Three-Dimensional Graphics

Enrico Gobbetti and Riccardo Scateni

CRS4, Center for Advanced Studies, Research, and Development in Sardinia

Via Nazario Sauro 10

09123 Cagliari, Italy

E-mail: {Enrico.Gobbetti|Riccardo.Scateni}@crs4.it

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

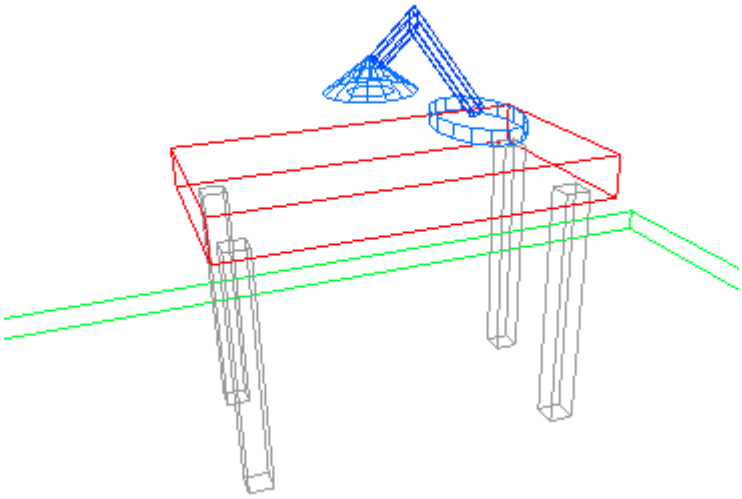

Figure 1. Wire-frame representation of a simple scene.

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

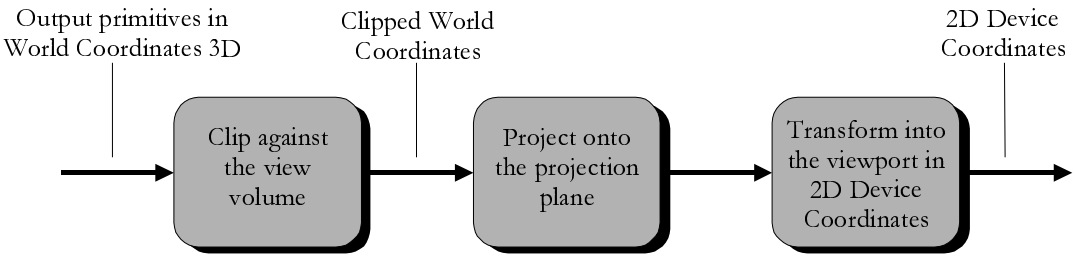

Figure 2. Three-dimensional viewing pipeline.

1. *Modeling geometric relationships among scene objects*, and in particular efficiently representing the situation in 3D space of objects and virtual cameras;
2. *Culling and clipping*, i.e. 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.

References (1 - 4) provide excellent overviews of the field of three-dimensional graphics. This chapter 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 coordinatize 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 always perform numerical computation 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 an animating a three-dimensional scene: objects can be easily placed one relative to another, and the animation of articulated s can be done in a natural way. Figure 3 shows a possible structuring of the scene presented in figure 1.

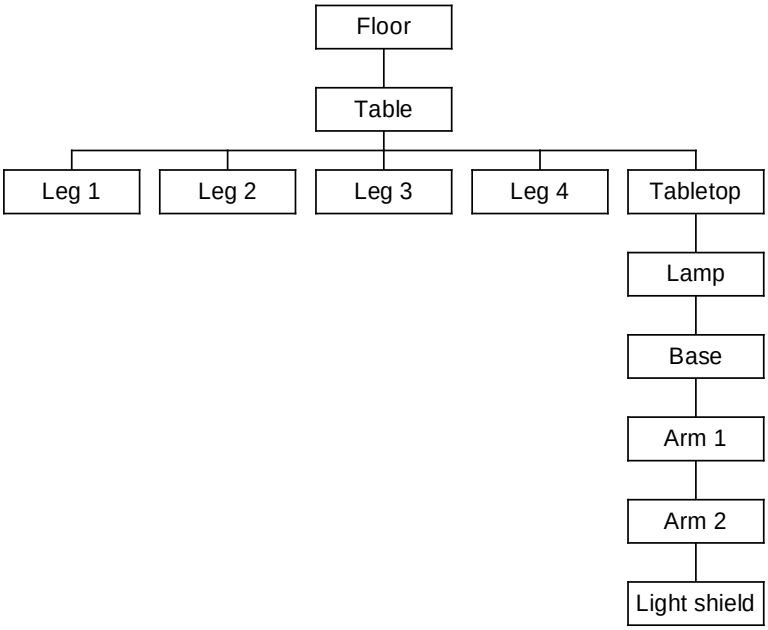

Figure 3. Scene graph of the scene in figure 1.

The scene graph provides additional features to simple 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 system, e.g. 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 efficiently support transformation composition, three-dimensional graphics systems impose restrictions on the type of transformations used in a scene graph, typically limiting them to be linear ones.

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 of a few characteristic ones which obviously reduces the computational burden with respect to supporting arbitrary transformations. For instance, 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 to performing elementary ones.

Using 3D Cartesian coordinates does not permit the representation of all types of transformations in matrix form (e.g. 3D translations cannot be represented as 3x3 matrices),

which is desirable to efficiently support transformation composition. For this reason, three-dimensional graphics systems usually represent geometric entities using *homogeneous coordinates*.

Homogeneous Coordinates

Ferdinand Möbius introduced the concept of homogeneous coordinates in the XIX 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 reflects on a variation of P. We have, thus, a coordinate system where 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.

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 $\mathrm{P} = (x, y)$ and its homogeneous coordinates $(X, Y, W)^{\mathrm{T}}$ will be:

$$x = X/W; \quad y = Y/W; \quad W \neq 0.$$

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 $\mathrm{P} = (x, y, z)$ and its homogeneous coordinates $(X, Y, Z, W)^{\mathrm{T}}$ will be:

$$x = X/W; \quad y = Y/W; \quad z = Z/W; \quad W \neq 0.$$

Notice that, when *w* is 1 the other coordinates coincide with the Cartesian ones.

Since the curves and surfaces equations, defined using this coordinate definition, are homogeneous (all the terms have the same degree), this coordinate system is called *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 **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 $\mathrm{P}' = (\mathrm{p}'_x\ \mathrm{p}'_y\ \mathrm{p}'_z\ 1)^{\mathrm{T}}$ relative to F_b are obtained from the coordinates $\mathrm{P} = (\mathrm{p}_x\ \mathrm{p}_y\ \mathrm{p}_z 1)^{\mathrm{T}}$ relative to F_a in two steps:

1. Let $(x\ y\ z\ w)^{\mathrm{T}} = \mathbf{T} \cdot (\mathrm{p}'_x\ \mathrm{p}'_x\ \mathrm{p}'_x\ 1)$;

2. $P' = (x/w \; y/w \; z/w \; 1)$.

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 three-dimensional graphics applications, it is important to introduce the idea of a *normal vector*. For example, polygonal models usually have normals associated to vertices, which are used for performing shading computations. It is easy to demonstrate that if the normal to a plane passing though 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, normals have to 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 **T** is the matrix representation of a geometric transformation, then dual vectors are transformed by multiplying them by the inverse transpose of **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 where 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 3D positions and transformations. However, the collection of all possible orientations of a three-dimensional rigid body forms an orientation space which is quite different from the Euclidean space of positions, and a good parameterization is needed in

order to easily perform meaningful operations. Representing orientations and rotations as matrices is sufficient for applications which 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 coordinatize 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 parameterization for orientation and rotation.

Quaternion Arithmetic for 3D Graphics

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

$$\mathbf{q_1 q_2} = [s_1, \mathbf{v}_1][s_1, \mathbf{v}_2] = [(s_1 s_2 - \mathbf{v}_1 \bullet \mathbf{v}_2), (s_1 \mathbf{v}_2 + s_2 \mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2)]$$

A rotation of angle θ and axis aligned with a unit vector **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

$$\mathbf{q_1} \bullet \mathbf{q_2} = \cos\theta$$

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 three-dimensional scenes, we are interested in projections from three to two dimensions.

The process of projecting 3D object on a planar surface is performed casting straight rays from a single point, possibly at infinity, trough 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 project its end-points and then connect 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 (figure 4).

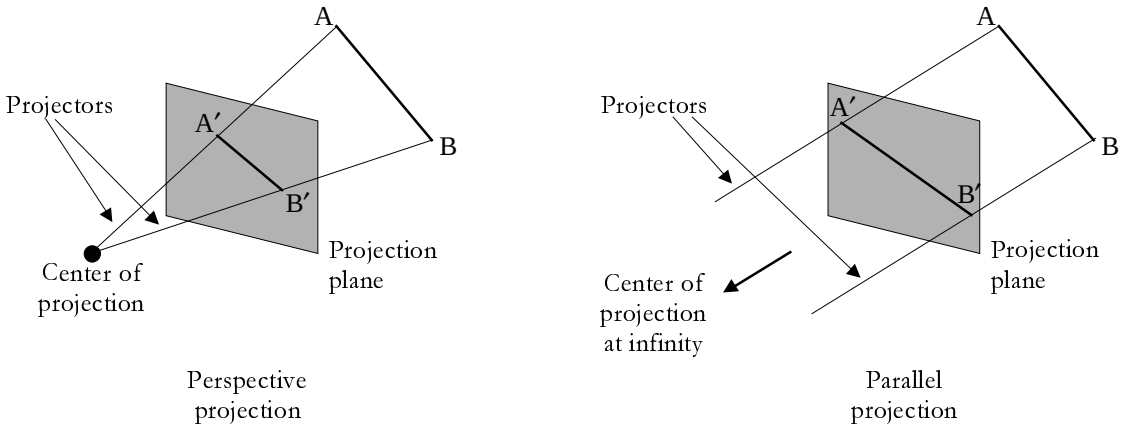

Figure 4. Perspective and parallel projections.

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

Just 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 $Q(d_x, d_y, d_z)$

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

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

Three-Dimensional Viewing Process

Specifying a view in 3D space

As summarized in figure 2, to define a 3D 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 schematize 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, this 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 *world coordinate system*. The meaning of 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 relations with 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 boundaries is not visible, that is it is not part of the final 2D image. To define the window we place a coordinate system on the plane; we call it *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 (figure 5).

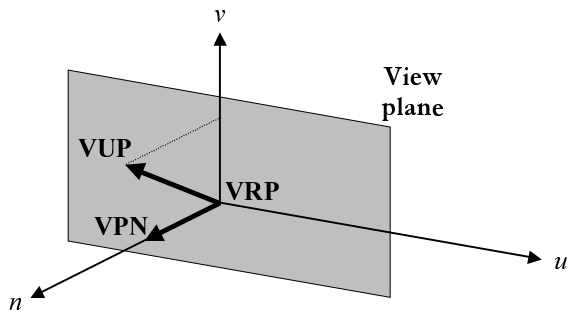

Figure 5. Parameters defining the view plane.

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

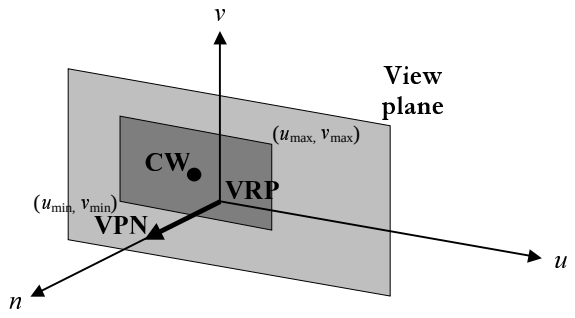

Figure 6. Parameters defining the window on the view plane.

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 case of perspective projection the center of projection is PRP, in case of parallel projections the direction of projection is from PRP to CW (figure 7).

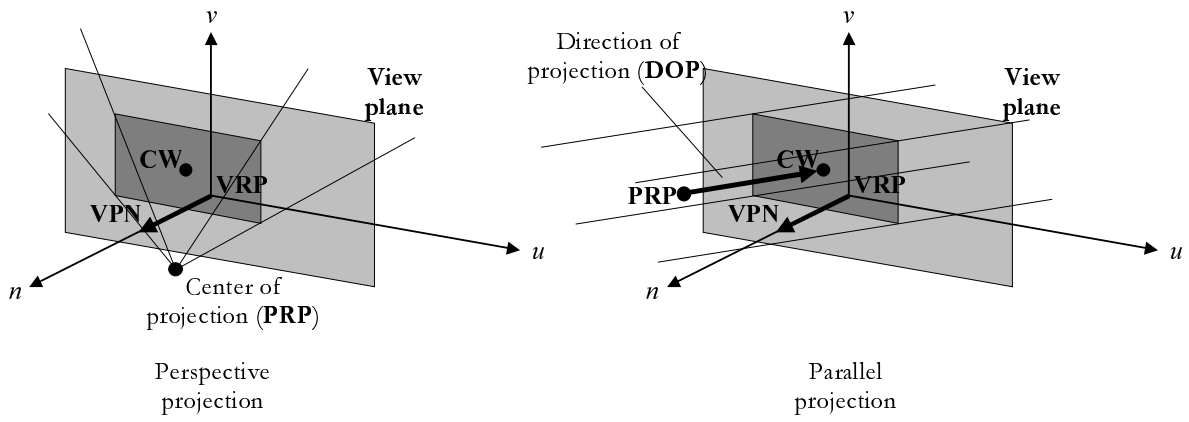

Figure 7. View volumes for perspective and parallel projections.

In the perspective projection the view volume is a semi-infinite pyramid, called *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 that objects too close to the PRP occlude the view, and objects too far away are rendered, since they would anyway 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 cone-like shaped region viewed by the eye, the view plane is placed at 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 just eliminating the *z* information from all the points. The normalization matrix ($\mathbf{N}_{par}$) for parallel projection is a composition of:

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

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, pre-multiply $\mathbf{N}_{\text{per}}$ by the transformation matrix from the perspective to the parallel canonical view volume:

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

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 in determining which parts of an object are visible from the camera and need to be projected on the screen for rendering. This operation is performed on each graphical 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:

$$x = -1; x = 1; y = -1; y = 1; z = -1; z = 0.$$

The points satisfying the inequalities:

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

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

The same inequalities, expressed in homogeneous coordinates are:

$$-1 \le {}^{X}\!/_{W} \le 1, \quad -1 \le {}^{Y}\!/_{W} \le 1, \quad -1 \le {}^{Z}\!/_{W} \le 0$$

corresponding to the plane equations:

$$X = -W; X = W; Y = -W; Y = W; Z = -W; Z = 0.$$

Clipping of Line Segments

The most popular line segments 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 two-dimensional rectangle, the plane is tessellated in nine regions (figure 8); each one identified by a four-bit code, where 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.

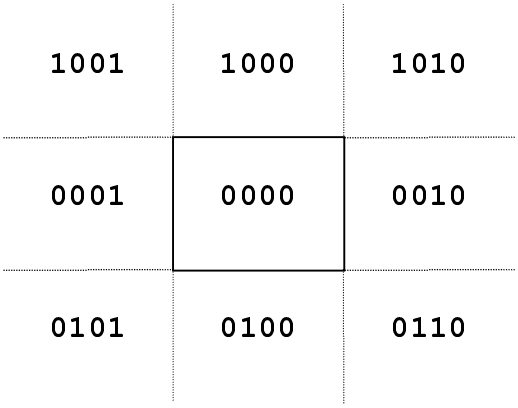

Figure 8. Tessellation of the plane in the two-dimensional Cohen-Sutherland algorithm.

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 $x < x_{\min}$

The first step of the algorithm assigns a code to both end-points of the segment, according to the position of the points with respect to the clipping rectangle. If both the end-points have a code of 0000, than the segment is totally visible. If the logic AND of the two code gives a result different from 0, than both the end-points lay 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 remained end-point.

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

$$W > 0: \quad X \geq -W, X \leq W, Y \geq -W, Y \leq W, Z \geq -W, Z \leq 0,$$
$$W > 0: \quad X \leq -W, X \geq W, Y \leq -W, Y \geq W, Z \leq -W, Z \geq 0.$$

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 than in 2D. 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 in a sequence of simpler clipping of the polygon against each plane delimiting the canonical view volume.

The polygon is originally defined by the list of its vertices $P = P_1, \mathrm{K}\ , P_\mathrm{n}$ which implies a list of edges $\overline{P_1P_2}, \overline{P_2P_3}, \mathrm{K}\ , \overline{P_{\mathrm{n}-1}P_\mathrm{n}}, \overline{P_\mathrm{n}P_1}$. 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

which are all inside H by traversing each edge $\overline{P_i P_j}$ in sequence and producing at each edge-clipping plane comparison zero, one, or two vertices (figure 9):

1. If $\overline{P_i P_j}$ is entirely inside H, P_i is inserted into Q.
2. If P_i is inside H and P_j is outside, the intersection of $\overline{P_i P_j}$ with h is inserted into Q.
3. If $\overline{P_i P_j}$ is entirely outside H, nothing is inserted into Q.
4. If P_i is outside H and P_j is inside, the intersection of $\overline{P_i P_j}$ 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, A. van Dam, S. K. Feiner, J. F. Hughes, *Computer Graphics: Principles and Practice*, 2nd ed., Addison-Wesley, 1990.
2. D.F. Rogers, *Procedural Elements for Computer Graphics*, Mc Graw Hill, 1985.
3. D.F. Rogers, J. Allan Adams, *Mathematical Elements for Computer Graphics*, 2nd ed., Mc Graw Hill, 1990.
4. A. Watt, *Fundamentals of Three-Dimensional Computer Graphics*, Addison Wesley, 1990.

5. I. Sutherland, “Sketchpad: A Man-Machine Graphical Communication System”. In *Proceedings of the Spring Joint Computer Conference*, 329 - 346, 1963.
6. D.B. Conner, A. van Dam, “Sharing Between Graphical Objects Using Delegation”. In *Proceedings of the Third Eurographics Workshop on Object-Oriented Graphics*, 63 - 82, 1992.
7. The OpenInventor Architecture Group, *OpenInventor C++ Reference Manual: The Official Reference Document for Open Systems,* Addison-Wesley, 1994.
8. R. Carey, G. Bell, *The VRML 2.0 Annotated Reference Manual*, Addison-Wesley, 1997.
9. T. Gaskins, *PHIGS Programming Manual*, O'Reilly and Associates, 1992.
10. F. Möbius, *Gesammelte Werke*, vol 1. *Die Barycentrische Calcul*. Dr. M. Saendig oHG, Wiesbaden, Germany, 1967, 36 - 49.
11. J. Plücker, “Ueber ein neues Coordinatensystem”, *Journal für die Reine und Angewandte Mathematik*, vol. 5, 1830, 1 - 36.
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, H. Seidel (ed.) *Theory and Practice of Geometric Modeling*, Springer, 291 - 306, 1989.
14. W.R. Hamilton, “On Quaternions; or on a New System of Imaginaries in Algebra”, *Philosophical Magazine*, XXV, 10 - 13, 1844.
15. K. Schoemake, “Animating Rotation with Quaternion Curves”, *Computer Graphics*, 19(3), 245 - 254, 1985.
16. K. Schoemake, “Polar Decomposition for Rotation Extraction”, Notes for Course #C2, *Math for SIGGRAPH*, SIGGRAPH Tutorial Notes, 1991.
17. J.F Blinn, M.E. Newell, “A Homogeneous Formulation for Lines in 3-Space”, *Proceedings SIGGRAPH*, 237 - 241, 1977.

18. I. Sutherland, G.W. Hodgman, “Reentrant Polygon Clipping”, Communications of the ACM, 17, 32 - 42, 1974.