THREE-DIMENSIONAL GRAPHICS

Three-dimensional graphics is the area of computer graphics that deals with producing two-dimensional representations, or images, of three-dimensional (3-D) synthetic scenes, as seen from a given viewing configuration. The level of sophistication of these images may vary from simple wire-frame representations, in which objects are depicted as a set of segment lines with no data on surfaces and volumes (Fig. 1), to photorealistic rendering, in which illumination effects are computed using the physical laws of light propagation.

All the different approaches are based on the metaphor of a virtual camera positioned in 3-D space and looking at the scene. Hence, independently of the rendering algorithm used, producing an image of the scene always requires the resolution of the following problems (Fig. 2):

1. *Modeling geometric relationships among scene objects,* and in particular efficiently representing the situation in 3-D space of objects and virtual cameras
2. *Culling and clipping,* that is, efficiently determining which objects are visible from the virtual camera
3. *Projecting* visible objects on the film plane of the virtual camera in order to render them

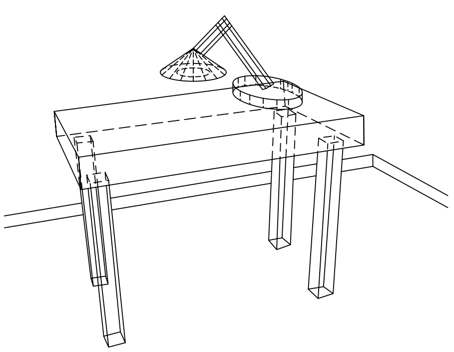

Figure 1. Wire-frame representation of a simple scene.

References 1–4a provide excellent overviews of the field of 3-D graphics. This article provides an introduction to the field by presenting the standard approaches for solving the aforementioned problems.

THREE-DIMENSIONAL SCENE DESCRIPTION

Three-dimensional scenes are typically composed of many objects, each of which may be in turn composed of simpler parts. In order to efficiently model this situation, the collection of objects that comprise the model handled in a three-dimensional graphics application is typically arranged in a hierarchical fashion. This kind of hierarchical structure, known as a *scene graph,* has been introduced by Sutherland (5) and later used in most graphics systems to support information sharing (6).

In the most common case, a transformation hierarchy defines the position, orientation, and scaling of a set of reference frames that create coordinates for the space in which graphical objects are defined. Geometrical objects in a scene graph are thus always represented in their own reference frame, and geometric transformations define the mapping from a coordinate system to another one. This makes it possible to perform numerical computation always using the most appropriate coordinate systems.

During the rendering process, the graph is traversed in order and transformations are composed to implement relative positioning. This kind of hierarchical structure is very handy for many of the operations that are needed for modeling and animating a three-dimensional scene: objects can be easily placed relative to one another, and the animation of articulated objects can be done in a natural way. Figure 3 shows a possible structuring of the scene presented in Fig. 1.

The scene graph provides additional features for simplifying transformation composition, and in particular can be used to factor out commonality. Since graphical attributes are usually propagated from parent to child, setting attributes high in the scene hierarchy effectively sets the attributes for the entire subgraph. As an example, setting to red the color of the root object of the scene graph defines red as the default color of all objects in the scene.

Most modern three-dimensional graphics systems implement some form of scene graph [e.g., OpenInventor (7), VRML (8)]. A few systems, for example, PHIGS and PHIGS+ (9), provide multiple hierarchies, allowing different graphs to specify different attributes.

GEOMETRIC TRANSFORMATIONS

Geometric transformations describe the mathematical relationship between coordinates in two different reference frames. In order to support transformation composition efficiently, three-dimensional graphics systems impose restrictions on the type of transformations used in a scene graph, typically limiting them to be linear ones.

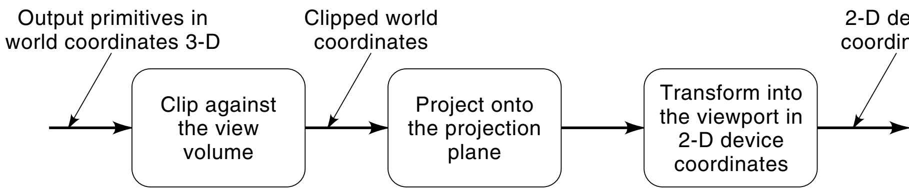

Figure 2. Three-dimensional viewing pipeline.

Linear transformations have the remarkable property that, since line segments are always mapped to line segments, it is not necessary to compute the transformation of all points of an object but only that of a few characteristic points. This obviously reduces the computational burden with respect to supporting arbitrary transformations. For example, only the vertices of a polygonal object need to be transformed to obtain the image of the original object. Furthermore, each elementary linear transformation can be represented mathematically using linear equations for each of the coordinates of a point, which remains true for transformation sequences. It is thus possible to perform complex transformations with the same cost associated with performing elementary ones.

Using 3-D Cartesian coordinates does not permit the representation of all types of transformations in matrix form (e.g., 3-D translations cannot be represented as 3×3 matrices), which is desirable to support transformation composition efficiently. For this reason, 3-D graphics systems usually represent geometric entities using *homogeneous coordinates*.

Homogeneous Coordinates

Ferdinand Möbius introduced the concept of homogeneous coordinates in the 19th century as a method for mathematically representing the position P of the center of gravity of three masses lying onto a plane (10). Once the three masses are arbitrarily placed, the *weights* of the masses define the placement of P, and a variation in one of the weights is reflected in a variation of P. Thus we have a coordinate system in which three coordinates define a point on the plane inside the triangle identified by the three masses. Forgetting the physics and using negative masses, we can represent any point on the plane even if it is outside the triangle. An interesting property of such a system is given by the fact that scaling the three weights by the same scale factor does not change the position of the center of gravity: this implies that the coordinates of a point are not unique.

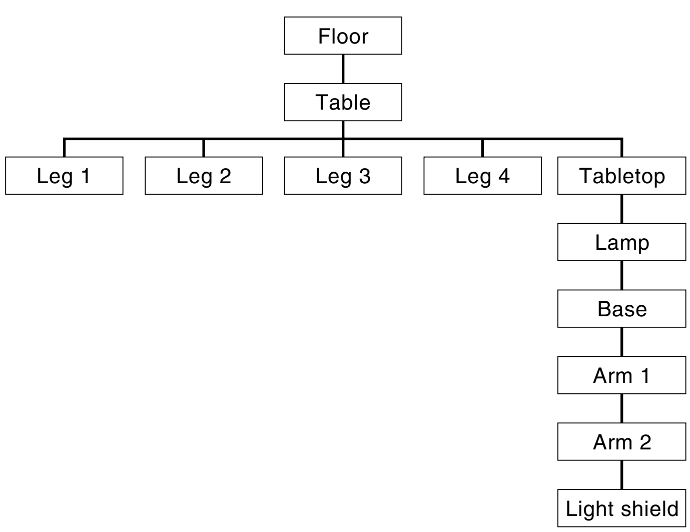

Figure 3. Scene graph of the scene in Fig. 1.

A slightly different formulation of this concept, due to Plücker, defines the coordinates of the point P on the Cartesian plane in terms of the distances from the edges of a fixed triangle (11). A particular case consists in placing one of the edges of the triangle at infinity; under this assumption the relation between the Cartesian coordinates of a point $P = (x, y)$ and its homogeneous coordinates $(X, Y, W)^T$ is

$$x = X/W, \qquad y = Y/W \qquad W \neq 0$$

The same notation extended to the Cartesian space will use the distances from the four sides of an arbitrary tetrahedron. The relation between the Cartesian coordinates of a point $P = (x, y, z)$ and its homogeneous coordinates $(X, Y, Z, W)^T$ is

$$x = X/W, \qquad y = Y/W, \qquad z = Z/W, \qquad W \neq 0$$

Notice that when W is 1 the other coordinates coincide with the Cartesian ones.

Since the curve and surface equations, defined using this coordinate definition, are homogeneous (all the terms have the same degree), this coordinate system is called a *homogeneous coordinate system.*

Matrix Representation of Geometric Entities. Using homogeneous coordinates any three-dimensional linear transformation can be represented by a 4×4 matrix. Points are represented in homogeneous coordinates as column vectors by setting their w coordinate to 1, while vectors have their w coordinate set to 0. Geometric transformations are then performed simply by matrix multiplication.

If $\mathbf{T}$ is the matrix representation of a transformation mapping coordinates in a reference frame F_a to coordinates in a reference frame F_b, the coordinates of a point $P' = (p_x', p_y', p_z', 1)^T$ relative to F_b are obtained from the coordinates $P = (p_x, p_y, p_z, 1)^T$ relative to F_a in two steps:

1. Let $(x, y, z, w)^T = \mathbf{T} \cdot (p_x', p_x', p_x', 1)$.
2. Then $P' = (x/w, y/w, z/w, 1)$.

Vectors are instead transformed by simply performing matrix multiplication followed by setting the w coordinate to 0.

Since any transformation is represented by a 4×4 matrix, matrix composition can be used to minimize the number of algebraic operations needed to perform multiple geometrical transformations. The composed matrix is computed only once and then used on any object of the scene that should be transformed. Homogeneous coordinates therefore unify the treatment of common graphical transformations and operations. The value of this fact has been recognized early in the development of computer graphics (12), and homogeneous coordinates have become the standard coordinate system for programming three-dimensional graphics systems.

Normal Vectors and the Dual Space. In many 3-D graphics applications, it is important to introduce the idea of a *normal vector.* For example, polygonal models usually have normal vectors associated with vertices, which are used for performing shading computations. It is easy to demonstrate that if the normal to a plane passing through three points is transformed as a vector, its image does not remain orthogonal to the plane passing through the images of the three points (13).

In order to obtain the correct behavior, normal vectors must be modeled as algebraic entities called *dual vectors,* which intuitively represent oriented planes. The consequences of this fact can be summarized as follows (13):

1. Dual vectors are represented as row vectors.
2. If $\mathbf{T}$ is the matrix representation of a geometric transformation, then dual vectors are transformed by multiplying them by the inverse transpose of $\mathbf{T}$, followed by setting the last component to 0.

Matrix Representation of Primitive Linear Transformations. In a right-handed system the translation matrix is

$$\mathbf{T}(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The scaling matrix is

$$\mathbf{S}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Notice that reflections about one of the Cartesian axes or about the origin of the coordinate system are special cases of scaling, where one or all the scale factors are set to -1.

The rotation matrices around the Cartesian axes are

$$\mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The general form of a shear matrix is an identity matrix plus six shear factors:

$$\mathbf{H} = \begin{bmatrix} 1 & h_{xy} & h_{xz} & 0 \\ h_{yx} & 1 & h_{yz} & 0 \\ h_{zx} & h_{zy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Manipulation of Orientation and Rotation

In a synthetic scene, cameras and visible objects are often manipulated as if they were rigid objects, and their situation in space is described as a rotation plus a translation from an initial orientation and position. We have seen that homogeneous coordinates are a general way to describe 3-D positions and transformations. However, the collection of all possible orientations of a 3-D rigid body forms an orientation space that is quite different from the Euclidean space of positions, and a good parametrization is needed in order to perform meaningful operations easily. Representing orientations and rotations as matrices is sufficient for applications that require only transformation composition but does not support transformation interpolation, a required feature for applications such as *key-framing.* In particular, interpolation of rotation matrices does not produce orientation interpolation but introduces unwanted shearing effects as the matrix deviates from being orthogonal.

It can be demonstrated that four parameters are needed to create coordinates for the orientation space without singularities (14). Common three-value systems such as Euler angles (i.e., sequences of rotations about the Cartesian axes) are therefore not appropriate solutions. *Unit quaternions,* invented by Hamilton in 1843 (14) and introduced to the computer graphics community by Schoemake (15), have proven to be the most natural parametrization for orientation and rotation.

Quaternion Arithmetic for 3-D Graphics. A quaternion $\mathbf{q} = [w, \mathbf{v}]$ consists of a scalar part, the real number w, and an imaginary part, the 3-D vector $\mathbf{v}$. It can be interpreted as a point in four-space, with coordinates $[x, y, w, z]$, equivalent to homogeneous coordinates for a point in projective three-space. Quaternion arithmetic is defined as the usual 4-D vector arithmetic, augmented with a multiplication operation defined as follows:

$$\begin{aligned} \mathbf{q}_1\mathbf{q}_2 &= [s_1, \mathbf{v}_1][s_1, \mathbf{v}_2] \\ &= [(s_1, s_2 - \mathbf{v}_1 \cdot \mathbf{v}_2), (s_1\mathbf{v}_2 + s_2\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2)] \end{aligned}$$

A rotation of angle θ and an axis aligned with a unit vector $\mathbf{a}$ is represented in quaternion form by the unit quaternion $\mathbf{q} = [\cos(\theta/2), \sin(\theta/2), \mathbf{a}]$.

With this convention, composition of rotations is obtained by quaternion multiplication, and linear interpolation of orientations is obtained by linearly interpolating quaternion components. The formula for spherical linear interpolation from $\mathbf{q}_1$ to $\mathbf{q}_2$, with parameter u moving from 0 to 1, is the following:

$$\text{slerp}(\mathbf{q}_1, \mathbf{q}_2, u) = \frac{\sin[(1-u)\theta]}{\sin\theta}\mathbf{q}_1 + \frac{\sin(u\theta)}{\sin\theta}\mathbf{q}_2$$

where

$$\mathbf{q}_1 \cdot \mathbf{q}_2 = \cos\theta$$

Quaternions are easily converted to and from transformation matrices. The rotation matrix equivalent to a quaternion $\mathbf{q} = [w, x, y, z]$ is

$$\mathbf{R}(\mathbf{q}) = \begin{pmatrix} 1-2(y^2+z^2) & 2(xy+wz) & 2(xz-wy) & 0 \\ 2(xy-wz) & 1-2(x^2+z^2) & 2(yz+wx) & 0 \\ 2(xz+wy) & 2(yz-wx) & 1-2(x^2+y^2) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Shoemake (16) presents a simple algorithm for performing the inverse operation of transforming a rotation matrix into a quaternion.

Projections

A *projection* is a geometrical transformation from a domain of dimension n to a co-domain of dimension $n - 1$ (or less). When producing images of 3-D scenes, we are interested in projections from three to two dimensions.

The process of projecting a 3-D object on a planar surface is performed casting straight rays from a single point, possibly at infinity, through each of the points forming the object, and computing the intersections of the rays with the projection plane. Projecting all the points forming a segment is equivalent to projecting its end points and then connecting them on the projection plane. The projection process can be then reduced to project only the vertices of the objects forming the scene. This particular class of projections is the class of *planar geometric projections.*

There are two major categories of planar geometric projections: *parallel* and *perspective.* When the distance between the projection plane and the center of projection is finite the projection is perspective; otherwise it is parallel (Fig. 4).

A perspective projection is typically used to simulate a realistic view of the scene, while a parallel one is more suited for technical purposes.

To give an example, assuming that:

1. The projection plane is normal to the z axis at distance z_p
2. The normalized distance between the center of projection and the intersection between the projection plane and the z axis is $Q(d_x, d_y, d_z)$

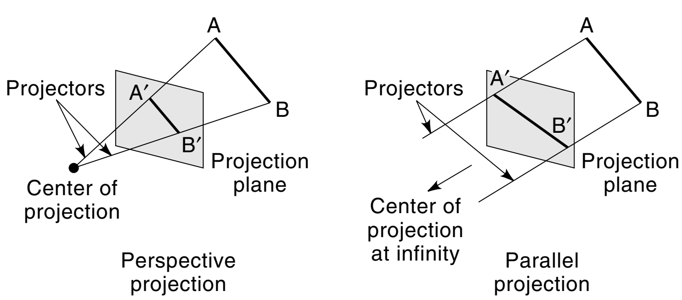

Figure 4. Perspective and parallel projections.

we can generically represent this class of projections by a matrix of the form

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & -\dfrac{d_x}{d_y} & z_p\dfrac{d_x}{d_z} \\ 0 & 1 & -\dfrac{d_y}{d_y} & z_p\dfrac{d_y}{d_z} \\ 0 & 0 & -\dfrac{z_p}{Qd_z} & \dfrac{z_p^2}{Qd_z}+z_p \\ 0 & 0 & -\dfrac{1}{Qd_z} & \dfrac{z_p^2}{Qd_z}+1 \end{bmatrix}$$

THREE-DIMENSIONAL VIEWING PROCESS

Specifying a View in 3-D Space

As summarized in Fig. 2, to define a 3-D view, we do not only need to define a *projection* but also to bound a *view volume,* that is, the region of the space including all and only the visible objects. The projection and view volume together give us all the information necessary to clip and project.

While this process could be totally described using the mathematics seen before, it is much more natural to describe the entire transformation process using the so-called *camera metaphor.* Setting the parameter of a synthetic view is analogous to taking a photograph with a camera. We can make a schematic of the process of taking a picture in the following steps:

1. Place the camera and point it to the scene.
2. Arrange the objects in the scene.
3. Choose the lens or adjust the zoom.
4. Decide the size of the final picture.

Generating a view of a synthetic scene on a computer, these four actions correspond to define, respectively, the following four transformations:

1. Viewing transformation
2. Modeling transformation
3. Projection transformation
4. Viewport transformation

The modeling transformation is, typically, a way to define objects in the scene in a convenient coordinate system and then transform them in a single, general, coordinate system called the *world coordinate system.* The meaning of the other three is explained in detail in the following.

Viewing Transformation. The projection plane (*view plane*) is defined by a point, the *view reference point* (VRP) and a normal to the plane, the *view plane normal* (VPN). In the real world we are accustomed to place the projection plane always beyond the projected objects with respect to the observer (e.g., a cinema screen). In a synthetic scene, instead, the plane can be in any relation to the objects composing the scene: in front of, behind, or even cutting through them.

A rectangular window on the plane results from the intersection between the projection plane and the view volume. Any object projected on the plane outside the window's bound-

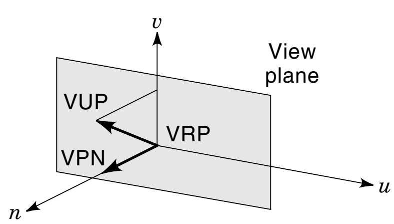

Figure 5. Parameters defining the view plane.

aries is not visible, that is, it is not part of the final 2-D image. To define the window we place a coordinate system on the plane; we call it the *viewing reference coordinate* (VRC) system. One of the axes of the VRC system, the *n* axis, is defined by VPN, another one, the *v* axis, by the projection of the *view up vector* (VUP) onto the plane, and the third one, the *u* axis, is chosen such that *u*, *v*, and *n* form a right-handed coordinate system (Fig. 5).

It is thus possible to define the window in terms of its $u_{\min}$, $u_{\max}$, $v_{\min}$, and $v_{\max}$ coordinates (Fig. 6).

The window does not need to be symmetrical about the VRP. In other words, the *center of the window* (CW) can be distinct from the VRP.

Projection Transformation. The *center of projection* or the *direction of projection* (DOP) is defined by the *projection reference point* (PRP) plus the chosen projection type: parallel or perspective. In the case of perspective projection the center of projection is PRP; in the case of parallel projections the direction of projection is from PRP to CW (Fig. 7).

In the perspective projection the view volume is a semi-infinite pyramid, called the *view frustum,* while in parallel projection it is an infinite parallelepiped with sides parallel to the direction of projection.

It is useful to set up a method limiting the view volume to be finite. This avoids objects being too close to the PRP to occlude the view, and objects too far away to be rendered, since they would be too small to influence the final image. Two more attributes of the view make this possible: the *front (hither) clipping plane* and the *back (yon) clipping plane.* They are both parallel to the view plane and specified by, respectively, the *front distance* (*F*) and the *back distance* (*B*). When the front clipping plane is further away from the PRP than the back clipping plane, the view volume is empty.

We can compare the synthetic viewing process to the real human single-eyed perspective one. The PRP represents the position of the human eye, the view volume is an approximation of the conelike shaped region viewed by the eye, the view

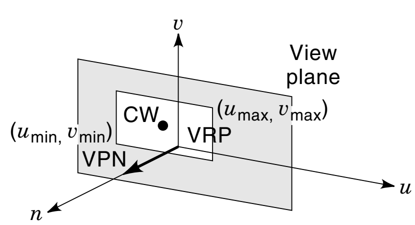

Figure 6. Parameters defining the window on the view plane.

plane is placed at the focal distance from the eye, and the VUP points from the top of the head up.

Viewport Transformation. The content of the view volume is transformed in *normalized projection coordinate* (NPC) into the so-called *canonical view volume* and then projected on the display viewport by eliminating the *z* information from all the points. The normalization matrix ($\mathbf{N}_{\text{par}}$) for parallel projection is a composition of

- Translation of VRP to the origin, $\mathbf{T}(-\text{VRP})$
- Rotation of VRC to align *n* (VUP) with *z*, *u* with *x*, and *v* with *y*, $\mathbf{R}$
- Shearing to make the direction of projection parallel to the *z* axis, $\mathbf{H}_{\text{par}}$
- Translation and scaling to the parallel canonical volume, a parallelepiped, defined by the equations $x = -1$, $x = 1$, $y = -1$, $y = 1$, $z = -1$, $z = 0$, $\mathbf{T}_{\text{par}}$, and $\mathbf{S}_{\text{par}}$.

In formula:

$$\mathbf{N}_{\text{par}} = \mathbf{S}_{\text{par}} \cdot \mathbf{T}_{\text{par}} \cdot \mathbf{H}_{\text{par}} \cdot \mathbf{R} \cdot \mathbf{T}(-\text{VRP})$$

For a perspective projection the normalization matrix (N_{per}) is a composition of

- Translation of VRP to the origin, $\mathbf{T}(-\text{VRP})$
- Rotation of VRC to align *n* (VUP) with *z*, *u* with *x*, and *v* with *y*, $\mathbf{R}$
- Translation of PRP to the origin, $\mathbf{T}(-\text{PRP})$
- Shearing to make the center line of the view volume being the *z* axis, $\mathbf{H}_{\text{per}}$
- Scaling to the perspective canonical volume, a truncated pyramid, defined by the equations $x = z$, $x = -z$, $y = z$, $y = -z$, $z = -z_{\min}$, $z = -1$, $\mathbf{S}_{\text{per}}$

In formula:

$$\mathbf{N}_{\text{per}} = \mathbf{S}_{\text{per}} \cdot \mathbf{H}_{\text{per}} \cdot \mathbf{T}(-\text{PRP}) \cdot \mathbf{R} \cdot \mathbf{T}(-\text{VRP})$$

If we, then, premultiply $\mathbf{N}_{\text{per}}$ by the transformation matrix from the perspective to the parallel canonical view volume:

$$\mathbf{M}_{\text{per}\to\text{par}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{1}{1+z_{\min}} & \dfrac{-z_{\min}}{1+z_{\min}} \\ 0 & 0 & -1 & 0 \end{bmatrix}, \qquad z_{\min} \neq -1$$

we obtain

$$\mathbf{N}'_{\text{per}} = \mathbf{M}_{\text{per}\to\text{par}} \cdot \mathbf{N}_{\text{per}} = \mathbf{S}_{\text{per}} \cdot \mathbf{H}_{\text{per}} \cdot \mathbf{T}(-\text{PRP}) \cdot \mathbf{R} \cdot \mathbf{T}(-\text{VRP})$$

that is, the matrix transforming the object in the scene to the canonical parallepided defined before.

Using $\mathbf{N}'_{\text{per}}$ and $\mathbf{N}_{\text{par}}$ we are thus able to perform the clipping operation against the same volume using a single procedure.

Culling and Clipping

The clipping operation consists of determining which parts of an object are visible from the camera and need to be projected

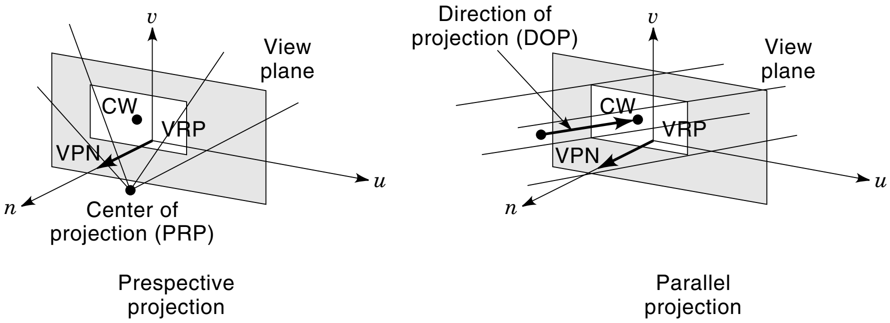

Figure 7. View volumes for perspective and parallel projections.

on the screen for rendering. This operation is performed on each graphic and is composed of two different steps. First, during culling, objects completely outside of the view volume are eliminated. Then, partially visible objects are cut against the view volume to obtain only totally visible primitives.

Culling of Points. At the end of the projection stage all the visible points describing the scene are inside the volume defined by the equations

$$x = -1, \qquad x = 1, \qquad y = -1, \qquad y = 1, \qquad z = -1 \qquad z = 0$$

The points satisfying the inequalities

$$-1 \le x \le 1, \qquad -1 \le y \le 1, \qquad -1 \le z \le 0$$

are visible; all the others have to be clipped out.

The same inequalities expressed in homogeneous coordinates are:

$$-1 \le X/W \le 1, \qquad -1 \le Y/W \le 1, \qquad -1 \le Z/W \le 0$$

corresponding to the plane equations

$$X = -W, \quad X = W, \quad Y = -W, \quad Y = W, \quad Z = -W, \quad Z = 0$$

Clipping of Line Segments. The most popular line-segment clipping algorithm, and perhaps the most used, is the Cohen–Sutherland algorithm. Since it is a straightforward extension of the two-dimensional clipping algorithm, we illustrate this one first for sake of simplicity of explanation.

When clipping a line against a 2-D rectangle, the plane is tessellated in nine regions (Fig. 8); each one identified by a four-bit code, in which each bit is associated with an edge of the rectangle. Each bit is set to 1 or 0 when the conditions listed in Table 1 are, respectively, true or false.

The first step of the algorithm assigns a code to both endpoints of the segment, according to the position of the points with respect to the clipping rectangle. If both endpoints have a code of 0000, then the segment is totally visible. If the logic AND of the two bit codes gives a result different from 0, then both the endpoints lie in a half-plane not containing the visible rectangle and thus the segment is totally invisible. Otherwise the next step computes the intersection of the segment with one edge of the rectangle and the process iterates on the segment connecting the found intersection and the remaining endpoint.

In three dimensions a code of six bits is used. When the segments are clipped against the canonical view volume the conditions associated with the bits are

$$X \ge -W,\ X \le W,\ Y \ge -W,\ Y \le W,\ Z \ge -W,\ Z \le 0 \text{ for } W > 0$$
$$X \le -W,\ X \ge W,\ Y \le -W,\ Y \ge W,\ Z \le -W,\ Z \ge 0 \text{ for } W < 0$$

When clipping ordinary lines and points, only the first set of inequalities applies. For further discussion refer to Blinn and Newell (17).

The trivial acceptance and rejection tests are the same as in 2-D. There is a change in the line subdivision step, since the intersections are computed between lines and planes instead of lines and lines.

Clipping of Polygons. Clipping of polygons differs from clipping of a collection of segment lines when they are considered as solid areas. In this case it is necessary that closed polygons remain closed.

The standard algorithm for clipping polygons is due to Sutherland and Hodgman (18). Their algorithm uses a "divide and conquer approach," decomposing the problem as a sequence of simpler clippings of the polygon against each plane delimiting the canonical view volume.

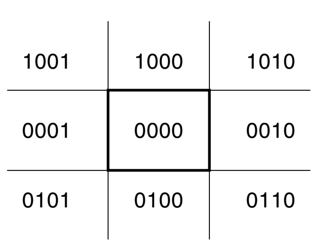

Figure 8. Tessellation of the plane in the 2-D Cohen–Sutherland algorithm.

Table 1. Bit Codes for the Classification of Points in the Two-Dimensional Cohen–Sutherland Algorithm

Bit 1	Point in the half-plane over the upper edge	$y > y_{max}$
Bit 2	Point in the half-plane under the lower edge	$y < y_{min}$
Bit 3	Point in the half-plane to the right of the right edge	$x > x_{max}$
Bit 4	Point in the half-plane to the left of the left edge	$x < x_{min}$

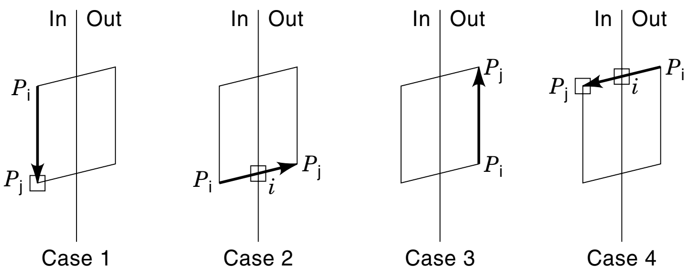

Figure 9. Different possibilities for edge-clipping plane comparison.

The polygon is originally defined by the list of its vertices $P = P_1, \ldots, P_n$, which implies a list of edges $\overline{P_1P_2}, \overline{P_2P_3}, \ldots, P_{n-1}P_n, P_nP_1$. Let H be the half-space, defined by the current clipping plane h, containing the view volume. The algorithm produces a list of polygon vertices Q that are all inside H by traversing each edge $\overline{P_iP_j}$ in sequence and producing at each edge-clipping plane comparison zero, one, or two vertices (Fig. 9):

1. If $\overline{P_iP_j}$ is entirely inside H, P_j is inserted into Q.
2. If P_i is inside H and P_j is outside, the intersection of $\overline{P_iP_j}$ with h is inserted into Q.
3. If $\overline{P_iP_j}$ is entirely outside H, nothing is inserted into Q.
4. If P_i is outside H and P_j is inside, the intersection of $\overline{P_iP_j}$ with h and P_j are inserted into Q.

The output polygon Q is then used to feed the next clipping step. The algorithm terminates when all planes bounding the canonical view volume have been considered.

Sutherland and Hodgman (18) presented a version of this algorithm that does not require storing intermediate results and is therefore better suited to hardware implementation.

BIBLIOGRAPHY

1. J. D. Foley et al., *Computer Graphics: Principles and Practice,* 2nd ed., Reading, MA: Addison-Wesley, 1990.
2. D. F. Rogers, *Procedural Elements for Computer Graphics,* New York: McGraw-Hill, 1985.
3. D. F. Rogers and J. A. Adams, *Mathematical Elements for Computer Graphics,* 2nd ed., New York: McGraw-Hill, 1990.
4. A. Watt, *Fundamentals of Three-Dimensional Computer Graphics,* Reading, MA: Addison-Wesley, 1990.

4a. W. F. Taylor, *The Geometry of Computer Graphics,* Pacific Grove, CA: Wadsworth & Brooks, 1992.

5. I. Sutherland, Sketchpad: A man-machine graphical communication system, in *Proc. Spring Joint Comput. Conf.,* 1963, pp. 329–346.
6. D. B. Conner and A. van Dam, Sharing between graphical objects using delegation, in *Proc. 3rd Eurographics Workshop Object-Oriented Graphics,* 1992, pp. 63–82.
7. The OpenInventor Architecture Group, *OpenInventor C++ Reference Manual: The Official Reference Document for Open Systems,* Reading, MA: Addison-Wesley, 1994.
8. R. Carey and G. Bell, *The VRML 2.0 Annotated Reference Manual,* Reading, MA: Addison-Wesley, 1997.
9. T. Gaskins, *PHIGS Programming Manual,* Sebartopol, CA: O'Reilly and Associates, 1992.
10. F. Möbius, *Gesammelte Werke,* Vol. 1: *Die Barycentrische Calcul,* Wiesbaden, Germany: Dr. M. Saendig oHG, 1967, pp. 36–49.
11. J. Plücker, Ueber ein neues Coordinatensystem, *J. Reine Angew. Math.,* **5**: 1–36, 1830.
12. L. G. Roberts, *Homogeneous Matrix Representation and Manipulation of N-Dimensional Constructs,* Technical Report MS-1405, Lincoln Laboratory, MIT, May, 1965.
13. T. DeRose, A coordinate-free approach to geometric programming, in W. Strasser and H. Seidel (eds.) *Theory and Practice of Geometric Modeling,* New York: Springer, 1989, pp. 291–306.
14. W. R. Hamilton, On quaternions; or on a new system of imaginaries in algebra, *Philos. Mag.,* **XXV**: 10–13, 1844.
15. K. Schoemake, Animating rotation with quaternion curves, *Comput. Graphics,* **19** (3): 245–254, 1985.
16. K. Schoemake, Polar decomposition for rotation extraction, Notes for Course No. C2, *Math for SIGGRAPH,* SIGGRAPH Tutorial Notes, 1991.
17. J. F. Blinn and M. E. Newell, A Homogeneous Formulation for Lines in 3-Space, *Proc. SIGGRAPH,* 237–241, 1977.
18. I. Sutherland and G. W. Hodgman, Reentrant polygon clipping, *Commun. ACM,* **17**: 32–42, 1974.

ENRICO GOBBETTI
RICCARDO SCATENI
Center for Advanced Studies, Research, and Development in Sardinia, CRS4

THREE-DIMENSIONAL SCANNERS. See RANGE IMAGES.