Chapter 7

Three-Dimensional Viewing

Methods for implementing various aspects of the three-dimensional viewing pipeline are explored in the exercises for this chapter. These problems include implementations of the algorithms for coordinate transformations, projection transformations, and clipping, in addition to programming assignments using the three-dimensional OpenGL viewing functions.

Exercises

7-1. Input parameters for this procedure are the viewing-position coordinates, the view-plane normal vector, and the up direction for the viewing reference frame. The elements of matrix 7-4 are then calculated, using Eqs. 7-1 and 7-5.

7-2. This exercise can combine the viewing transformation in the preceding exercise with a parallel projection. Input values are specified for the elements of the projection vector $\mathbf{V}_p$, and the viewing-transformation matrix is concatenated with matrix 7-13.

Assume that the polyhedron is to be positioned at the world-coordinate origin, and the view plane is beyond the coordinate extents of the polyhedron along the viewing z axis. Input coordinates for the polyhedron vertices are then transformed to projection coordinates.

7-3. The procedures from the preceding exercise can be modified to apply a rotation transformation before the projection transformation. Rotation parameters can be specified as input, or a number of predefined rotations can be generated. For all variations, coordinate-axis rotations can be used (Section 5-10)

7-4. For this problem, the object can be a cube with its center at the coordinate origin, as in Fig. 7-44. Each face of the cube is parallel to one of the coordinate planes. A one-point perspective projection is then obtained by setting the projection reference point on one of the coordinate axes and placing the projection plane between the cube and the projection reference point, with that coordinate axis perpendicular to the projection plane. If the projection reference point is on the z axis, for example, transformation equations 7-18 can be used to compute the transformed vertex positions.

7-5. A two-point perspective projection is obtained by positioning the projection plane so that it is parallel to only one of the tree principal axes of an object. This can be accomplished for the cube in the preceding exercise by rotating the cube so that two of its principal axes intersect the projection plane. If the projection plane is positioned on the z axis, for instance, a y-axis rotation could be applied to rotate the cube through a 45° angle before performing the perspective calculations.

7-6. A three-point perspective projection is obtained by positioning the projection plane so that it intersects all three principal axes. This can be accomplished for the cube in the preceding exercise by rotating the cube around both the y and x axes before applying the perspective-projection transformation, assuming the projection plane position is on the z axis.

7-7. This routine could be limited to symmetric frustums (Figs. 7-47 and 7-51). The transformation calculations can then be performed for an input set of object vertices using matrix 7-26, with the homogeneous coordinates divided by parameter h. In addition, arbitrary values can be used for the z-coordinate normalization parameters, or clipping-window coordinates can be supplied as input. In general, however, the perspective transformation could be carried out using matrix 7-34.

7-8. For the three-dimensional Cohen-Sutherland line-clipping algorithm, a 6-bit region code (Section 7-11) is used to identify the position of a line endpoint relative to the bounding planes of the normalized symmetric cube. A line segment with the region code 000000 for both endpoints is completely inside the clipping region (the normalized view volume). As in two-dimensional clipping, this condition can be detected by combining the two region codes for a line segment using the logical *or* operation, which yields the value *false* (000000) for an inside line. Similarly, a line segment is completely outside the view volume if the *and* operation produces a *true* value (not 000000) for the two region codes of the line. Otherwise, an input line segment must be tested for intersection with the boundaries of the cube.

Input to the algorithm are the coordinates $(x_0,\ y_0,\ z_0)$ and $(x_{end},\ y_{end},\ z_{end})$ for a line segment, and we can extend the Cohen-Sutherland algorithm to three-dimensions by describing a line segment using the following pair of equations.

$$y = y_0 + m_{xy}(x - x_0), \qquad z = z_0 + m_{xz}(x - x_0)$$

The first equation is written in terms of the slope for the projection of the line in the xy plane, and the second is written in terms of the slope for the projection of the line in the xz plane.

The symmetric normalized cube surfaces are described with the equations $x = \pm 1$, $y = \pm 1$, and $z = \pm 1$. An intersection calculation is performed using one of these six plane equations and the line equations. For example, for the plane $x = 1$, first determine whether the value 1 lies within the x interval from x_0 to x_{end}. If it does, then calculate the y and z values and check to determine whether these are valid line coordinates. As in two-dimensional line clipping, the input line is processed against each of the clipping

boundaries until either the entire line has been clipped or the inside portion has been identified.

This problem could be extended to combine the clipping operations with other parts of the viewing pipeline. For example, the clipped line could be viewed using a parallel projection.

7-9. Input coordinates for a three-dimensional line segment are $(x_0,\ y_0,\ z_0)$ and $(x_{end},\ y_{end},\ z_{end})$. Thus, the Liang-Barsky line-clipping algorithm (Section 6-7) is extended to three-dimensions by including a third parametric equation for the line:

$$z = z_0 + u\,\Delta z, \qquad 0 \le u \le 1$$

Also, inequalities 6-17 and 6-18 now include the conditions

$$zw_{min} \le z_0 + u\,\Delta z \le zw_{max}$$

$$u\,p_k \le q_k, \qquad k = 1, 2, 3, 4, 5, 6$$

and the algorithm now processes the line segment against a clipping volume instead of a clipping rectangle.

As an option, the clipping region could be normalized before processing the line segment. If the clipping region is also symmetric about the origin, the minimum and maximum coordinate values for the clipping region are ± 1.

7-10. Input for this procedure is the set of vertices for each face of a polyhedron. Assume that all faces are convex polygons. As in the Sutherland-Hodgman algorithm (Section 6-8), the edges for each polygon surface can be processed through the clipping boundaries to form a set of output vertices. For the Liang-Barsky algorithm, a parametric representation is used for the polygon edges (Exercise 7-9), and the parallelepiped can be aligned so that its faces are aligned with the coordinate planes. To simplify the calculations, the clipping volume could be normalized to a symmetric cube.

Three-dimensional region codes can be used to quickly identify those edges that are completely inside the clipping volume and those that are completely outside one of the clipping planes. Intersection calculations are then applied to the remaining lines to locate those sections of the polygon faces that are inside the clipping region (see Exercise 7-13).

In general, the parametric equations can be used to locate an edge intersection position with an infinite clipping plane. Then, tests can be applied to determine whether the calculated intersection position is within the bounds of the clipping region. The bounds are ± 1 if the clipping region is a normalized symmetric cube.

7-11. Input values are the coordinate positions for a line segment. In addition, a clipping region could be specified, otherwise use the normalized symmetric cube as the clipping region.

Three-dimensional region codes are first assigned to each endpoint, then the logical *or* and *and* operations are performed to determine whether the line can be saved or rejected without further processing. If not, intersection positions with the clipping planes are

calculated and tested to determine whether any portion of the line is to be saved. The methods for performing these calculations and tests are described in Section 7-11 for the normalized symmetric clipping volume.

7-12. For this problem, a frustum could be defined with a set of vertex positions, or it could be defined using a projection reference point, near and far clipping planes, and a clipping window. Plane parameters can then be computed for each face of the frustum, and the clipping boundaries are described with equations of the form

$$A\,x + B\,y + C\,z + D = 0$$

We can also write the plane equation in vector form as

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

where vector $\mathbf{N}$ is the normal vector for the plane and $\mathbf{P} = (x, y, z)$ is any position on the plane.

A polyhedron can be specified with a set of vertex positions $\mathbf{V}_j$ for each surface facet. And a vector parametric representation for edge k of a surface facet is

$$\mathbf{P} = \mathbf{V}_k + (\mathbf{V}_{k+1} - \mathbf{V}_k)u, \qquad 0 \le u \le 1$$

where $\mathbf{P}$ is a position on the edge, $\mathbf{V}_k$ is the first endpoint of the edge, and $\mathbf{V}_{k+1}$ is the second endpoint. Combining the vector edge equation with the vector plane equation, we obtain a value for u

$$u = -\frac{\mathbf{N}\cdot\mathbf{V}_k + D}{\mathbf{N}\cdot(\mathbf{V}_{k+1} - \mathbf{V}_k)}$$

If this value is outside the range from 0 to 1, the edge does not intersect that clipping plane. If it does intersect the clipping plane, we next need to determine if the intersection point is within the bounds of the frustum face that lies in that clipping plane.

Process the input vertex list for each polyhedron surface facet to determine edge intersections with the frustum clipping surfaces, producing an output list of clipped vertex positions. The surfaces for the clipped polyhedron are specified by the set of output vertex lists.

Thus, frustum clipping involves computation of the plane parameters, and intersection tests using the general form of the plane equation. To simplify the frustum clipping, the frustum could be transformed into a rectangular parallelepiped. Region testing and intersection calculations are then performed for clipping planes that can be aligned with the coordinate planes.

Clipping routines for a rectangular parallelepiped clipping region are much simpler than those for frustum clipping. If the surfaces of the specified parallelepiped are not aligned with the coordinate planes, the parallelepiped can be rotated so that its surfaces are aligned with the coordinate planes. Then the clipping boundaries are described with a plane equation that involves only one coordinate parameter. This allows the use of region-code testing and simple intersection calculations.

7-13. All surfaces of the polyhedron are convex polygons, and the input for this procedure is a vertex list for each surface. A three-dimensional region code (Section 7-11) is assigned to each surface vertex to identify the position of the vertex relative to the clipping boundaries. For each polyhedron surface, the vertices are processed in order around the perimeter of the surface, using methods similar to those illustrated in Fig. 6-26, to produce a clipped vertex list for the surface.

Each edge is processed against each clipping boundary. As each edge of the polyhedron surface is processed against a clipping plane, test the positions of the edge endpoints to determine the relative position of the edge. If the edge is completely outside that clipping boundary, proceed to the next surface edge. If the edge is completely inside, add the second edge endpoint to the output vertex list for that surface and process the next clipping boundary or the next edge. Otherwise, determine the direction of the edge crossing (either "in-out" or "out-in") for that clipping plane and calculate the intersection position. Since the clipping planes are aligned with the coordinate planes at a unit distance from the coordinate origin, an intersection position is obtained by substituting a value of $+1$ or -1 for one of the coordinates in the line equation for that edge and solving for the other coordinate. The output vertex list is then updated, and the clipping boundary or next surface edge is processed. The final set of vertex lists contains the clipped vertex positions for the polyhedron.

7-14. Input for this routine can be a simple object such as a wire-frame regular tetrahedron (four-sided pyramid). And the output can be a simple orthogonal projection of the object. A split-screen view, as in Fig. 6-10, can be used to show the object before and after clipping.

Solid-color surfaces could be used instead of a wire-frame representation. Each surface of the polyhedron could be displayed in a different color. Another option is to display all surfaces using the same fill color, but display surface edges in a different color.

7-15. Clip a convex polyhedron against a normalized symmetric view volume using methods similar to those discussed for the preceding two exercises. Surface edges can be described with a parametric representation, and the edges are processed through the clipping procedures using the homogeneous-coordinate equations given in Section 7-11.

7-16. The example program can be modified in various ways to provide for user input. For example, statements can be added to procedure `init` to display a screen prompt and accept keyboard input for the viewing-coordinate origin.

Also, the output display could be modified by user input. A solid fill color or a wire-frame representation could be selected for either the front or the back face of the square.

7-17. As in the preceding exercise, a screen prompt and commands for accepting keyboard input can be added to procedure `init`. For this exercise, the frustum parameters are specified as user input. In addition, the viewing position could be selected by the user, as well as the other viewing parameters.

7-18. A file containing the list of vertex coordinates and other parameters for each surface of the polyhedron can be provided as the input for this program. A call to an input routine can be used in procedure `displayFcn` to replace the specification of the quadrilateral. The input routine then passes the polyhedron information to procedure `displayFcn`.

In addition to the vertex coordinates, the data file can contain the color for the front and back of each polyhedron surface, the color for the surface edges, and other surface information. For example, one side of each polygon can be filled with a specified color and the other side can be displayed using a wire-frame representation, with a different color for the surface edges. Front faces would then be displayed in a solid fill color and an edge color, while the back faces of the surfaces would be displayed as wire-frame polygons.

7-19. Replace the OpenGL perspective functions in the program with the functions for generating an orthogonal projection (Section 7-10).

7-20. Input parameters can be used to specify the components of the oblique parallel projection vector. There are no oblique-projection functions in OpenGL, but an oblique-projection view of the polyhedron can be generated using the methods discussed in Section 7-7. One way to do this is to set the current OpenGL projection matrix to the normalized oblique-projection matrix. Another method is to rotate the object so that the projection vector is perpendicular to the projection plane, then perform an orthogonal projection.