43}$

2.1 Supporting Non-Identity World and View Matrices

Until now we have assumed that both, the world and the view matrix are identity matrices. However, the goal obviously is to make the algorithm work for an arbitrary view. This is, in fact, so easy that it's almost unbelievable. If you think about it for a moment then you'll immediately understand it, and that's why we are not going to explain this in any great detail. All we are giving to you is this:

1. If the matrix $\mathbf{M}$ is equal to the projection matrix $\mathbf{P}$ (i.e., $\mathbf{M} = \mathbf{P}$), then the algorithm gives the clipping planes in view space (i.e., camera space).

2. If the matrix $\mathbf{M}$ is equal to the combined view and projection matrices, then the algorithm gives the clipping planes in world space (i.e., $\mathbf{M} = \mathbf{V} \cdot \mathbf{P}$, where $\mathbf{V}$ is the view matrix, and $\mathbf{P}$ is the projection matrix).
3. If the matrix $\mathbf{M}$ is equal to the combined world, view, and projection matrices, then the algorithm gives the clipping planes in object space (i.e., $\mathbf{M} = \mathbf{W} \cdot \mathbf{V} \cdot \mathbf{P}$, where $\mathbf{W}$ is the world matrix, $\mathbf{V}$ is the view matrix, and $\mathbf{P}$ is the projection matrix).
4. and so on...

3. Plane Extraction in OpenGL

We start off by extracting the frustum planes from the projection matrix only. That is, the modelview matrix is an identity matrix. This means that the camera is located at the origin of the world coordinate system, and we are looking along the positive z-axis.

Let $\mathbf{v} = \begin{pmatrix} x & y & z & w=1 \end{pmatrix}^T$ be a vertex and let $\mathbf{M} = (m_{ij})$ be a 4×4 projection matrix. Transforming the vertex $\mathbf{v}$ with the matrix $\mathbf{M}$ results in the transformed vertex $\mathbf{v}' = \begin{pmatrix} x' & y' & z' & w' \end{pmatrix}^T$. This can be written as:

$$\mathbf{M}\mathbf{v} = \mathbf{v}' \Rightarrow \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} x \cdot m_{11} + y \cdot m_{12} + z \cdot m_{13} + w \cdot m_{14} \\ x \cdot m_{21} + y \cdot m_{22} + z \cdot m_{23} + w \cdot m_{24} \\ x \cdot m_{31} + y \cdot m_{32} + z \cdot m_{33} + w \cdot m_{34} \\ x \cdot m_{41} + y \cdot m_{42} + z \cdot m_{43} + w \cdot m_{44} \end{pmatrix} = \begin{pmatrix} \mathbf{v} \bullet row_1 \\ \mathbf{v} \bullet row_2 \\ \mathbf{v} \bullet row_3 \\ \mathbf{v} \bullet row_4 \end{pmatrix}$$

where $\bullet$ denotes the dot product, and $row_i = \begin{pmatrix} m_{i1} & m_{i2} & m_{i3} & m_{i4} \end{pmatrix}$ denotes the vector represented by the i^{th} row of matrix $\mathbf{M}$. For example, $\mathbf{v} \bullet row_2$ denotes the dot product of $\mathbf{v}$ with the vector represented by the 2nd row of matrix $\mathbf{M}$.

After this transformation, the vertex $\mathbf{v}'$ is in homogeneous clipping space. In this space the viewing frustum actually is an axis-aligned box whose size is API-specific. If vertex $\mathbf{v}'$ is inside this box, then the untransformed vertex $\mathbf{v}$ is inside the 'untransformed' viewing frustum. Using OpenGL, the vertex $\mathbf{v}'$ is inside the box, if the following inequalities are all true for the components of $\mathbf{v}'$:

$$-w' < x' < w'$$
$$-w' < y' < w'$$
$$-w' < z' < w'$$

There are a couple conclusions we can draw from these inequalities. They are listed in the following table.

If	$-w' < x'$	then	x' is in the inside-halfspace of the left clipping plane
If	$x' < w'$	then	x' is in the inside-halfspace of the right clipping plane
If	$-w' < y'$	then	y' is in the inside-halfspace of the bottom clipping plane
If	$y' < w'$	then	y' is in the inside-halfspace of the top clipping plane
If	$-w' < z'$	then	z' is in the inside-halfspace of the near clipping plane
If	$z' < w'$	then	z' is in the inside-halfspace of the far clipping plane

Now suppose that we wanted to test, if x' is in the inside-halfspace of the left clipping plane. This is the case, if the following inequality is true:

$$-w' < x'$$

Using the information from the beginning of this chapter, we can rewrite this inequality as:

$$-(\mathbf{v} \bullet row_4) < (\mathbf{v} \bullet row_1)$$

This in turn is equal to:

$$0 < (\mathbf{v} \bullet row_4) + (\mathbf{v} \bullet row_1)$$

And finally:

$$0 < \mathbf{v} \bullet (row_4 + row_1)$$

This already represents the plane equation for the left clipping plane of the 'untransformed' viewing frustum:

$$x(m_{41} + m_{11}) + y(m_{42} + m_{12}) + z(m_{43} + m_{13}) + w(m_{44} + m_{14}) = 0$$

Since $w = 1$, we can simplify this as follows:

$$x(m_{41} + m_{11}) + y(m_{42} + m_{12}) + z(m_{43} + m_{13}) + (m_{44} + m_{14}) = 0$$

This gives a standard plane equation:

$$ax + by + cz + d = 0,$$

where

$$a = m_{41} + m_{11}, \quad b = m_{42} + m_{12}, \quad c = m_{43} + m_{13}, \quad d = m_{44} + m_{14}$$

We have now shown that the left clipping plane of the viewing frustum can be directly extracted from the projection matrix. It is important to note, that the resulting plane equation is not normalized (i.e., the plane's normal vector is not a unit vector), and that the normal vector is pointing to the inside halfspace. This means, that $0 < ax + by + cz + d$, if the vertex $\mathbf{v}$ is in the inside halfspace of the left clipping plane.

We can now repeat the above steps for the other inequalities to determine the plane equations for the remaining clipping planes.

Clipping plane	**Inequality**			**Coefficients for the plane equation**
left	$-w'$	$<$	x'	$a = m_{41} + m_{11}$
	$-(\mathbf{v} \bullet row_4)$	$<$	$\mathbf{v} \bullet row_1$	$b = m_{42} + m_{12}$
	0	$<$	$(\mathbf{v} \bullet row_4) + (\mathbf{v} \bullet row_1)$	$c = m_{43} + m_{13}$
	0	$<$	$\mathbf{v} \bullet (row_4 + row_1)$	$d = m_{44} + m_{14}$
right	x'	$<$	w'	$a = m_{41} - m_{11}$
	$\mathbf{v} \bullet row_1$	$<$	$\mathbf{v} \bullet row_4$	$b = m_{42} - m_{12}$
	0	$<$	$(\mathbf{v} \bullet row_4) - (\mathbf{v} \bullet row_1)$	$c = m_{43} - m_{13}$
	0	$<$	$\mathbf{v} \bullet (row_4 - row_1)$	$d = m_{44} - m_{14}$

bottom	$-w' < y'$ $-(\mathbf{v} \bullet row_4) < \mathbf{v} \bullet row_2$ $0 < (\mathbf{v} \bullet row_4) + (\mathbf{v} \bullet row_2)$ $0 < \mathbf{v} \bullet (row_4 + row_2)$	$a = m_{41} + m_{21}$ $b = m_{42} + m_{22}$ $c = m_{43} + m_{23}$ $d = m_{44} + m_{24}$
top	$y' < w'$ $\mathbf{v} \bullet row_2 < \mathbf{v} \bullet row_4$ $0 < (\mathbf{v} \bullet row_4) - (\mathbf{v} \bullet row_2)$ $0 < \mathbf{v} \bullet (row_4 - row_2)$	$a = m_{41} - m_{21}$ $b = m_{42} - m_{22}$ $c = m_{43} - m_{23}$ $d = m_{44} - m_{24}$
near	$-w' < z'$ $-(\mathbf{v} \bullet row_4) < \mathbf{v} \bullet row_3$ $0 < (\mathbf{v} \bullet row_4) + (\mathbf{v} \bullet row_3)$ $0 < \mathbf{v} \bullet (row_4 + row_3)$	$a = m_{41} + m_{31}$ $b = m_{42} + m_{32}$ $c = m_{43} + m_{33}$ $d = m_{44} + m_{34}$
far	$z' < w'$ $\mathbf{v} \bullet row_3 < \mathbf{v} \bullet row_4$ $0 < (\mathbf{v} \bullet row_4) - (\mathbf{v} \bullet row_3)$ $0 < \mathbf{v} \bullet (row_4 - row_3)$	$a = m_{41} - m_{31}$ $b = m_{42} - m_{32}$ $c = m_{43} - m_{33}$ $d = m_{44} - m_{34}$

3.1 Supporting a Non-Identity Modelview Matrix

Until now we have assumed that the modelview matrix is an identity matrices. However, the goal obviously is to make the algorithm work for an arbitrary view. This is, in fact, so easy that it's almost unbelievable. If you think about it for a moment then you'll immediately understand it, and that's why we are not going to explain this in any great detail. All we are giving to you is this:

1. If the matrix $\mathbf{M}$ is equal to the projection matrix $\mathbf{P}$ (i.e., $\mathbf{M} = \mathbf{P}$), then the algorithm gives the clipping planes in view space (i.e., camera space).
2. If the matrix $\mathbf{M}$ is equal to the combined projection and modelview matrices, then the algorithm gives the clipping planes in model space (i.e., $\mathbf{V} \cdot \mathbf{P} = \mathbf{M}$, where $\mathbf{V}$ is the modelview matrix, and $\mathbf{P}$ is the projection matrix).

Appendix A – Plane Equations

Given a fixed point $\mathbf{p}$ and a non-zero normal vector $\mathbf{n}$, a plane is defined as the set of all points $\mathbf{x} = (x, y, z)$, such that $\mathbf{n} \bullet (\mathbf{x} - \mathbf{p}) = 0$.

In English that means, that the vector from $\mathbf{p}$ to any point $\mathbf{x}$ is orthogonal (perpendicular) to the plane's normal vector $\mathbf{n}$, if the point $\mathbf{x}$ is on the plane. Remember that the dot product of two vectors is zero, if the angle between those two vectors is 90 degrees.

We are now going to bring the plane equation $\mathbf{n} \bullet (\mathbf{x} - \mathbf{p}) = 0$ into another form:

$$\mathbf{n} \bullet (\mathbf{x} - \mathbf{p}) = 0$$

Is equal to:

$$(\mathbf{n} \bullet \mathbf{x}) - (\mathbf{n} \bullet \mathbf{p}) = 0$$

If $\mathbf{n} = \begin{pmatrix} a & b & c \end{pmatrix}$, then we can rewrite the above as:

$$ax + by + cz - (\mathbf{n} \bullet \mathbf{p}) = 0$$

The term $-(\mathbf{n} \bullet \mathbf{p})$ is constant and we now define that $d := -(\mathbf{n} \bullet \mathbf{p})$. Thus the plane equation becomes:

$$ax + by + cz + d = 0$$

It should be noted, that many people prefer to use the form $ax + by + cz = d$. In this case, the constant d is defined as $d := (\mathbf{n} \bullet \mathbf{p})$.

For use in a computer language, we can store such a plane equation in a simple structure.

```
struct Plane
{
    float   a, b, c, d;
};
```

A.1 – Halfspaces

A plane cuts three-dimensional space into two separate parts. These parts are called halfspaces. The halfspace the plane's normals vector points into is called the positive halfspace, and the other halfspace is called the negative halfspace.

A.2 – Normalizing the Plane Equation

Normalizing a plane equation $ax + by + cz + d = 0$ means to change the plane equation, such that the normal vector $\mathbf{n} = \begin{pmatrix} a & b & c \end{pmatrix}$ becomes a unit vector (i.e., $\|\mathbf{n}\| = 1$). All we need to do to normalize a vector is to divide each of its components by the magnitude of the vector. However, in order to normalize a plane equation it is not enough to simply divide the coefficients a, b, c by the magnitude of the normal vector $\mathbf{n}$. We also need to divide the constant d by the magnitude of the normal vector $\mathbf{n}$:

$$\mathbf{n} \bullet (\mathbf{x} - \mathbf{p}) = 0$$

First we normalize $\mathbf{n}$ by dividing its components by the magnitude of $\mathbf{n}$:

$$\frac{\mathbf{n}}{\|\mathbf{n}\|} \bullet (\mathbf{x} - \mathbf{p}) = 0$$

This is equal to:

$$\left(\frac{\mathbf{n}}{\|\mathbf{n}\|} \bullet \mathbf{x} \right) - \left(\frac{\mathbf{n}}{\|\mathbf{n}\|} \bullet \mathbf{p} \right) = 0$$

If $\mathbf{n} = \begin{pmatrix} a & b & c \end{pmatrix}$, then we can rewrite the above as:

$$\frac{a}{\|\mathbf{n}\|}x + \frac{b}{\|\mathbf{n}\|}y + \frac{c}{\|\mathbf{n}\|}z - \left(\frac{\mathbf{n}}{\|\mathbf{n}\|} \bullet \mathbf{p}\right) = 0$$

Now define $d := -(\mathbf{n} \bullet \mathbf{p})$:

$$\frac{a}{\|\mathbf{n}\|}x + \frac{b}{\|\mathbf{n}\|}y + \frac{c}{\|\mathbf{n}\|}z + \frac{d}{\|\mathbf{n}\|} = 0$$

Which is equal to:

$$\frac{a}{\sqrt{a^2+b^2+c^2}}x + \frac{b}{\sqrt{a^2+b^2+c^2}}y + \frac{c}{\sqrt{a^2+b^2+c^2}}z + \frac{d}{\sqrt{a^2+b^2+c^2}} = 0$$

Now simplify:

$$\frac{ax+by+cz+d}{\sqrt{a^2+b^2+c^2}} = 0$$

So in order to normalize a plane equation, we need to divide the coefficients a, b, c and the constant d by the magnitude of the normal vector $\mathbf{n} = \begin{pmatrix} a & b & c \end{pmatrix}$. It is useful to have a function that does this for you:

```
void NormalizePlane(Plane & plane)
{
    float   mag;

    mag = sqrt(plane.a * plane.a + plane.b * plane.b + plane.c * plane.c);

    plane.a = plane.a / mag;
    plane.b = plane.b / mag;
    plane.c = plane.c / mag;
    plane.d = plane.d / mag;
}
```

A.3 – The Signed Distance From a Plane to a Point

It is more than often necessary to compute the shortest signed distance from a given plane $ax + by + cz + d = 0$ to a point $\mathbf{p} = (x, y, z)$. As it turns out, we can simply plug the coordinates of the point $\mathbf{p}$ into the plane equation, and the result is the signed distance from the plane to $\mathbf{p}$. This distance, however, is not necessarily a 'true' distance. Instead it is the signed distance in units of the magnitude of the plane's normal vector $\mathbf{n} = \begin{pmatrix} a & b & c \end{pmatrix}$. So in order to obtain a 'true' distance, you need to normalize the plane equation.

```
float DistanceToPoint(const Plane & plane, const Point & pt)
{
    return plane.a*pt.x + plane.b*pt.y + plane.c*pt.z + plane.d;
}
```

If the plane equation is not normalized, then we can still get some valuable information from the 'non-true' distance $dist$:

1. If $dist < 0$, then the point $\mathbf{p}$ lies in the negative halfspace.
2. If $dist = 0$, then the point $\mathbf{p}$ lies in the plane.
3. If $dist > 0$, then the point $\mathbf{p}$ lies in the positive halfspace.

This gives us another useful function that also works for non-normalized plane equations:

```
enum Halfspace
{
    NEGATIVE = -1
    ON_PLANE = 0,
    POSITIVE = 1,
};

Halfspace ClassifyPoint(const Plane & plane, const Point & pt)
{
    float d;

    d = plane.a*pt.x + plane.b*pt.y + plane.c*pt.z + plane.d;

    if (d < 0) return NEGATIVE;
    if (d > 0) return POSITIVE;
    return ON_PLANE;
}
```

Appendix B – Implementation

In the following we are going to give sample implementations of the plane extraction method for both, OpenGL and Direct3D.

Note, that the normal vectors of the resulting planes are pointing to the inside of the viewing frustum. If you prefer planes where the normal vectors are pointing to the outside, then all you need to do is to negate the coefficients a, b, c and the constant d of each plane equation.

B.1 Plane Extraction for OpenGL

```
struct Matrix4x4
{
    // The elements of the 4x4 matrix are stored in
    // column-major order (see "OpenGL Programming Guide",
    // 3rd edition, pp 106, glLoadMatrix).
    float   _11, _21, _31, _41;
    float   _12, _22, _32, _42;
    float   _13, _23, _33, _43;
    float   _14, _24, _34, _44;
};

void ExtractPlanesGL(
    Plane * p_planes,
    const Matrix4x4 & comboMatrix,
    bool normalize)
{
    // Left clipping plane
```

```
    p_planes[0].a = comboMatrix._41 + comboMatrix._11;
    p_planes[0].b = comboMatrix._42 + comboMatrix._12;
    p_planes[0].c = comboMatrix._43 + comboMatrix._13;
    p_planes[0].d = comboMatrix._44 + comboMatrix._14;

    // Right clipping plane
    p_planes[1].a = comboMatrix._41 - comboMatrix._11;
    p_planes[1].b = comboMatrix._42 - comboMatrix._12;
    p_planes[1].c = comboMatrix._43 - comboMatrix._13;
    p_planes[1].d = comboMatrix._44 - comboMatrix._14;

    // Top clipping plane
    p_planes[2].a = comboMatrix._41 - comboMatrix._21;
    p_planes[2].b = comboMatrix._42 - comboMatrix._22;
    p_planes[2].c = comboMatrix._43 - comboMatrix._23;
    p_planes[2].d = comboMatrix._44 - comboMatrix._24;

    // Bottom clipping plane
    p_planes[3].a = comboMatrix._41 + comboMatrix._21;
    p_planes[3].b = comboMatrix._42 + comboMatrix._22;
    p_planes[3].c = comboMatrix._43 + comboMatrix._23;
    p_planes[3].d = comboMatrix._44 + comboMatrix._24;

    // Near clipping plane
    p_planes[4].a = comboMatrix._41 + comboMatrix._31;
    p_planes[4].b = comboMatrix._42 + comboMatrix._32;
    p_planes[4].c = comboMatrix._43 + comboMatrix._33;
    p_planes[4].d = comboMatrix._44 + comboMatrix._34;

    // Far clipping plane
    p_planes[5].a = comboMatrix._41 - comboMatrix._31;
    p_planes[5].b = comboMatrix._42 - comboMatrix._32;
    p_planes[5].c = comboMatrix._43 - comboMatrix._33;
    p_planes[5].d = comboMatrix._44 - comboMatrix._34;

    // Normalize the plane equations, if requested
    if (normalize == true)
    {
        NormalizePlane(p_planes[0]);
        NormalizePlane(p_planes[1]);
        NormalizePlane(p_planes[2]);
        NormalizePlane(p_planes[3]);
        NormalizePlane(p_planes[4]);
        NormalizePlane(p_planes[5]);
    }
}
```

B.2 Plane Extraction for Direct3D

```
struct Matrix4x4
{
    // The elements of the 4x4 matrix are stored in
    // row-major order.
    float   _11, _12, _13, _14;
    float   _21, _22, _23, _24;
    float   _31, _32, _33, _34;
    float   _41, _42, _43, _44;
};
```

```
void ExtractPlanesD3D(
    Plane * p_planes,
    const Matrix4x4 & comboMatrix,
    bool normalize)
{
    // Left clipping plane
    p_planes[0].a = comboMatrix._14 + comboMatrix._11;
    p_planes[0].b = comboMatrix._24 + comboMatrix._21;
    p_planes[0].c = comboMatrix._34 + comboMatrix._31;
    p_planes[0].d = comboMatrix._44 + comboMatrix._41;

    // Right clipping plane
    p_planes[1].a = comboMatrix._14 - comboMatrix._11;
    p_planes[1].b = comboMatrix._24 - comboMatrix._21;
    p_planes[1].c = comboMatrix._34 - comboMatrix._31;
    p_planes[1].d = comboMatrix._44 - comboMatrix._41;

    // Top clipping plane
    p_planes[2].a = comboMatrix._14 - comboMatrix._12;
    p_planes[2].b = comboMatrix._24 - comboMatrix._22;
    p_planes[2].c = comboMatrix._34 - comboMatrix._32;
    p_planes[2].d = comboMatrix._44 - comboMatrix._42;

    // Bottom clipping plane
    p_planes[3].a = comboMatrix._14 + comboMatrix._12;
    p_planes[3].b = comboMatrix._24 + comboMatrix._22;
    p_planes[3].c = comboMatrix._34 + comboMatrix._32;
    p_planes[3].d = comboMatrix._44 + comboMatrix._42;

    // Near clipping plane
    p_planes[4].a = comboMatrix._13;
    p_planes[4].b = comboMatrix._23;
    p_planes[4].c = comboMatrix._33;
    p_planes[4].d = comboMatrix._43;

    // Far clipping plane
    p_planes[5].a = comboMatrix._14 - comboMatrix._13;
    p_planes[5].b = comboMatrix._24 - comboMatrix._23;
    p_planes[5].c = comboMatrix._34 - comboMatrix._33;
    p_planes[5].d = comboMatrix._44 - comboMatrix._43;

    // Normalize the plane equations, if requested
    if (normalize == true)
    {
        NormalizePlane(p_planes[0]);
        NormalizePlane(p_planes[1]);
        NormalizePlane(p_planes[2]);
        NormalizePlane(p_planes[3]);
        NormalizePlane(p_planes[4]);
        NormalizePlane(p_planes[5]);
    }
}
```