Chapter 5

TRANSFORMATIONS, CLIPPING AND PROJECTION

5.1 Geometric transformations

Three-dimensional graphics aims at producing an image of 3D objects. This means that the geometrical representation of the image is generated from the geometrical data of the objects. This change of geometrical description is called the **geometric transformation**. In computers the world is represented by numbers; thus geometrical properties and transformations must also be given by numbers in computer graphics. Cartesian coordinates provide this algebraic establishment for the Euclidean geometry, which define a 3D point by three component distances along three, non-coplanar axes from the origin of the coordinate system.

The selection of the origin and the axes of this coordinate system may have a significant effect on the complexity of the definition and various calculations. As mentioned earlier, the world coordinate system is usually not suitable for the definition of all objects, because here we are not only concerned with the geometry of the objects, but also with their relative position and orientation. A brick, for example, can be simplistically defined in a coordinate system having axes parallel to its edges, but the description of the box is quite complicated if arbitrary orientation is required. This consideration necessitated the application of local coordinate systems. Viewing and visibility calculations, on the other hand, have special requirements from a coordinate system where the objects are represented, to facilitate simple operations. This means that the definition and the photographing of the objects may involve several different coordinate systems suitable for the different specific operations. The transportation of objects from one coordinate system to another also requires geometric transformations.

Working in several coordinate systems can simplify the various phases of modeling and image synthesis, but it requires additional transformation steps. Thus, this approach is advantageous only if the computation needed for geometric transformations is less than the decrease of the computation of the various steps due to the specifically selected coordinate systems. Representations invariant of the transformations are the primary candidates for methods working in several coordinate systems, since they can easily be transformed by transforming the control or definition points. Polygon mesh models, Bezier and B-spline surfaces are invariant for linear transformation, since their transformation will also be polygon meshes, Bezier or B-spline surfaces, and the vertices or the control points of the transformed surface will be those coming from the transformation of the original vertices and control points.

Other representations, sustaining non-planar geometry, and containing, for example, spheres, are not easily transformable, thus they require all the calculations to be done in a single coordinate system.

Since computer graphics generates 2D images of 3D objects, some kind of projection is always involved in image synthesis. Central projection, however, creates problems (singularities) in Euclidean geometry, it is thus worthwhile considering another geometry, namely the **projective geometry**, to be used for some phases of image generation. Projective geometry is a classical branch of mathematics which cannot be discussed here in detail. A short introduction, however, is given to highlight those features that are widely used in computer graphics. Beyond this elementary introduction, the interested reader is referred to [Her91] [Cox74].

Projective geometry can be approached from the analysis of central projection as shown in figure 5.1.

For those points to which the projectors are parallel with the image plane no projected image can be defined in Euclidean geometry. Intuitively speaking these image points would be at "infinity" which is not part of the Eu-

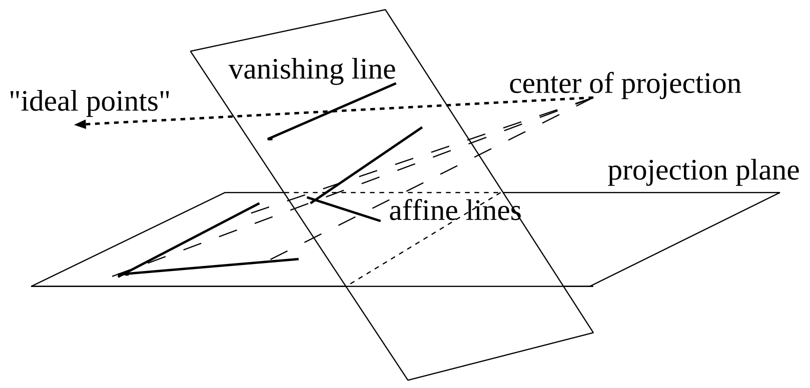

Figure 5.1: Central projection of objects on a plane

clidean space. Projective geometry fills these holes by extending the Euclidean space by new points, called **ideal points**, that can serve as the image of points causing singularities in Euclidean space. These ideal points can be regarded as "intersections" of parallel lines and planes, which are at "infinity". These ideal points form a plane of the projective space, which is called the **ideal plane**.

Since there is a one-to-one correspondence between the points of Euclidean space and the coordinate triples of a Cartesian coordinate system, the new elements obviously cannot be represented in this coordinate system, but a new algebraic establishment is needed for projective geometry. This establishment is based on **homogeneous coordinates**.

For example, by the method of homogeneous coordinates a point of space can be specified as the center of gravity of the structure containing mass X_h at reference point p_1, mass Y_h at point p_2, mass Z_h at point p_3 and mass w at point p_4. Weights are not required to be positive, thus the center of gravity can really be any point of the space if the four reference points are not co-planar. Alternatively, if the total mass, that is $h = X_h + Y_h + Z_h + w$, is not zero and the reference points are in Euclidean space, then the center of gravity will also be in the Euclidean space.

Let us call the quadruple (X_h, Y_h, Z_h, h), where $h = X_h + Y_h + Z_h + w$, the homogeneous coordinates of the center of gravity.

Note that if all weights are multiplied by the same (non-zero) factor, the center of gravity, that is the point defined by the homogeneous coordinates, does not change. Thus a point (X_h, Y_h, Z_h, h) is equivalent to points $(\lambda X_h, \lambda Y_h, \lambda Z_h, \lambda h)$, where λ is a non-zero number.

The center of gravity analogy used to illustrate the homogeneous coordinates is not really important from a mathematical point of view. What should be remembered, however, is that a 3D point represented by homogeneous coordinates is a four-vector of real numbers and all scalar multiples of these vectors are equivalent.

Points of the projective space, that is the points of the Euclidean space (also called **affine points**) plus the ideal points, can be represented by homogeneous coordinates. First the representation of affine points which can establish a correspondence between the Cartesian and the homogeneous coordinate systems is discussed. Let us define the four reference points of the homogeneous coordinate system in points [1,0,0], [0,1,0], [0,0,1] and in [0,0,0] respectively. If $h = X_h + Y_h + Z_h + w$ is not zero, then the center of gravity in Cartesian coordinate system defined by axes $\mathbf{i}, \mathbf{j}, \mathbf{k}$ is:

$$r(X_h, Y_h, Z_h, h) = \frac{1}{h}(X_h \cdot [1,0,0] + Y_h \cdot [0,1,0] + Z_h \cdot [0,0,1] + w \cdot [0,0,0]) =$$

$$\frac{X_h}{h} \cdot \mathbf{i} + \frac{Y_h}{h} \cdot \mathbf{j} + \frac{Z_h}{h} \cdot \mathbf{k}. \tag{5.1}$$

Thus with the above selection of reference points the correspondence between the homogeneous coordinates (X_h, Y_h, Z_h, h) and Cartesian coordinates (x, y, z) of affine points $(h \neq 0)$ is:

$$x = \frac{X_h}{h}, \qquad y = \frac{Y_h}{h}, \qquad z = \frac{Z_h}{h}. \tag{5.2}$$

Homogeneous coordinates can also be used to characterize planes. In the Cartesian system a plane is defined by the following equation:

$$a \cdot x + b \cdot y + c \cdot z + d = 0 \tag{5.3}$$

Applying the correspondence between the homogeneous and Cartesian coordinates, we get:

$$a \cdot X_h + b \cdot Y_h + c \cdot Z_h + d \cdot h = 0 \tag{5.4}$$

Note that the set of points that satisfy this plane equation remains the same if this equation is multiplied by a scalar factor. Thus a quadruple $[a, b, c, d]$

of homogeneous coordinates can represent not only single points but planes as well. In fact all theorems valid for points can be formulated for planes as well in 3D projective space. This symmetry is often referred to as the **duality principle**. The intersection of two planes (which is a line) can be calculated as the solution of the linear system of equations. Suppose that we have two parallel planes given by quadruples $[a, b, c, d]$ and $[a, b, c, d']$ $(d \neq d')$ and let us calculate their intersection. Formally all points satisfy the resulting equations for which

$$a \cdot X_h + b \cdot Y_h + c \cdot Z_h = 0 \quad \text{and} \quad h = 0 \tag{5.5}$$

In Euclidean geometry parallel planes do not have intersection, thus the points calculated in this way cannot be in Euclidean space, but form a subset of the ideal points of the projective space. This means that ideal points correspond to those homogeneous quadruples where $h = 0$. As mentioned, these ideal points represent the infinity, but they make a clear distinction between the "infinities" in different directions that are represented by the first three coordinates of the homogeneous form.

Returning to the equation of a projective plane or considering the equation of a projective line, we can realize that ideal points may also satisfy these equations. Therefore, projective planes and lines are a little bit more than their Euclidean counterparts. In addition to all Euclidean points, they also include some ideal points. This may cause problems when we want to return to Euclidean space because these ideal points have no counterparts.

Homogeneous coordinates can be visualized by regarding them as Cartesian coordinates of a higher dimensional space (note that 3D points are defined by 4 homogeneous coordinates). This procedure is called the embedding of the 3D projective space into the 4D Euclidean space or the **straight model** [Her91] (figure 5.2). Since it is impossible to create 4D drawings, this visualization uses a trick of reducing the dimensionality and displays the 4D space as a 3D one, the real 3D subspace as a 2D plane and relies on the reader's imagination to interpret the resulting image.

A homogeneous point is represented by a set of equivalent quadruples

$$\{(\lambda X_h, \lambda Y_h, \lambda Z_h, \lambda h) \mid \lambda \neq 0\},$$

thus a point is described as a 4D line crossing the origin, [0,0,0,0], in the straight model. Ideal points are in the $h = 0$ plane and affine points are represented by those lines that are not parallel to the $h = 0$ plane.

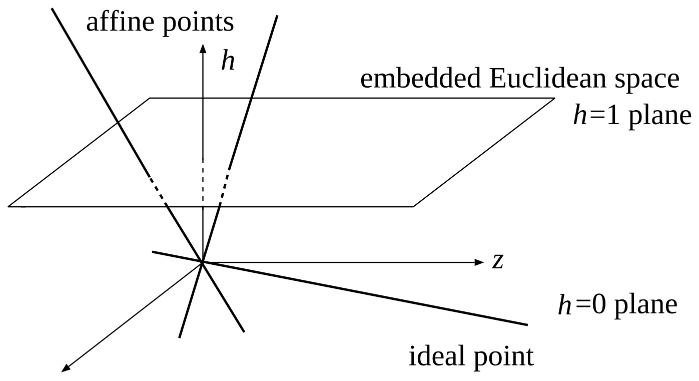

Figure 5.2: Embedding of projective space into a higher dimensional Euclidean space

Since points are represented by a set of quadruples that are equivalent in homogeneous terms, a point may be represented by any of them. Still, it is worth selecting a single representative from this set to identify points unambiguously. For affine points, this representative quadruple is found by making the fourth (h) coordinate equal to 1, which has a nice property that the first three homogeneous coordinates are equal to the Cartesian coordinates of the same point taking equation 5.2 into account, that is:

$$(\frac{X_h}{h}, \frac{Y_h}{h}, \frac{Z_h}{h}, 1) = (x, y, z, 1). \tag{5.6}$$

In the straight model thus the representatives of affine points correspond to the $h = 1$ hyperplane (a 3D set of the 4D space), where they can be identified by Cartesian coordinates. This can be interpreted as the 3D Euclidean space and for affine points the homogeneous to Cartesian conversion of coordinates can be accomplished by projecting the 4D point onto the $h = 1$ hyperplane using the origin as the center of projection. This projection means the division of the first three coordinates by the fourth and is usually called **homogeneous division**.

Using the algebraic establishment of Euclidean and projective geometries, that is the system of Cartesian and homogeneous coordinates, geometric transformations can be regarded as functions that map tuples of coordinates onto tuples of coordinates. In computer graphics linear functions are preferred that can conveniently be expressed as a vector-matrix multiplication and a vector addition. In Euclidean geometry this linear function has the following general form:

$$[x', y', z'] = [x, y, z] \cdot \mathbf{A_{3\times 3}} + [p_x, p_y, p_z]. \tag{5.7}$$

Linear transformations of this kind map affine points onto affine points, therefore they are also **affine transformations**.

When using homogeneous representation, however, it must be taken into account that equivalent quadruples differing only by a scalar multiplication must be transformed to equivalent quadruples, thus no additive constant is allowed:

$$[X'_h, Y'_h, Z'_h, h'] = [X_h, Y_h, Z_h, h] \cdot \mathbf{T_{4\times 4}}. \tag{5.8}$$

Matrix $\mathbf{T_{4\times 4}}$ defines the transformation uniquely in homogeneous sense; that is, matrices differing in a multiplicative factor are equivalent.

Note that in equations 5.7 and 5.8 row vectors are used to identify points unlike the usual mathematical notation. The preference for row vectors in computer graphics has partly historical reasons, partly stems from the property that in this way the concatenation of transformations corresponds to matrix multiplication in "normal", that is left to right, order. For column vectors, it would be the reverse order. Using the straight model, equation 5.7 can be reformulated for homogeneous coordinates:

$$[x', y', z', 1] = [x, y, z, 1] \cdot \begin{bmatrix} & 0 \\ \mathbf{A}_{3\times 3} & 0 \\ & 0 \\ \mathbf{p^T} & 1 \end{bmatrix}. \tag{5.9}$$

Note that the 3×3 matrix $\mathbf{A}$ is accommodated in $\mathbf{T}$ as its upper left minor matrix, while $\mathbf{p}$ is placed in the last row and the fourth column vector of $\mathbf{T}$ is set to constant [0,0,0,1]. This means that the linear transformations of Euclidean space form a subset of homogeneous linear transformations. This is a real subset since, as we shall see, by setting the fourth column to a vector different from [0,0,0,1] the resulting transformation does not have an affine equivalent, that is, it is not linear in the Euclidean space.

Using the algebraic treatment of homogeneous (linear) transformations, which identifies them by a 4×4 matrix multiplication, we can define the concatenation of transformations as the product of transformation matrices and the inverse of a homogeneous transformation as the inverse of its transformation matrix if it exists, i.e. its determinant is not zero. Taking into account the properties of matrix operations we can see that the concatenation of homogeneous transformations is also a homogeneous transformation and the inverse of a homogeneous transformation is also a homogeneous transformation if the transformation matrix is invertible. Since matrix multiplication is an associative operation, consecutive transformations can always be replaced by a single transformation by computing the product of the matrices of different transformation steps. Thus, any number of linear transformations can be expressed by a single 4×4 matrix multiplication. The transformation of a single point of the projective space requires 16 multiplications and 12 additions. If the point must be mapped back to the Cartesian coordinate system, then 3 divisions by the fourth homogeneous coordinate may be necessary in addition to the matrix multiplication. Since linear transformations of Euclidean space have a $[0, 0, 0, 1]$ fourth column in the transformation matrix, which is preserved by multiplications with matrices of the same property, any linear transformation can be calculated by 9 multiplications and 9 additions.

According to the theory of projective geometry, transformations defined by 4×4 matrix multiplication map points onto points, lines onto lines, planes onto planes and intersection points onto intersection points, and therefore are called **collinearities** [Her91]. The reverse of this statement is also true; each collinearity corresponds to a homogeneous transformation matrix. Instead of proving this statement in projective space, a special case that has importance in computer graphics is investigated in detail. In computer graphics the geometry is given in 3D Euclidean space and having applied some homogeneous transformation the results are also required in Euclidean space. From this point of view, the homogeneous transformation of a 3D point involves:

1. A 4×4 matrix multiplication of the coordinates extended by a fourth coordinate of value 1.

2. A homogeneous division of all coordinates in the result by the fourth coordinate if it is different from 1, meaning that the transformation forced the point out of 3D space.

It is important to note that a clear distinction must be made between the central or parallel projection defined earlier which maps 3D points onto 2D points on a plane and projective transformations which map projective space onto projective space. Now let us start the discussion of the homogeneous transformation of a special set of geometric primitives. A Euclidean line can be defined by the following equation:

$$\vec{r}(t) = \vec{r}_0 + \vec{v} \cdot t, \qquad \text{where } \ t \text{ is a real parameter.} \tag{5.10}$$

Assuming that vectors $\vec{v}_1$ and $\vec{v}_2$ are not parallel, a Euclidean plane, on the other hand, can be defined as follows:

$$\vec{r}(t_1, t_2) = \vec{r}_0 + \vec{v}_1 \cdot t_1 + \vec{v}_2 \cdot t_2, \qquad \text{where } \ t_1, t_2 \text{ are real parameters.} \tag{5.11}$$

Generally, lines and planes are special cases of a wider range of geometric structures called linear sets. By definition, a **linear set** is defined by a position vector $\vec{r}_0$ and some axes $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n$ by the following equation:

$$\vec{r}(t_1, \ldots, t_n) = \vec{r}_0 + \sum_{i=1}^{n} t_i \cdot \vec{v}_i. \tag{5.12}$$

First of all, the above definition is converted to a different one that uses homogeneous-like coordinates. Let us define the so-called spanning vectors $\vec{p}_0, \ldots, \vec{p}_n$ of the linear set as:

$$\begin{aligned} \vec{p}_0 &= \vec{r}_0, \\ \vec{p}_1 &= \vec{r}_0 + \vec{v}_1, \\ &\vdots \\ \vec{p}_n &= \vec{r}_0 + \vec{v}_n. \end{aligned} \tag{5.13}$$

The equation of the linear set is then:

$$\vec{r}(t_1, \ldots, t_n) = (1 - t_1 - \ldots - t_n) \cdot \vec{p}_0 + \sum_{i=1}^{n} t_i \cdot \vec{p}_i. \tag{5.14}$$

Introducing the new coordinates as

$$\alpha_0 = 1 - t_1 - \ldots - t_n, \quad \alpha_1 = t_1, \quad \alpha_2 = t_2, \quad \ldots, \quad \alpha_n = t_n, \tag{5.15}$$

the linear set can be written in the following form:

$$S = \{\vec{p} \ \mid \ \vec{p} = \sum_{i=0}^{n} \alpha_i \cdot \vec{p}_i \ \wedge \ \sum_{i=0}^{n} \alpha_i = 1\}. \tag{5.16}$$

The weights (α_i) are also called the **baricentric coordinates** of the point $\vec{p}$ with respect to $\vec{p}_0$, $\vec{p}_1$,...,$\vec{p}_n$. This name reflects the interpretation that $\vec{p}$ would be the center of gravity of a structure of weights $(\alpha_0, \alpha_1, \ldots, \alpha_n)$ at points $\vec{p}_0, \vec{p}_1, \ldots, \vec{p}_n$.

The homogeneous transformation of such a point $\vec{p}$ is:

$$[\vec{p}, 1] \cdot \mathbf{T} = [\sum_{i=0}^{n} \alpha_i \cdot \vec{p}_i, 1] \cdot \mathbf{T} = [\sum_{i=0}^{n} \alpha_i \cdot \vec{p}_i, \sum_{i=0}^{n} \alpha_i] \cdot \mathbf{T} =$$

$$(\sum_{i=0}^{n} \alpha_i \cdot [\vec{p}_i, 1]) \cdot \mathbf{T} = \sum_{i=0}^{n} \alpha_i \cdot ([\vec{p}_i, 1] \cdot \mathbf{T}) \qquad (5.17)$$

since $\sum_{i=0}^{n} \alpha_i = 1$. Denoting $[\vec{p}_i, 1] \cdot \mathbf{T}$ by $[\vec{P}_i, h_i]$ we get:

$$[\vec{p}, 1] \cdot \mathbf{T} = \sum_{i=0}^{n} \alpha_i \cdot [\vec{P}_i, h_i] = [\sum_{i=0}^{n} \alpha_i \cdot \vec{P}_i, \sum_{i=0}^{n} \alpha_i \cdot h_i]. \qquad (5.18)$$

If the resulting fourth coordinate $\sum_{i=0}^{n} \alpha_i \cdot h_i$ is zero, then the point $\vec{p}$ is mapped onto an ideal point, therefore it cannot be converted back to Euclidean space. These ideal points must be eliminated before the homogeneous division (see section 5.5 on clipping).

After homogeneous division we are left with:

$$[\sum_{i=0}^{n} \frac{\alpha_i \cdot h_i}{\sum_{j=1}^{n} \alpha_j \cdot h_j} \cdot \frac{\vec{P}_i}{h_i}, 1] = [\sum_{i=0}^{n} \alpha_i^* \cdot \vec{p}_i^{\,*}, 1] \qquad (5.19)$$

where $\vec{p}_i^{\,*}$ is the homogeneous transformation of $\vec{p}_i$. The derivation of α_i^* guarantees that $\sum_{i=0}^{n} \alpha_i^* = 1$. Thus, the transformation of the linear set is also linear. Examining the expression of the weights (α_i^*), we can conclude that generally $\alpha_i \neq \alpha_i^*$ meaning the homogeneous transformation may destroy equal spacing. In other words the **division ratio** is not projective invariant. In the special case when the transformation is affine, coordinates h_i will be 1, thus $\alpha_i = \alpha_i^*$, which means that equal spacing (or division ratio) is affine invariant.

A special type of linear set is the **convex hull**. The convex hull is defined by equation 5.16 with the provision that the baricentric coordinates must be non-negative.

To avoid the problems of mapping onto ideal points, let us assume the spanning vectors to be mapped onto the same side of the $h = 0$ hyperplane, meaning that the h_i-s must have the same sign. This, with $\alpha_i \geq 0$, guarantees that no points are mapped onto ideal points and

$$\alpha_i^* = \sum_{i=0}^{n} \frac{\alpha_i \cdot h_i}{\sum_{i=0}^{n} \alpha_i \cdot h_i} \geq 0 \tag{5.20}$$

Thus, baricentric coordinates of the image will also be non-negative, that is, convex hulls are also mapped onto convex hulls by homogeneous transformations if their transformed image does not contain ideal points. An arbitrary planar polygon can be broken down into triangles that are convex hulls of three spanning vectors. The transformation of this polygon will be the composition of the transformed triangles. This means that a planar polygon will also be preserved by homogeneous transformations if its image does not intersect with the $h = 0$ plane.

As mentioned earlier, in computer graphics the objects are defined in Euclidean space by Cartesian coordinates and the image is required in a 2D pixel space that is also Euclidean with its coordinates which correspond to the physical pixels of the frame buffer. Projective geometry may be needed only for specific stages of the transformation from modeling to pixel space. Since projective space can be regarded as an extension of the Euclidean space, the theory of transformations could be discussed generally only in projective space. For pedagogical reasons, however, we will use the more complicated homogeneous representations if they are really necessary for computer graphics algorithms, and deal with the Cartesian coordinates in simpler cases. This combined view of Euclidean and projective geometries may be questionable from a purely mathematical point of view, but it is accepted by the computer graphics community because of its clarity and its elimination of unnecessary abstractions.

We shall consider the transformation of points in this section, which will lead on to the transformation of planar polygons as well.

5.1.1 Elementary transformations

Translation

Translation is a very simple transformation that adds the translation vector $\vec{p}$ to the position vector $\vec{r}$ of the point to be transformed:

$$\vec{r}\,' = \vec{r} + \vec{p}. \tag{5.21}$$

Scaling along the coordinate axes

Scaling modifies the distances and the size of the object independently along the three coordinate axes. If a point originally has $[x, y, z]$ coordinates, for example, after scaling the respective coordinates are:

$$x' = S_x \cdot x, \qquad y' = S_y \cdot y, \qquad z' = S_z \cdot z. \tag{5.22}$$

This transformation can also be expressed by a matrix multiplication:

$$\vec{r}\,' = \vec{r} \cdot \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & S_z \end{bmatrix}. \tag{5.23}$$

Rotation around the coordinate axes

Rotating around the z axis by an angle ϕ, the x and y coordinates of a point are transformed according to figure 5.3, leaving coordinate z unaffected.

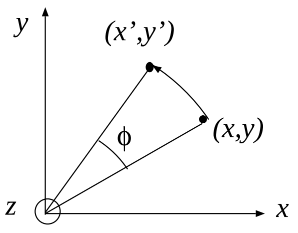

Figure 5.3: Rotation around the z axis

By geometric considerations, the new x, y coordinates can be expressed as:

$$x' = x \cdot \cos\phi - y \cdot \sin\phi, \qquad y' = x \cdot \sin\phi + y \cdot \cos\phi. \tag{5.24}$$

Rotations around the y and x axes have similar form, just the roles of x, y and z must be exchanged. These formulae can also be expressed in matrix form:

$$\vec{r}\,'(x,\phi) = \vec{r}\cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix}$$

$$\vec{r}\,'(y,\phi) = \vec{r}\cdot \begin{bmatrix} \cos\phi & 0 & -\sin\phi \\ 0 & 1 & 0 \\ \sin\phi & 0 & \cos\phi \end{bmatrix} \tag{5.25}$$

$$\vec{r}\,'(z,\phi) = \vec{r}\cdot \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

These rotations can be used to express any orientation [Lan91]. Suppose that K and K''' are two Cartesian coordinate systems sharing a common origin but having different orientations. In order to determine three special rotations around the coordinate axes which transform K into K''', let us define a new Cartesian system K' such that its z' axis is coincident with z and its y' axis is on the intersection line of planes $[x, y]$ and $[x''', y''']$. To transform axis y onto axis y' a rotation is needed around z by angle α. Then a new rotation around y' by angle β has to be applied that transforms x' into x''' resulting in a coordinate system K''. Finally the coordinate system K'' is rotated around axis $x'' = x'''$ by an angle γ to transform y'' into y'''.

The three angles, defining the final orientation, are called **roll, pitch and yaw angles**. If the roll, pitch and yaw angles are α, β and γ respectively, the transformation to the new orientation is:

$$\vec{r}\,' = \vec{r}\cdot \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\beta & 0 & -\sin\beta \\ 0 & 1 & 0 \\ \sin\beta & 0 & \cos\beta \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & \sin\gamma \\ 0 & -\sin\gamma & \cos\gamma \end{bmatrix}. \tag{5.26}$$

Rotation around an arbitrary axis

Let us examine a linear transformation that corresponds to a rotation by angle ϕ around an arbitrary unit axis $\vec{t}$ going through the origin. The original and the transformed points are denoted by vectors $\vec{u}$ and $\vec{v}$ respectively.

Let us decompose vectors $\vec{u}$ and $\vec{v}$ into perpendicular $(\vec{u}_\perp, \vec{v}_\perp)$ and parallel $(\vec{u}_{||}, \vec{v}_{||})$ components with respect to $\vec{t}$. By geometrical considerations we can write:

$$\begin{aligned} \vec{u}_{||} &= \vec{t}(\vec{t}\cdot\vec{u}) \\ \vec{u}_\perp &= \vec{u} - \vec{u}_{||} = \vec{u} - \vec{t}(\vec{t}\cdot\vec{u}) \end{aligned} \tag{5.27}$$

Since the rotation does not affect the parallel component, $\vec{v}_{||} = \vec{u}_{||}$.

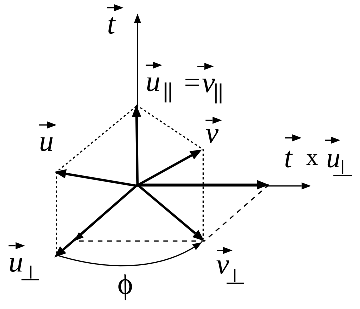

Figure 5.4: Rotating around $\vec{t}$ by angle ϕ

Since vectors $\vec{u}_\perp, \vec{v}_\perp$ and $\vec{t}\times\vec{u}_\perp = \vec{t}\times\vec{u}$ are in the plane perpendicular to $\vec{t}$, and $\vec{u}_\perp$ and $\vec{t}\times\vec{u}_\perp$ are perpendicular vectors (figure 5.4), $\vec{v}_\perp$ can be expressed as:

$$\vec{v}_\perp = \vec{u}_\perp \cdot \cos\phi + \vec{t}\times\vec{u}_\perp \cdot \sin\phi. \tag{5.28}$$

Vector $\vec{v}$, that is the rotation of $\vec{u}$, can then be expressed as follows:

$$\vec{v} = \vec{v}_{||} + \vec{v}_\perp = \vec{u}\cdot\cos\phi + \vec{t}\times\vec{u}\cdot\sin\phi + \vec{t}(\vec{t}\cdot\vec{u})(1-\cos\phi). \tag{5.29}$$

This equation, also called the **Rodrigues formula**, can also be expressed in matrix form. Denoting $\cos\phi$ and $\sin\phi$ by C_ϕ and S_ϕ respectively and assuming $\vec{t}$ to be a unit vector, we get:

$$\vec{v} = \vec{u}\cdot\begin{bmatrix} C_\phi(1-t_x^2)+t_x^2 & t_xt_y(1-C_\phi)+S_\phi t_z & t_xt_z(1-C_\phi)-S_\phi t_y \\ t_yt_x(1-C_\phi)-S_\phi t_z & C_\phi(1-t_y^2)+t_y^2 & t_xt_z(1-C_\phi)+S_\phi t_x \\ t_zt_x(1-C_\phi)+S_\phi t_y & t_zt_y(1-C_\phi)-S_\phi t_x & C_\phi(1-t_z^2)+t_z^2 \end{bmatrix}. \tag{5.30}$$

It is important to note that any orientation can also be expressed as a rotation around an appropriate axis. Thus, there is a correspondence between roll-pitch-yaw angles and the axis and angle of final rotation, which can be given by making the two transformation matrices defined in equations 5.26 and 5.30 equal and solving the equation for unknown parameters.

Shearing

Suppose a shearing stress acts on a block fixed on the xy face of figure 5.5, deforming the block to a parallepiped. The transformation representing the distortion of the block leaves the z coordinate unaffected, and modifies the x and y coordinates proportionally to the z coordinate.

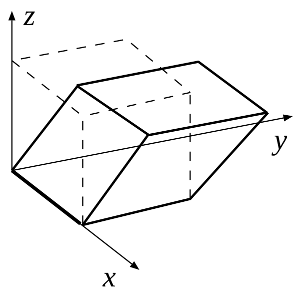

Figure 5.5: Shearing of a block

In matrix form the shearing transformation is:

$$\vec{r}\,' = \vec{r} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ a & b & 1 \end{bmatrix}. \tag{5.31}$$

5.2 Transformation to change the coordinate system

Objects defined in one coordinate system are often needed in another coordinate system. When we decide to work in several coordinate systems and to make every calculation in the coordinate system in which it is the simplest, the coordinate system must be changed for each different phase of the calculation.

Suppose unit coordinate vectors $\vec{u}$, $\vec{v}$ and $\vec{w}$ and the origin $\vec{o}$ of the new coordinate system are defined in the original x, y, z coordinate system:

$$\vec{u} = [u_x, u_y, u_z], \quad \vec{v} = [v_x, v_y, v_z], \quad \vec{w} = [w_x, w_y, w_z], \quad \vec{o} = [o_x, o_y, o_z]. \tag{5.32}$$

Let a point $\vec{p}$ have x, y, z and α, β, γ coordinates in the x, y, z and in the u, v, w coordinate systems respectively. Since the coordinate vectors $\vec{u}, \vec{v}, \vec{w}$ as well as their origin, $\vec{o}$, are known in the x, y, z coordinate system, $\vec{p}$ can be expressed in two different forms:

$$\vec{p} = \alpha \cdot \vec{u} + \beta \cdot \vec{v} + \gamma \cdot \vec{w} + \vec{o} = [x, y, z]. \tag{5.33}$$

This equation can also be written in homogeneous matrix form, having introduced the matrix formed by the coordinates of the vectors defining the u, v, w coordinate system:

$$\mathbf{T_c} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ o_x & o_y & o_z & 1 \end{bmatrix}, \tag{5.34}$$

$$[x, y, z, 1] = [\alpha, \beta, \gamma, 1] \cdot \mathbf{T_c}. \tag{5.35}$$

Since $\mathbf{T_c}$ is always invertible, the coordinates of a point of the x, y, z coordinate system can be expressed in the u, v, w coordinate system as well:

$$[\alpha, \beta, \gamma, 1] = [x, y, z, 1] \cdot \mathbf{T_c}^{-1}. \tag{5.36}$$

Note that the inversion of matrix $\mathbf{T_c}$ can be calculated quite effectively since its upper-left minor matrix is orthonormal, that is, its inverse is given by mirroring the matrix elements onto the diagonal of the matrix, thus:

$$\mathbf{T_c^{-1}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -o_x & -o_y & -o_z & 1 \end{bmatrix} \cdot \begin{bmatrix} u_x & v_x & w_x & 0 \\ u_y & v_y & w_y & 0 \\ u_z & v_z & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{5.37}$$

5.3 Definition of the camera

Having defined transformation matrices we can now look at their use in image generation, but first some basic definitions.

In 3D image generation, a **window** rectangle is placed into the 3D space of the virtual world with arbitrary orientation, a **camera** or **eye** is put behind the window, and a photo is taken by projecting the model onto the window plane, supposing the camera to be the center of projection, and ignoring the parts mapped outside the window rectangle or those which are not in the specified region in front of the camera. The data, which define how the virtual world is looked at, are called **camera parameters**, and include:

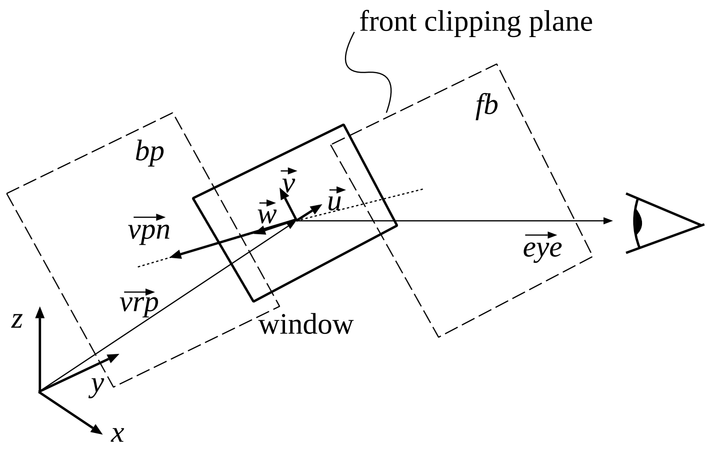

Figure 5.6: Definition of the camera

- **Position and orientation of the window**. The center of the window, called the **view reference point**, is defined as a point, or a vector $v\vec{r}p$, in the world coordinate system. The orientation is defined by a u, v, w orthogonal coordinate system, which is also called the **window coordinate system**, centered at the view reference point, with $\vec{u}$ and $\vec{v}$ specifying the direction of the horizontal and vertical sides of the window rectangle, and $\vec{w}$ determining the normal of the plane of the window. Unit coordinate vectors $\vec{u}, \vec{v}, \vec{w}$ are obviously not independent, because each of them is perpendicular to the other two, thus that dependence has also to be taken care of during the setting of camera parameters. To ease the parameter setting phase, instead of specifying the coordinate vector triple, two almost independent vectors are used for the definition of the orientation, which are the normal vector to the plane of the window, called the **view plane normal**, or $v\vec{p}n$ for short, and a so-called **view up vector**, or $v\vec{u}p$, whose component that is perpendicular to the normal and is in the plane of $v\vec{p}n$ and $v\vec{u}p$ defines the direction of the vertical edge of the window. There is a slight dependence between them, since they should not be parallel, that is, it must always hold that $v\vec{u}p \times v\vec{p}n \neq 0$. The $\vec{u}, \vec{v}, \vec{w}$ coordinate vectors can easily be calculated from the view plane normal and the view up vectors:

$$\vec{w} = \frac{v\vec{p}n}{|v\vec{p}n|}, \qquad \vec{u} = \frac{\vec{w} \times v\vec{u}p}{|\vec{w} \times v\vec{u}p|}, \qquad \vec{v} = \vec{u} \times \vec{w}. \tag{5.38}$$

Note that unlike the x, y, z world coordinate system, the u, v, w system has been defined left handed to meet the user's expectations that $\vec{u}$ points to the right, $\vec{v}$ points upwards and $\vec{w}$ points away from the camera located behind the window.

- **Size of the window**. The length of the edges of the window rectangle are defined by two positive numbers, the width by $wwidth$, the height by $wheight$. Photographic operations, such as zooming in and out, can be realized by proper control of the size of the window. To avoid distortions, the width/height ratio has to be equal to width/height ratio of the viewport on the screen.

- **Type of projection**. The image is the projection of the virtual world onto the window. Two different types of projection are usually used in computer graphics, the **parallel projection** (if the projectors are parallel), and the **perspective projection** (if all the projectors go through a given point, called the center of projection). Parallel projections are further classified into **orthographic** and **oblique** projections depending on whether or not the projectors are perpendicular to the plane of projection (window plane). The attribute "*oblique*" may also refer to perspective projection if the projector from the center of the window is not perpendicular to the plane of the window. Oblique projections may cause distortion of the image.

- **Location of the camera or eye**. The camera is placed behind the window in our conceptual model. For perspective projection, the camera position is, in fact, the center of projection, which can be defined by a point $\vec{eye}$ in the u, v, w coordinate system. For parallel projection, the direction of the projectors has to be given by the u, v, w coordinates of the direction vector. Both in parallel and perspective projections the depth coordinate w is required to be negative in order to place the camera "behind" the window. It also makes sense to consider parallel projection as a special perspective projection, when the camera is at an infinite distance from the window.

- **Front and back clipping planes**. According to the conceptual model of taking photos of the virtual world, it is obvious that only those portions of the model which lie in the infinite pyramid defined by the camera as the apex, and the sides of the 3D window (for perspective projection), and in a half-open, infinite parallelepiped (for parallel projection) can affect the photo. These infinite regions are usually limited to a finite frustum of a pyramid, or to a finite parallelepiped respectively, to avoid overflows and also to ease the projection task by eliminating the parts located behind the camera, by defining two clipping planes called the **front clipping plane** and the **back clipping plane**. These planes are parallel with the window and thus have constant w coordinates appropriate for the definition. Thus the front plane is specified by an fp value, meaning the plane $w = fp$, and the back plane is defined by a bp value. Considering the objectives of the clipping planes, their w coordinates have to be greater than the w coordinate of the eye, and $fp < bp$ should also hold.

5.4 Viewing transformation

Image generation involves:

1. the projection of the virtual world onto the window rectangle,
2. the determination of the closest surface at each point (visibility calculation) by depth comparisons if more than one surface can be projected onto the same point in the window, and
3. the placement of the result in the viewport rectangle of the screen.

Obviously, the visibility calculation has to be done prior to the projection of the 3D space onto the 2D window rectangle, since this projection destroys the depth information.

These calculations could also be done in the world coordinate system, but each projection would require the evaluation of the intersection of an arbitrary line and rectangle (window), and the visibility problem would require the determination of the distance of the surface points along the projectors. The large number of multiplications and divisions required by such geometric computations makes the selection of the world coordinate system disadvantageous even if the required calculations can be reduced by the application of the incremental concept, and forces us to look for other coordinate systems where these computations are simple and effective to perform.

In the optimal case the points should be transformed to a coordinate system where X, Y coordinates would represent the pixel location through which the given point is visible, and a third Z coordinate could be used to decide which point is visible, i.e. closest to the eye, if several points could be transformed to the same X, Y pixel. Note that Z is not necessarily proportional to the distance from the eye, it should only be a monotonously increasing function of the distance. The appropriate transformation is also expected to map lines onto lines and planes onto planes, allowing simple representations and linear interpolations during clipping and visibility calculations. Coordinate systems meeting all the above requirements are called **screen coordinate systems**. In a coordinate system of this type, the visibility calculations are simple, since should two or more points have the same X, Y pixel coordinates, then the visible one has the smallest Z coordinate.

From a different perspective, if it has to be decided whether one point will hide another, two comparisons are needed to check whether they project onto the same pixel, that is, whether they have the same X, Y coordinates, and a third comparison must be used to select the closest. The projection is very simple, because the projected point has, in fact, X, Y coordinates due to the definition of the screen space.

For pedagogical reasons, the complete transformation is defined through several intermediate coordinate systems, although eventually it can be accomplished by a single matrix multiplication. For both parallel and perspective cases, the first step of the transformation is to change the coordinate system to u, v, w from x, y, z, but after that there will be differences depending on the projection type.

5.4.1 World to window coordinate system transformation

First, the world is transformed to the u, v, w coordinate system fixed to the center of the window. Since the coordinate vectors $\vec{u}$, $\vec{v}$, $\vec{w}$ and the origin $\vec{vrp}$ are defined in the x, y, z coordinate system, the necessary transformation can be developed based on the results of section 5.2 of this chapter. The matrix formed by the coordinates of the vectors defining the u, v, w coordinate system is:

$$\mathbf{T_{uvw}} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ vrp_x & vrp_y & vrp_z & 1 \end{bmatrix}, \tag{5.39}$$

$$[x, y, z, 1] = [\alpha, \beta, \gamma, 1] \cdot \mathbf{T_{uvw}}. \tag{5.40}$$

Since $\vec{u}$, $\vec{v}$, $\vec{w}$ are perpendicular vectors, $\mathbf{T_{uvw}}$ is always invertible. Thus, the coordinates of an arbitrary point of the world coordinate system can be expressed in the u, v, w coordinate system as well:

$$[\alpha, \beta, \gamma, 1] = [x, y, z, 1] \cdot \mathbf{T_{uvw}^{-1}}. \tag{5.41}$$

5.4.2 Window to screen coordinate system transformation for parallel projection

Shearing transformation

For oblique transformations, that is when eye_u or eye_v is not zero, the projectors are not perpendicular to the window plane, thus complicating visibility calculations and projection (figure 5.7). This problem can be solved by distortion of the object space, applying a **shearing transformation** in such a way that the non-oblique projection of the distorted objects should provide the same images as the oblique projection of the original scene, and the depth coordinate of the points should not be affected. A general

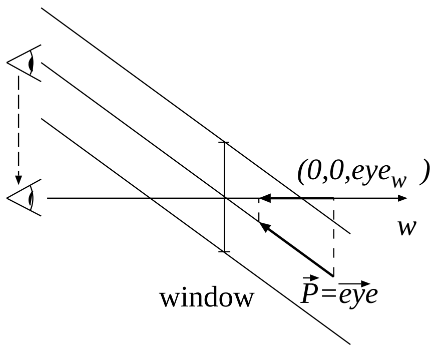

Figure 5.7: Shearing

shearing transformation which does not affect the w coordinate is:

$$\mathbf{T_{shear}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ s_u & s_v & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \qquad (5.42)$$

The unknown elements, s_u and s_v, can be determined by examining the transformation of the projector $\vec{P} = [eye_u, eye_v, eye_w, 1]$. The transformed projector is expected to be perpendicular to the window and to have depth coordinate eye_w, that is:

$$\vec{P} \cdot \mathbf{T_{shear}} = [0, 0, eye_w, 1]. \qquad (5.43)$$

Using the definition of the shearing transformation, we get:

$$s_u = -\frac{eye_u}{eye_w}, \qquad s_v = -\frac{eye_v}{eye_w}. \tag{5.44}$$

Normalizing transformation

Having accomplished the shearing transformation, the objects for parallel projection are in a space shown in figure 5.8. The subspace which can be projected onto the window is a rectangular box between the front and back clipping plane, having side faces coincident to the edges of the window. To allow uniform treatment, a normalizing transformation can be applied, which maps the box onto a normalized block, called the **canonical view volume**, moving the front clipping plane to 0, the back clipping plane to 1, the other boundaries to $x = 1$, $y = 1$, $x = -1$ and $y = -1$ planes respectively.

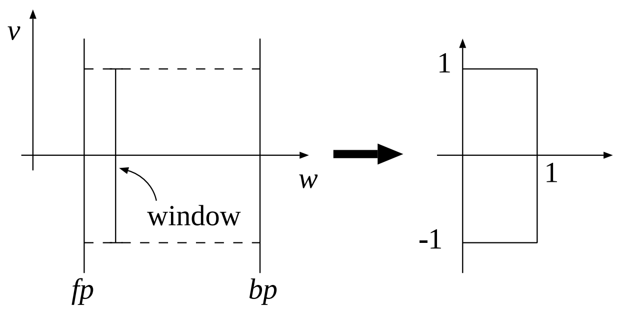

Figure 5.8: Normalizing transformation for parallel projection

The normalizing transformation can also be expressed in matrix form:

$$\mathbf{T_{norm}} = \begin{bmatrix} 2/wwidth & 0 & 0 & 0 \\ 0 & 2/wheight & 0 & 0 \\ 0 & 0 & 1/(bp-fp) & 0 \\ 0 & 0 & -fp/(bp-fp) & 1 \end{bmatrix}. \tag{5.45}$$

The projection in the canonical view volume is very simple, since the projection does not affect the (X, Y) coordinates of an arbitrary point, but only its depth coordinate.

Viewport transformation

The space inside the clipping volume has been projected onto a 2×2 rectangle. Finally, the image has to be placed into the specified viewport of the screen, defined by the center point, (V_x, V_y) and by the horizontal and vertical sizes, V_{sx} and V_{sy}. For parallel projection, the necessary viewport transformation is:

$$\mathbf{T_{viewport}} = \begin{bmatrix} V_{sx}/2 & 0 & 0 & 0 \\ 0 & V_{sy}/2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ V_x & V_y & 0 & 1 \end{bmatrix}. \tag{5.46}$$

Summarizing the results, the complete viewing transformation for parallel projection can be generated. The screen space coordinates formed by the (X, Y) pixel addresses and the Z depth value mapped into the range of $[0..1]$ can be determined by the following transformation:

$$\begin{aligned} \mathbf{T_V} &= \mathbf{T_{uvw}^{-1}} \cdot \mathbf{T_{shear}} \cdot \mathbf{T_{norm}} \cdot \mathbf{T_{viewport}}, \\ [X, Y, Z, 1] &= [x, y, z, 1] \cdot \mathbf{T_V}. \end{aligned} \tag{5.47}$$

Matrix $\mathbf{T_V}$, called the **viewing transformation**, is the concatenation of the transformations representing the different steps towards the screen coordinate system. Since $\mathbf{T_V}$ is affine, it obviously meets the requirements of preserving lines and planes, making both the visibility calculation and the projection easy to accomplish.

5.4.3 Window to screen coordinate system transformation for perspective projection

As in the case of parallel projection, objects are first transformed from the world coordinate system to the window, that is u, v, w, coordinate system by applying $\mathbf{T_{uvw}^{-1}}$.

View-eye transformation

For perspective projection, the center of the u, v, w coordinate system is translated to the camera position without altering the direction of the axes.

Since the camera is defined in the u, v, w coordinate system by a vector $\vec{eye}$, this transformation is a translation by vector $-\vec{eye}$, which can also be expressed by a homogeneous matrix:

$$\mathbf{T}_{\mathbf{eye}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -eye_u & -eye_v & -eye_w & 1 \end{bmatrix}. \tag{5.48}$$

Shearing transformation

As for parallel projection, if eye_u or eye_v is not zero, the projector from the center of the window is not perpendicular to the window plane, requiring the distortion of the object space by a **shearing transformation** in such a way that the non-oblique projection of the distorted objects provides the same images as the oblique projection of the original scene and the depth coordinate of the points is not affected. Since the projector from the center of the window ($\vec{P} = [eye_u, eye_v, eye_w, 1]$) is the same as all the projectors for parallel transformation, the shearing transformation matrix will have the same form, independently of the projection type:

$$\mathbf{T}_{\mathbf{shear}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -eye_u/eye_w & -eye_v/eye_w & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{5.49}$$

Normalizing transformation

After shearing transformation the region which can be projected onto the window is a symmetrical, finite frustum of the pyramid in figure 5.9. By normalizing this pyramid, the back clipping plane is moved to 1, and the angle at its apex is set to 90 degrees. This is a simple scaling transformation, with scales S_u, S_v and S_w determined by the consideration that the back clipping plane goes to $w = 1$, and the window goes to the position d which is equal to half the height and half the width of the normalized window:

$$S_u \cdot wwidth/2 = d, \quad S_v \cdot wheight/2 = d, \quad eye_w \cdot S_w = d, \quad S_w \cdot bp = 1 \tag{5.50}$$

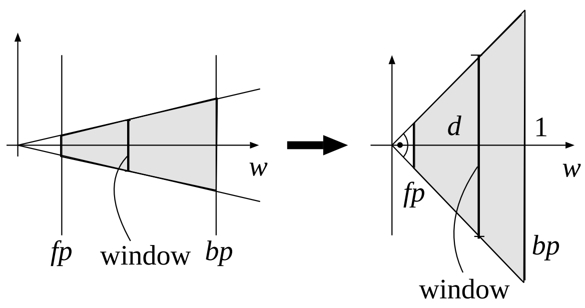

Figure 5.9: Normalizing transformation for perspective projection

Solving these equations and expressing the transformation in a homogeneous matrix form, we get:

$$\mathbf{T_{norm}} = \begin{bmatrix} 2 \cdot eye_w/(wwidth \cdot bp) & 0 & 0 & 0 \\ 0 & 2 \cdot eye_w/(wheight \cdot bp) & 0 & 0 \\ 0 & 0 & 1/bp & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{5.51}$$

In the canonical view volume, the central **projection** of a point X_c, Y_c, Z_c onto the window plane is:

$$X_p = d \cdot \frac{X_c}{Z_c}, \qquad Y_p = d \cdot \frac{Y_c}{Z_c}. \tag{5.52}$$

Perspective transformation

The projection and the visibility calculations are more difficult in the canonical view volume for central projection than they are for parallel projection because of the division required by the projection. When calculating visibility, it has to be decided if one point (X_c^1, Y_c^1, Z_c^1) hides another point (X_c^2, Y_c^2, Z_c^2). This involves the check for relations

$$[X_c^1/Z_c^1, Y_c^1/Z_c^1] = [X_c^2/Z_c^2, Y_c^2/Z_c^2] \quad \text{and} \quad Z_c^1 < Z_c^2$$

which requires division in a way that the visibility check for parallel projection does not. To avoid division during the visibility calculation, a transformation is needed which transforms the canonical view volume to meet the requirements of the screen coordinate systems, that is, X and Y coordinates are the pixel addresses in which the point is visible, and Z is a monotonous function of the original distance from the camera (see figure 5.10).

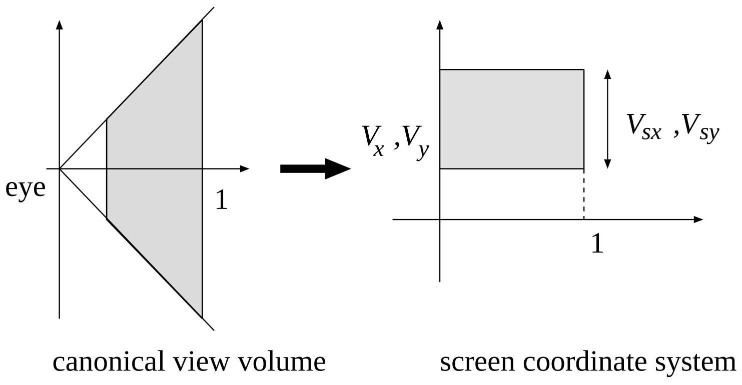

Figure 5.10: Canonical view volume to screen coordinate system transformation

Considering the expectations for the X and Y coordinates:

$$X = \frac{X_c}{Z_c} \cdot \frac{V_{sx}}{2} + V_x, \qquad Y = \frac{Y_c}{Z_c} \cdot \frac{V_{sy}}{2} + V_y. \tag{5.53}$$

The unknown function $Z(Z_c)$ can be determined by forcing the transformation to preserve planes and lines. Suppose a set of points of the canonical view volume are on a plane with the equation:

$$a \cdot X_c + b \cdot Y_c + c \cdot Z_c + d = 0 \tag{5.54}$$

The transformation of this set is also expected to lie in a plane, that is, there are parameters $a', b,' c,' d'$ satisfying the equation of the plane for transformed points:

$$a' \cdot X + b' \cdot Y + c' \cdot Z + d' = 0 \tag{5.55}$$

Inserting formula 5.53 into this plane equation and multiplying both sides by Z_c, we get:

$$a' \cdot \frac{V_{sx}}{2} \cdot X_c + b' \cdot \frac{V_{sy}}{2} \cdot Y_c + c' \cdot Z(Z_c) \cdot Z_c + (a' \cdot V_x + b' \cdot V_y + d') \cdot Z_c = 0 \tag{5.56}$$

Comparing this with equation 5.54, we can conclude that both $Z(Z_c) \cdot Z_c$ and Z_c are linear functions of X_c and Y_c, requiring $Z(Z_c) \cdot Z_c$ to be a linear function of Z_c also. Consequently:

$$Z(Z_c) \cdot Z_c = \alpha \cdot Z_c + \beta \quad \Longrightarrow \quad Z(Z_c) = \alpha + \frac{\beta}{Z_c}. \tag{5.57}$$

Unknown parameters α and β are set to map the front clipping plane of the canonical view volume ($fp' = fp/bp$) to 0 and the back clipping plane (1) to 1:

$$\begin{array}{cc} \alpha \cdot fp' + \beta = 0, & \alpha \cdot 1 + \beta = 1 \\ & \Downarrow \\ \alpha = bp/(bp - fp), & \beta = -fp/(bp - fp) \end{array} \tag{5.58}$$

The complete transformation, called the **perspective transformation**, is:

$$X = \frac{X_c}{Z_c} \cdot \frac{V_{sx}}{2} + V_x, \qquad Y = \frac{Y_c}{Z_c} \cdot \frac{V_{sy}}{2} + V_y, \qquad Z = \frac{Z_c \cdot bp - fp}{(bp - fp) \cdot Z_c}. \tag{5.59}$$

Examining equation 5.59, we can see that $X \cdot Z_c$, $Y \cdot Z_c$ and $Z \cdot Z_c$ can be expressed as a linear transformation of X_c, Y_c, Z_c, that is, in homogeneous coordinates $[X_h, Y_h, Z_h, h] = [X \cdot Z_c, Y \cdot Z_c, Z \cdot Z_c, Z_c]$ can be calculated with a single matrix product by $\mathbf{T_{persp}}$:

$$\mathbf{T_{persp}} = \begin{bmatrix} V_{sx}/2 & 0 & 0 & 0 \\ 0 & V_{sy}/2 & 0 & 0 \\ V_x & V_y & bp/(bp - fp) & 1 \\ 0 & 0 & -fp/(bp - fp) & 0 \end{bmatrix}. \tag{5.60}$$

The complete perspective transformation, involving homogeneous division to get real 3D coordinates, is:

$$[X_h, Y_h, Z_h, h] = [X_c, Y_c, Z_c, 1] \cdot \mathbf{T_{persp}},$$

$$[X, Y, Z, 1] = [\frac{X_h}{h}, \frac{Y_h}{h}, \frac{Z_h}{h}, 1]. \tag{5.61}$$

The division by coordinate h is meaningful only if $h \neq 0$. Note that the complete transformation is a homogeneous linear transformation which consists of a matrix multiplication and a homogeneous division to convert the homogeneous coordinates back to Cartesian ones.

This is not at all surprising, since one reason for the emergence of projective geometry has been the need to handle central projection somehow by linear means. In fact, the result of equation 5.61 could have been derived easily if it had been realized first that a homogeneous linear transformation would solve the problem (figure 5.10). This transformation would transform the eye onto an ideal point and make the side faces of the viewing pyramid parallel. Using homogeneous coordinates this transformation means that:

$$\mathcal{T}: \quad [0,0,0,1] \mapsto \lambda_1[0,0,-1,0]. \tag{5.62}$$

Multiplicative factor λ_1 indicates that all homogeneous points differing by a scalar factor are equivalent. In addition, the corner points where the side faces and the back clipping plane meet should be mapped onto the corner points of the viewport rectangle on the $Z = 1$ plane and the front clipping plane must be moved to the origin, thus:

$$\begin{array}{lll}
\mathcal{T}: & [1,1,1,1] & \mapsto & \lambda_2[V_x + V_{sx}/2, V_y + V_{sy}/2, 1, 1], \\
\mathcal{T}: & [1,-1,1,1] & \mapsto & \lambda_3[V_x + V_{sx}/2, V_y - V_{sy}/2, 1, 1], \\
\mathcal{T}: & [-1,1,1,1] & \mapsto & \lambda_4[V_x - V_{sx}/2, V_y + V_{sy}/2, 1, 1], \\
\mathcal{T}: & [0,0,fp',1] & \mapsto & \lambda_5[V_x, V_y, 0, 1].
\end{array} \tag{5.63}$$

Transformation $\mathcal{T}$ is defined by a matrix multiplication with $\mathbf{T}_{4\times4}$. Its unknown elements can be determined by solving the linear system of equations generated by equations 5.62 and 5.63. The problem is not determinant since the number of equations (20) is one less than the number of variables (21). In fact, it is natural, since scalar multiples of homogeneous matrices are equivalent. By setting λ_2 to 1, however, the problem will be determinant and the resulting matrix will be the same as derived in equation 5.60.

As has been proven, homogeneous transformation preserves linear sets such as lines and planes, thus deriving this transformation from the requirement that it should preserve planes also guaranteed the preservation of lines.

However, when working with finite structures, such as line segments, polygons, convex hulls, etc., homogeneous transformations can cause serious problems if the transformed objects intersect the $h = 0$ hyperplane. (Note that the preservation of convex hulls could be proven for only those cases when the image of transformation has no such intersection.)

To demonstrate this problem and how perspective transformation works, consider an example when $V_x = V_y = 0, V_{sx} = V_{sy} = 2, fp = 0.5, bp = 1$ and

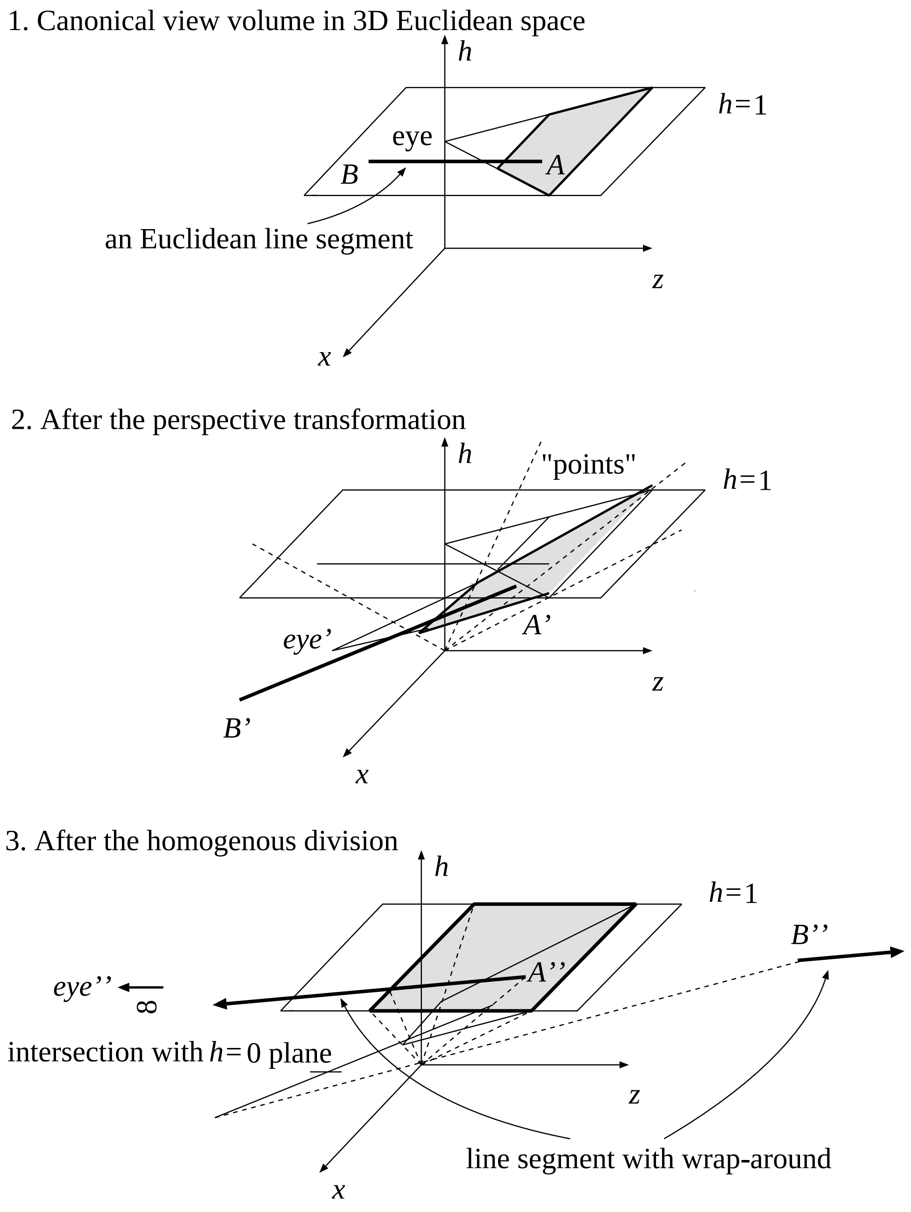
1. Canonical view volume in 3D Euclidean space
h
h=1
eye
B
A
an Euclidean line segment
z
x
2. After the perspective transformation
h
"points"
h=1
A'
eye'
B'
z
x
3. After the homogenous division
h
h=1
B''
eye''
∞
A''
intersection with h=0 plane
z
line segment with wrap-around
x

Figure 5.11: Steps of the perspective transformation and the wrap-around problem

examine what happens with the clipping region and with a line segment defined by endpoints [0.3,0,0.6] and [0.3,0,-0.6] in the Cartesian coordinate system (see figure 5.11). This line segment starts in front of the eye and goes behind it. When the homogeneous representation of this line is transformed by multiplying the perspective transformation matrix, the line will intersect the $h = 0$ plane, since originally it intersects the $Z_c = 0$ plane (which is parallel with the window and contains the eye) and the matrix multiplication sets $h = Z_c$. Recall that the $h = 0$ plane corresponds to the ideal points in the straight model, which have no equivalent in Euclidean geometry.

The conversion of the homogeneous coordinates to Cartesian ones by homogeneous division maps the upper part corresponding to positive h values onto a Euclidean half-line and maps the lower part corresponding to negative h values onto another half-line. This means that the line segment falls into two half-lines, a phenomenon which is usually referred to as the **wrap-around problem**.

Line segments are identified by their two endpoints in computer graphics. If wrap-around phenomena may occur we do not know whether the transformation of the two endpoints really define the new segment, or these are the starting points of two half-lines that form the complement of the Euclidean segment. This is not surprising in projective geometry, since a projective version of a Euclidean line, for example, also includes an ideal point in addition to all affine points, which glues the two "ends" of the line at infinity. From this point of view projective lines are similar (more precisely isomorphic) to circles. As two points on a circle cannot identify an arc unambiguously, two points on a projective line cannot define a segment either without further information. By knowing, however, that the projective line segment does not contain ideal points, this definition is unambiguous.

The elimination of ideal points from the homogeneous representation before homogeneous division obviously solves the problem. Before the homogeneous division, this procedure cuts the objects represented by homogeneous coordinates into two parts corresponding to the positive and negative h values respectively, then projects these parts back to the Cartesian coordinates separately and generates the final representation as the union of the two cases. Recall that a clipping that removes object parts located outside of the viewing pyramid must be accomplished somewhere in the viewing pipeline. The cutting proposed above is worth combining with this clipping step, meaning that the clipping (or at least the so-called depth clipping phase that can remove the vanishing plane which is transformed onto ideal points) must be carried out before homogeneous division. Clipping is accomplished by appropriate algorithms discussed in the next section.

Summarizing the transformation steps of viewing for the perspective case, the complete viewing transformation is:

$$\begin{aligned} \mathbf{T_V} &= \mathbf{T_{uvw}^{-1}} \cdot \mathbf{T_{eye}} \cdot \mathbf{T_{shear}} \cdot \mathbf{T_{norm}} \cdot \mathbf{T_{persp}}, \\ [X_h, Y_h, Z_h, h] &= [x, y, z, 1] \cdot \mathbf{T_V}, \\ [X, Y, Z, 1] &= [\frac{X_h}{h}, \frac{Y_h}{h}, \frac{Z_h}{h}, 1]. \end{aligned} \tag{5.64}$$

5.5 Clipping

Clipping is responsible for eliminating those parts of the scene which do not project onto the window rectangle, because they are outside the viewing volume. It consists of depth — front and back plane — clipping and clipping at the side faces of the volume. For perspective projection, depth clipping is also necessary to solve the wrap-around problem, because it eliminates the objects in the plane parallel to the window and incident to the eye, which are mapped onto the ideal plane by the perspective transformation.

For parallel projection, depth clipping can be accomplished in any coordinate system before the projection, where the depth information is still available. The selection of the coordinate system in which the clipping is done may depend on efficiency considerations, or more precisely:

1. The geometry of the clipping region has to be simple in the selected coordinate system in order to minimize the number of necessary operations.

2. The transformation to the selected coordinate system from the world coordinate system and from the selected coordinate system to pixel space should involve the minimum number of operations.

Considering the first requirement, for parallel projection, the brick shaped canonical view volume of the normalized eye coordinate system and the screen coordinate system are the best, but, unlike the screen coordinate system, the normalized eye coordinate system requires a new transformation after clipping to get to pixel space. The screen coordinate system thus ranks as the better option. Similarly, for perspective projection, the pyramid shaped canonical view volume of the normalized eye and the homogeneous coordinate systems require the simplest clipping calculations, but the latter does not require extra transformation before homogeneous division. For side face clipping, the screen coordinate system needs the fewest operations, but separating the depth and side face clipping phases might be disadvantageous for specific hardware realizations. In the next section, the most general case, clipping in homogeneous coordinates, will be discussed. The algorithms for other 3D coordinate systems can be derived from this general case by assuming the homogeneous coordinate h to be constant.

5.5.1 Clipping in homogeneous coordinates

The boundaries of the clipping region can be derived by transforming the requirements of the screen coordinate system to the homogeneous coordinate system. After homogeneous division, in the screen coordinate system the boundaries are $X_{\min} = V_x - V_{sx}/2$, $X_{\max} = V_x + V_{sx}/2$, $Y_{\min} = V_y - V_{sy}/2$ and $Y_{\max} = V_y + V_{sy}/2$. The points internal to the clipping region must satisfy:

$$\begin{aligned} X_{\min} \le X_h/h \le X_{\max}, \\ Y_{\min} \le Y_h/h \le Y_{\max}, \\ 0 \le Z_h/h \le 1 \end{aligned} \tag{5.65}$$

The visible parts of objects defined in an Euclidean world coordinate system must have positive Z_c coordinates in the canonical view coordinate system, that is, they must be in front of the eye. Since multiplication by the perspective transformation matrix sets $h = Z_c$, for visible parts, the fourth homogeneous coordinate must be positive. Adding $h > 0$ to the set of inequalities 5.65 and multiplying both sides by this positive h, an equivalent system of inequalities can be derived as:

$$\begin{aligned} X_{\min} \cdot h \le X_h \le X_{\max} \cdot h, \\ Y_{\min} \cdot h \le Y_h \le Y_{\max} \cdot h, \\ 0 \le Z_h \le h. \end{aligned} \tag{5.66}$$

Note that inequality $h > 0$ does not explicitly appear in the requirements, since it comes from $0 \le Z_h \le h$. Inequality $h > 0$, on the other hand, guarantees that all points are eliminated that are on the $h = 0$ ideal plane, which solves the wrap-around problem.

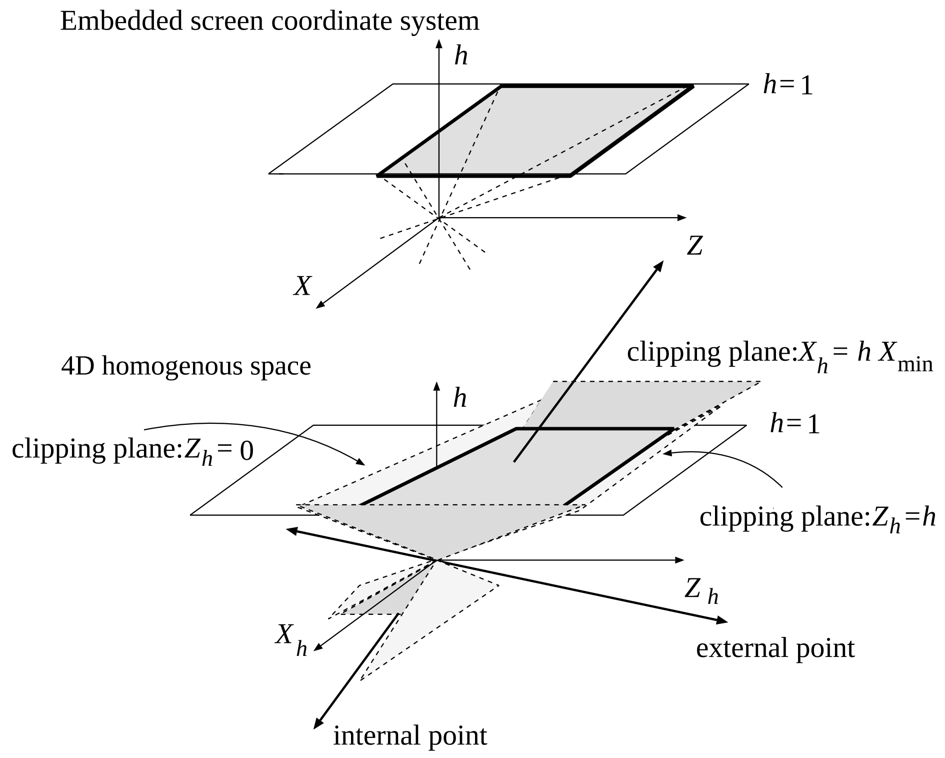

Figure 5.12: Transforming the clipping region back to projective space

Notice that the derivation of the homogeneous form of clipping has been achieved by transforming the clipping box defined in the screen coordinate system back to the projective space represented by homogeneous coordinates (figure 5.12).

When the definition of the clipping region was elaborated, we supposed that the objects are defined in a Cartesian coordinate system and relied on the camera construction discussed in section 5.3. There are fields of computer graphics, however, where none of these is true. Sometimes it is more convenient to define the objects directly in the projective space by homogeneous coordinates. A rational B-spline, for example, can be defined as a non-rational B-spline in homogeneous space, since the homogeneous to Cartesian mapping will carry out the division automatically. When dealing with homogeneous coordinates directly, scalar multiples of the coordinates are equivalent, thus both positive and negative h regions can contribute to the visible section of the final space. Thus, equation 5.65 must be converted to two system of inequalities, one supposing $h > 0$, the other $h < 0$.

$$\begin{array}{cc} \textbf{Case 1: } h > 0 & \textbf{Case 2: } h < 0 \\ X_{\min} \cdot h \leq X_h \leq X_{\max} \cdot h & X_{\min} \cdot h \geq X_h \geq X_{\max} \cdot h \\ Y_{\min} \cdot h \leq Y_h \leq Y_{\max} \cdot h & Y_{\min} \cdot h \geq Y_h \geq Y_{\max} \cdot h \\ 0 \leq Z_h \leq h & 0 \geq Z_h \geq h \end{array} \tag{5.67}$$

Clipping must be carried out for the two regions separately. After homogeneous division these two parts will meet in the screen coordinate system.

Even this formulation — which defined a front clipping plane in front of the eye to remove points in the vanishing plane — may not be general enough for systems where the clipping region is independent of the viewing transformation like in PHIGS [ISO90]. In the more general case the image of the clipping box in the homogeneous space may have intersection with the ideal plane, which can cause wrap-around. The basic idea remains the same in the general case; we must get rid of ideal points by some kind of clipping. The interested reader is referred to the detailed discussion of this approach in [Kra89],[Her91].

Now the clipping step is investigated in detail. Let us assume that the clipping region is defined by equation 5.66 (the more general case of equation 5.67 can be handled similarly by carrying out two clipping procedures).

Based on equation 5.66 the clipping of points is very simple, since their homogeneous coordinates must be examined to see if they satisfy all the equations. For more complex primitives, such as line segments and planar polygons, the intersection of the primitive and the planes bounding the clipping region has to be calculated, and that part of the primitive should be preserved where all points satisfy equation 5.66. The intersection calculation of bounding planes with line segments and planar polygons requires the solution of a linear equation involving multiplications and divisions. The case when there is no intersection happens when the solution for a parameter is outside the range of the primitive. The number of divisions and multiplications necessary can be reduced by eliminating those primitive-plane intersection calculations which do not provide intersection, assuming that there is a simple way to decide which these are. Clipping algorithms contain special geometric considerations to decide if there might be an intersection without solving the linear equation.

Clipping of line segments

One of the simplest algorithms for clipping line segments with fewer intersection calculations is the 3D extension of the Cohen and Sutherland clipping algorithm.

Each bounding plane of the clipping region divides the 3D space into two half-spaces. Points in 3D space can be characterized by a 6-bit code, where each bit corresponds to a respective plane defining whether the given point and the convex view volume are on opposite sides of the plane by 1 (or true) value, or on the same side of the plane, by 0 (or false) value. Formally the code bits $C[0] \ldots C[5]$ of a point are defined by:

$$
\begin{aligned}
C[0] &= \begin{cases} 1 \text{ if } X_h < X_{\min} \cdot h \\ 0 \text{ otherwise} \end{cases} & C[1] &= \begin{cases} 1 \text{ if } X_h > X_{\max} \cdot h \\ 0 \text{ otherwise} \end{cases} \\
C[2] &= \begin{cases} 1 \text{ if } Y_h < Y_{\min} \cdot h \\ 0 \text{ otherwise} \end{cases} & C[3] &= \begin{cases} 1 \text{ if } Y_h > Y_{\max} \cdot h \\ 0 \text{ otherwise} \end{cases} \\
C[4] &= \begin{cases} 1 \text{ if } Z_h < 0 \\ 0 \text{ otherwise} \end{cases} & C[5] &= \begin{cases} 1 \text{ if } Z_h > h \\ 0 \text{ otherwise} \end{cases}
\end{aligned}
\tag{5.68}
$$

Obviously, points coded by 000000 have to be preserved, while all other codes correspond to regions outside the view volume (figure 5.13).

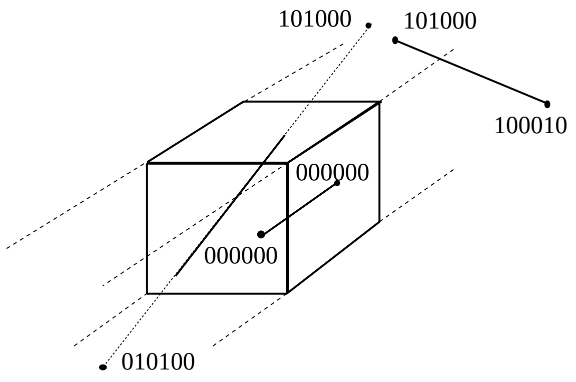

Figure 5.13: Clipping of line segments

Let the codes of the two endpoints of a line segment be C_1 and C_2 respectively. If both C_1 and C_2 are zero, the endpoints, as well as all inner points of the line segment, are inside the view volume, thus the whole line segment has to be preserved by clipping. If the corresponding bits of both C_1 and C_2 are non-zero at some position, then the endpoints, and the inner points too, are on the same side of the respective bounding plane, external to the view volume, thus requiring the whole line segment to be eliminated by clipping. These are the trivial cases where clipping can be accomplished without any intersection calculation.

If this is not the case — that is if at least one bit pair in the two codes are not equal, and for those bits where they are the same, they have a value of 0, then the intersection between the line and that plane which corresponds to the bit where the two codes are different has to be calculated, and the part of the line segment which is outside must be eliminated by replacing the endpoint having 1 code bit by the intersection point. Let the two endpoints have coordinates $[X_h^{(1)}, Y_h^{(1)}, Z_h^{(1)}, h^{(1)}]$ and $[X_h^{(2)}, Y_h^{(2)}, Z_h^{(2)}, h^{(2)}]$ respectively. The parametric representation of the line segment, supposing parameter range $[0..1]$ for t, is:

$$\begin{aligned} X_h(t) &= X_h^{(1)} \cdot t + X_h^{(2)} \cdot (1-t) \\ Y_h(t) &= Y_h^{(1)} \cdot t + Y_h^{(2)} \cdot (1-t) \\ Z_h(t) &= Z_h^{(1)} \cdot t + Z_h^{(2)} \cdot (1-t) \\ h(t) &= h^{(1)} \cdot t + h^{(2)} \cdot (1-t) \end{aligned} \tag{5.69}$$

Note that this representation expresses the line segment as a linear set spanned by the two endpoints. Special care has to be taken when the h coordinates of the two endpoints have different sign because this means that the linear set contains an ideal point as well.

Now let us consider the intersection of this line segment with a clipping plane (figure 5.14). If, for example, the code bits are different in the first bit corresponding to $X_{\min}$, then the intersection with the plane $X_h = X_{\min} \cdot h$ has to be calculated thus:

$$X_h^{(1)} \cdot t + X_h^{(2)} \cdot (1-t) = X_{\min} \cdot (h^{(1)} \cdot t + h^{(2)} \cdot (1-t)). \tag{5.70}$$

Solving for parameter t^* of the intersection point, we get:

$$t^* = \frac{X_{\min} \cdot h^{(2)} - X_h^{(2)}}{X_h^{(1)} - X_h^{(2)} - X_{\min} \cdot (h^{(1)} - h^{(2)})}. \tag{5.71}$$

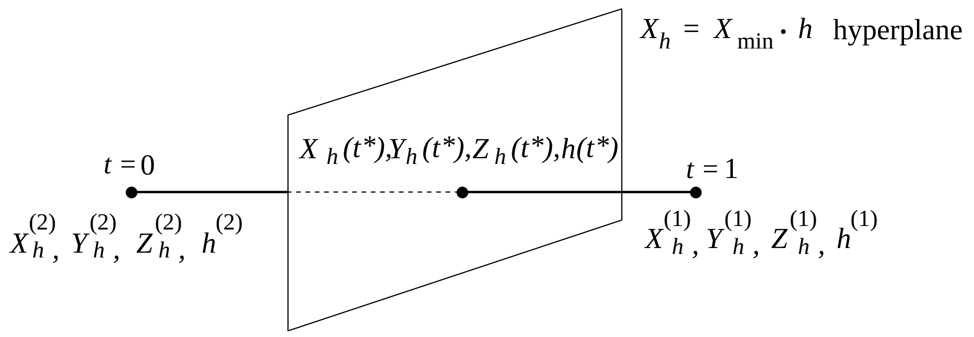
$X_h = X_{\min} \cdot h$ hyperplane
$X_h(t^*), Y_h(t^*), Z_h(t^*), h(t^*)$
$t = 0$
$t = 1$
$X_h^{(2)}, Y_h^{(2)}, Z_h^{(2)}, h^{(2)}$
$X_h^{(1)}, Y_h^{(1)}, Z_h^{(1)}, h^{(1)}$

Figure 5.14: Clipping by a homogeneous plane

Substituting t^* back to the equation of the line segment, the homogeneous coordinates of the intersection point are $[X_h(t^*), Y_h(t^*), Z_h(t^*), h(t^*)]$. For other bounding planes, the algorithm is similar. The steps discussed can be converted into an algorithm which takes and modifies the two endpoints and returns TRUE if some inner section of the line segment is found, and FALSE if the segment is totally outside the viewing volume, thus:

```
LineClipping($P_h^{(1)}$, $P_h^{(2)}$)
    $C_1$ = Calculate code bits for $P_h^{(1)}$;
    $C_2$ = Calculate code bits for $P_h^{(2)}$;
    loop
        if ($C_1 = 0$ AND $C_2 = 0$) then return TRUE;   // Accept
        if ($C_1$ & $C_2 \neq 0$) then return FALSE;     // Reject
        $f$ = Index of clipping face, where bit of $C_1$ differs from $C_2$;
        $P_h^*$ = Intersection of line ($P_h^{(1)}$, $P_h^{(2)}$) and plane $f$;
        $C^*$ = Calculate code bits for $P_h^*$;
        if $C_1[f] = 1$ then $P_h^{(1)} = P_h^*$; $C_1 = C^*$;
        else                 $P_h^{(2)} = P_h^*$; $C_2 = C^*$;
    endloop
```

The previously discussed Cohen–Sutherland algorithm replaces a lot of intersection calculations by simple arithmetics of endpoint codes, increasing the efficiency of clipping, but may still calculate intersections which later turn out to be outside the clipping region. This means that it is not optimal for the number of calculated intersections. Other algorithms make use of a different compromise in the number of intersection calculations and the complexity of other geometric considerations [CB78], [LB84], [Duv90].

Clipping of polygons

Unfortunately, polygons cannot be clipped by simply clipping the edges, because this may generate false results (see figure 5.15). The core of the problem is the fact that the edges of a polygon can go around the faces of the bounding box, and return through a face different from where they left the inner section, or they may not intersect the faces at all, when the polygon itself encloses or is enclosed by the bounding box.

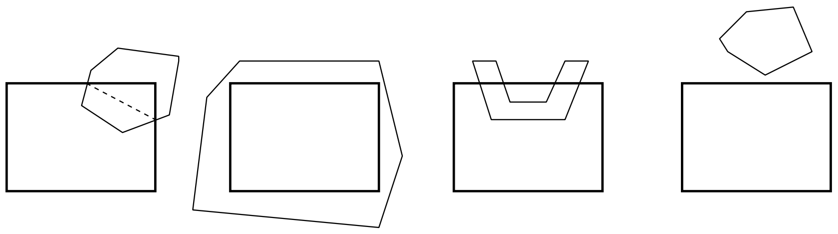

Figure 5.15: Cases of polygon clipping

This problem can be solved if clipping against a bounding box is replaced by six clipping steps to the planes of the faces of the bounding box, as has been proposed for the 2D equivalent of this problem by Hodgman and Sutherland [SH74]. Since planes are infinite objects, polygon edges cannot go around them, and a polygon, clipped against all the six boundary planes, is guaranteed to be inside the view volume.

When clipping against a plane, consecutive vertices have to be examined to determine whether they are on the same side of the plane. If both of them are on the same side of the plane as the region, then the edge is also the edge of the clipped polygon. If both are on the opposite side of the plane from the region, the edge has to be ignored. If one vertex is on the same side and one on the opposite side, the intersection of the edge and the plane has to be calculated, and a new edge formed from that to the point where the polygon returns back through the plane (figure 5.16).

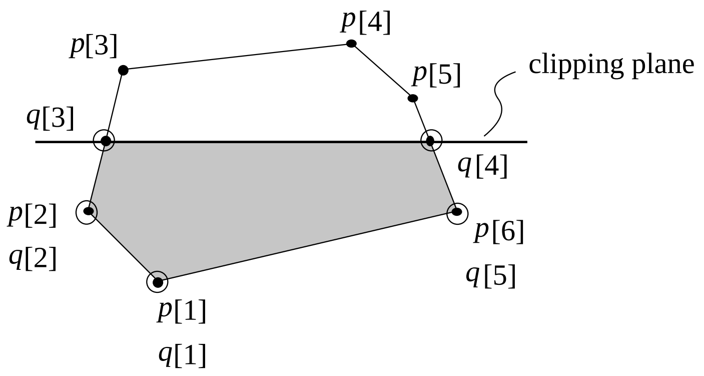

Figure 5.16: Clipping of a polygon against a plane

Suppose the vertices of the polygon are in an array $p[0], \ldots, p[n-1]$, and the clipped polygon is expected in $q[0], \ldots, q[m-1]$, while the number of vertices of the clipped polygon in variable m. The clipping algorithm, using the notation $\oplus$ for modulo n addition, is:

```
$m = 0$;
**for** $i = 0$ **to** $n - 1$ **do**
    **if** $p[i]$ is inside **then** {
        $q[m{+}{+}] = p[i]$;
        **if** $p[i \oplus 1]$ is outside **then**
            $q[m{+}{+}]$ = Intersection of edge $(p[i], p[i \oplus 1])$;
    } **else if** $p[i \oplus 1]$ is inside **then**
            $q[m{+}{+}]$ = Intersection of edge $(p[i], p[i \oplus 1])$;
    **endif**
**endfor**
```

Running this algorithm for concave polygons that should fall into several pieces due to clipping (figure 5.17) may result in an even number of edges where no edges should have been generated and the parts that are expected to fall apart are still connected by these even number of boundary lines.

For the correct interpretation of the inside of these polygons, the GKS concept must be used, that is, to test whether a point is inside or outside a polygon, a half-line is extended from the point to infinity and the num-

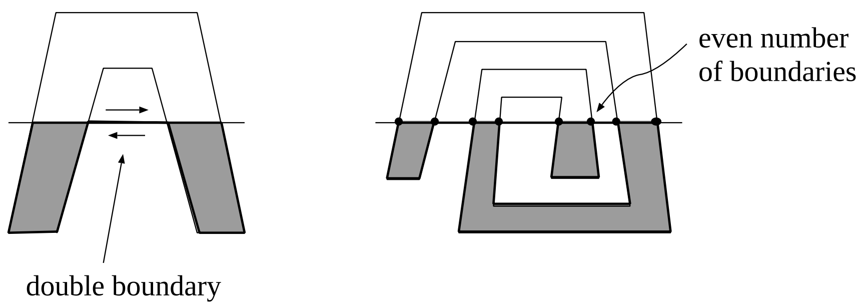

Figure 5.17: Clipping of concave polygons

ber of intersections with polygon boundaries counted. If the line cuts the boundary an odd number of times the point is inside the polygon, if there are even number of intersections the point is outside the polygon. Thus the superficial even number of boundaries connecting the separated parts do not affect the interpretation of inside and outside regions of the polygon.

The idea of Sutherland–Hodgman clipping can be used without modification to clip a polygon against a convex polyhedron defined by planes. A common technique of CAD systems requiring the clipping against an arbitrary convex polyhedron is called **sectioning**, when sections of objects have to be displayed on the screen.

5.6 Viewing pipeline

The discussed phases of transforming the primitives from the world coordinate system to a pixel space are often said to form a so-called **viewing pipeline**. The viewing pipeline is a dataflow model representing the transformations that the primitives have to go through.

Examining figure 5.18, we can see that these viewing pipelines are somehow different from the pipelines discussed by other computer graphics textbooks [FvDFH90], [NS79], because here the screen coordinates contain information about the viewport, in contrast to many authors who define the screen coordinate system as a normalized system independent of the final physical coordinates. For parallel projection, it makes no difference which of the two interpretations is chosen, because the transformations are eventu-

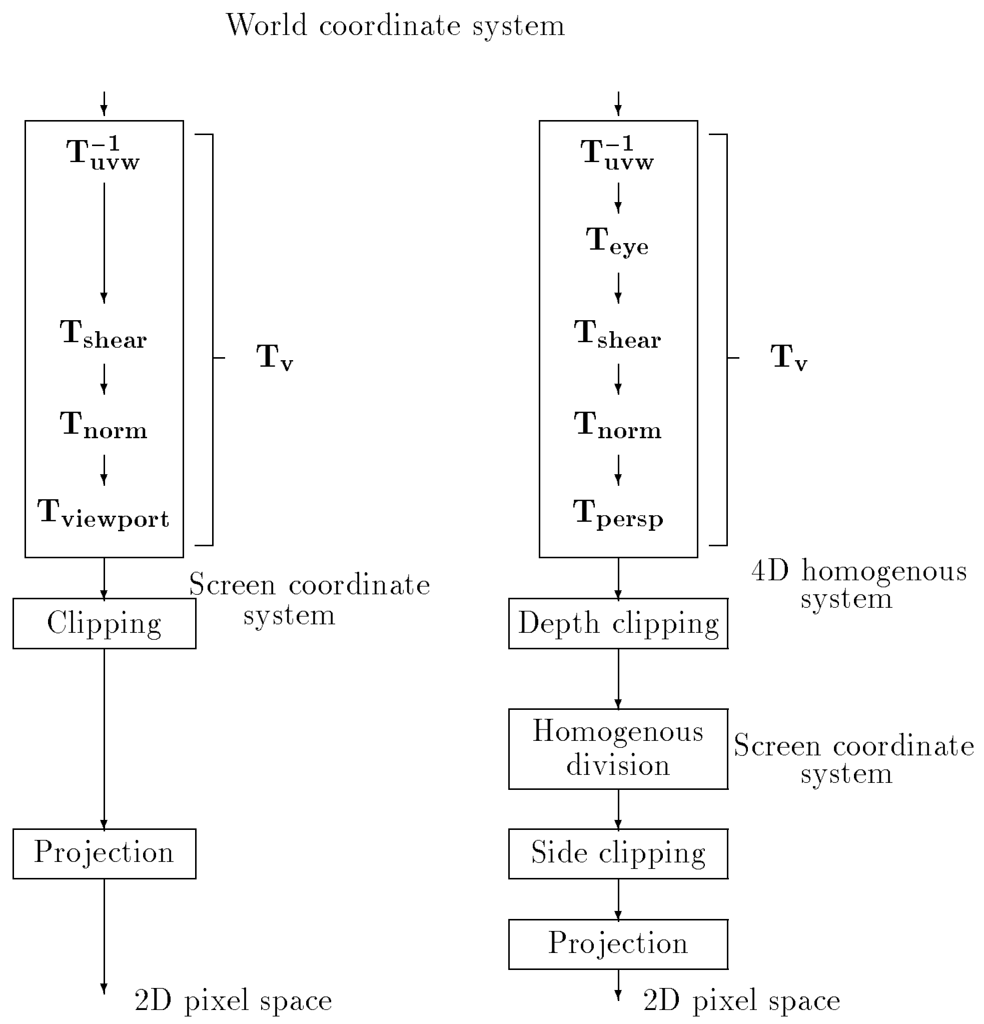
World coordinate system
$\mathbf{T}_{\mathbf{uvw}}^{-1}$
$\mathbf{T}_{\mathbf{shear}}$
$\mathbf{T}_{\mathbf{norm}}$
$\mathbf{T}_{\mathbf{viewport}}$
$\mathbf{T}_{\mathbf{v}}$
Screen coordinate system
Clipping
Projection
2D pixel space
$\mathbf{T}_{\mathbf{uvw}}^{-1}$
$\mathbf{T}_{\mathbf{eye}}$
$\mathbf{T}_{\mathbf{shear}}$
$\mathbf{T}_{\mathbf{norm}}$
$\mathbf{T}_{\mathbf{persp}}$
$\mathbf{T}_{\mathbf{v}}$
4D homogenous system
Depth clipping
Homogenous division
Screen coordinate system
Side clipping
Projection
2D pixel space

Figure 5.18: Viewing pipeline for parallel and perspective projection

ally concatenated to the same final matrix. For perspective transformation, however, the method discussed here is more efficient, although more difficult to understand, because it does not need an extra transformation to the viewport after the homogeneous division, unlike the approach based on the concept of normalized screen coordinates.

Concerning the coordinate system where the clipping has to be done, figure 5.18 represents only one of many possible alternatives. Nevertheless, this alternative is optimal in terms of the total number of multiplications and divisions required in the clipping and the transformation phases.

At the end of the viewing pipeline the clipped primitives are available in the screen coordinate system which is the primary place of visibility calculations, since here, as has been emphasized, the decision about whether one point hides another requires just three comparisons. Projection is also trivial in the screen coordinate system, since the X and Y coordinates are in fact the projected values.

The angles needed by shading are not invariant to the viewing transformation from the shearing transformation phase. Thus, color computation by the evaluation of the shading equation must be done before this phase. Most commonly, the shading equation is evaluated in the world coordinate system.

5.7 Complexity of transformation and clipping

Let the number of vertices, edges and faces of a polygon mesh model be v, e and f respectively. In order to transform a polygon mesh model from its local coordinate system to the screen coordinate system for parallel projection or to the homogeneous coordinate system for perspective projection, the vector of the coordinates of each vertex must be multiplied by the composite transformation matrix. Thus the time and the space required by this transformation are obviously proportional to the number of vertices, that is, the transformation is an $O(v)$ algorithm.

Clipping may alter the position and the number of vertices of the representation. For wireframe display, line clipping is accomplished, which can return a line with its original or new endpoints, or it can return no line at all. The time for clipping is $O(e)$, and it returns $2e$ number of points in the worst case. Projection processes these points or the resulting clipped edges independently, thus it is also an $O(e)$ process. For wireframe image synthesis, the complexity of the complete viewing pipeline operation is then $O(v+e)$. According to **Euler's theorem**, for normal solids, it holds that:

$$f + v = e + 2 \tag{5.72}$$

Thus, $e = v + f - 2 > v$ for normal objects, which means that pipeline operation has $O(e)$ complexity.

For shaded image synthesis, the polygonal faces of the objects must be clipped. Let us consider the intersection problem of polygon i having e_i edges and a clipping plane. In the worst case all edges intersect the plane, which can generate e new vertices on the clipping plane. The discussed clipping algorithm connects these new vertices by edges, which results in at most $e_i/2$ new edges. If all the original edges are preserved (partially) by the clipping, then the maximal number of edges of the clipped polygon is $e_i + e_i/2$. Thus an upper bound for the number of edges clipped by the 6 clipping plane is $(3/2)^6 \cdot e_i = \text{const} \cdot e_i$.

Since in the polygon mesh model an edge is adjacent to two faces, an upper bound for the number of points which must be projected is:

$$2 \cdot \text{const} \cdot (e_1 + \ldots + e_f) = 4 \cdot \text{const} \cdot e. \tag{5.73}$$

Hence the pipeline also requires $O(e)$ time for the polygon clipping mode.

In order to increase the efficiency of the pipeline operation, the method of **bounding boxes** can be used. For objects or group of objects, bounding boxes that completely include these objects are defined in the local or in the world coordinate system, and before transforming the objects their bounding boxes are checked whether or not their image is inside the clipping region. If it is outside, then the complete group of objects is rejected without further calculations.