Chapter 6

Two-Dimensional Viewing

Exercises for this chapter deal with the two-dimensional viewing pipeline. This includes problems relating to display windows and viewports, transformations from world to viewing coordinates, clipping procedures, and applications of the OpenGL two-dimension viewing functions.

Exercises

6-1. Matrix 6-1 is formed by first translating $\mathbf{P}_0$ to the world coordinate origin, then rotating the viewing-coordinate axes to align them with the world-coordinate axes. The elements of the rotation matrix can be obtained from axis vector $\mathbf{V}$ using the methods described in Section 6-2.

6-2. First set up the matrix for scaling the coordinates of the clipping window vertices, relative to the window center position, so that the window is the same size as the viewport. Next, premultiply the scaling matrix by a matrix that translates the coordinates of the window center point to the viewport center point. Then express the elements of the composite matrix in terms of the scaling and translation parameters in Eqs. 6-4 and 6-5.

6-3. Input to this procedure is the set of coordinates for the clipping-window vertices. The elements of the transformation matrix are then calculated from a sequence of scale-rotate operations, as in the preceding exercise. Any position on either the clipping window or the normalized square can be used as the fixed point for the transformation sequence.

6-4. A circle and a few polygons, such as a triangle, rectangle, and hexagon, can each be defined relative to the coordinate origin. For example, each polygon could be defined with its centroid coordinates as (0, 0). Then a scene is constructed by positioning the objects within a world-coordinate space. Viewing coordinates and a clipping window are defined using the methods described in Section 6-2, and the normalized-square transformation is given in Section 6-3.

A viewing table can be set up as a list of values for viewing-coordinate parameters $\mathbf{P}_0$ and $\mathbf{V}$ (Section 5-8), along with the coordinate limits for the clipping window. Each set of values can be referenced with a numerical index, such as the list position, and an input

index value specifies the view for the scene. In this way, various rotated and clipped views can be displayed by selecting different indices.

6-5. To complete the Cohen-Sutherland line-clipping code in Section 6-7, add the commands for specifying a clipping rectangle, generating pixel positions along the line path, setting the line color, setting the display-window color, setting the projection parameters, and setting the display-window parameters. Routines from the example programs in previous chapters can be used for this purpose. The clipped portions of lines could be displayed in a different color. Another option is to show the edges of the clipping rectangle as dashed lines.

6-6. The details of the Liang-Barsky line-clipping algorithm are given in Section 6-7.

6-7. For a given line, count the number of clipping operations performed by each of the two algorithms. Line orientations can include a line that intersects all four clipping-window boundaries (Fig. 6-15), a line that has one endpoint outside and one endpoint inside the clipping window, a line that is completely inside the clipping window, a line that is completely outside one of the clipping-window boundaries, and an outside line that intersects one or more of the clipping-window boundaries. The order for processing the line endpoints and the window boundaries could be specified for each line.

6-8. To complete the Liang-Barsky line-clipping code in Section 6-7, add the commands for specifying a clipping rectangle, generating pixel positions along the line path, setting the line color, setting the display-window color, setting the projection parameters, and setting the display-window parameters. Routines from the example programs in previous chapters can be used for this purpose. The clipped portions of lines could be displayed in a different color. Another option is to show the edges of the clipping rectangle as dashed lines.

6-9. The symmetry transformations can be accomplished using either reflections or rotations. An endpoint that is beyond the right clipping-window edge, between the top and bottom window boundaries, can be mapped into the left-edge region with a reflection about the vertical centerline of the window, or it can be transformed using a 180° counterclockwise (or clockwise) rotation with respect to the lower-left corner of the window. Once a left-edge intersection is calculated, the inverse transformation is performed to obtain the corresponding right-edge coordinates.

An endpoint position directly above the top window edge is transformed to the left-edge region using a 90° counterclockwise rotation with respect to the lower-left window corner. Similarly, an endpoint position directly below the bottom window edge is transformed to the left-edge region using a 90° clockwise rotation with respect to the lower-left window corner. In both cases, the inverse rotation is carried out after the edge intersection is calculated.

An endpoint in the upper-right corner is transferred to the upper-left corner using a reflection about the vertical centerline of the window. An endpoint in the lower-left corner is transferred to the upper-left corner using a reflection about the horizontal centerline

of the window. And an endpoint in the lower-right corner can be transformed to the upper-left corner using two reflections. First transfer the point to either the upper-right or the lower-left corner, then reflect it into the upper-left corner. For each case, after the edge-intersection position is located, transform the intersection coordinates to the correct edge using the inverse reflection (or rotation) calculations.

6-10. The details for the NLN line-clipping algorithm are given in Section 6-7 for the three regions shown in Fig. 6-16, and the symmetry operations from the preceding exercise can be used to complete the processing steps for the remaining clipping-window regions.

6-11. For a given line, count the number of clipping operations performed by each of the three algorithms. Line orientations can include a line that intersects all four clipping-window boundaries (Fig. 6-15), a line that has one endpoint outside and one endpoint inside the clipping window, a line that is completely inside the clipping window, a line that is completely outside one of the clipping-window boundaries, and an outside line that intersects one or more of the clipping-window boundaries. The order for processing the line endpoints and the window boundaries could be specified for each line.

6-12. An algorithm for Liang-Barsky polygon clipping can be set up using methods similar to those in the Sutherland-Hodgeman algorithm (Section 6-8). The major difference is that the Liang-Barsky algorithm describes line segments with parametric equations. Given an input list of vertices, the algorithm processes line segments in order around the perimeter of the polygon, and an output list of clipped vertices is produced. The general strategy described for Fig. 6-26 can be applied to the line-processing routine, using the intersection tests given in Section 6-7.

6-13. Section 6-8 lists the processing steps for the Weiler-Atherton polygon-clipping algorithm, which are illustrated in Fig. 6-29. Input to the algorithm is a list of polygon vertices, which can be given in counterclockwise order. As in the Sutherland-Hodgman algorithm, line segments can be classified according to their relationship to the clipping-window boundaries (Fig. 6-26). Process the input list of vertex positions sequentially, checking the polygon edges for intersection with the clipping boundaries and forming an output vertex list. When an "in-out" edge is encountered, add the intersection position to the output list and process the window edges, in counterclockwise order from the intersection point, until the next intersection with a polygon edge is encountered. This completes an output vertex list for a clipped section of the polygon. Set up the next output vertex list, and continue processing the input vertex list from the window intersection position with the "in-out" edge.

6-14. Basically, the processing steps for this algorithm are the same as those for the algorithm in the preceding exercise. The only difference is that the clipping boundaries are now described with general line equations. For a standard rectangular clipping window, the clipping boundaries are formed with horizontal and vertical lines. But with a general polygon, we need to describe a clipping boundary with either a parametric representation or the slope-intercept equation. Thus, more computation is needed to determine whether

two line segments intersect and to locate intersection coordinates. These calculations are discussed in detail in the solution to Exercise 3-35.

For this exercise, the clipping region is a convex polygon. Therefore, a line segment in an input polygon fill area can intersect at most two edges of the clipping region.

6-15. This exercise uses the same processing steps as in the preceding exercise. However, for a concave-polygon clipping region, an edge of an input polygon fill area can intersect more than one edge of the clipping region.

6-16. First the coordinate extents of the ellipse can be compared to the clipping-window limits. If there is no overlap, the ellipse is entirely clipped. If the coordinate extents are completely inside the clipping window, all of the ellipse is displayed. Otherwise, boundary intersection tests are necessary.

Each of the four clipping-window boundaries can be tested for an intersection with the ellipse equation 3-37. These intersection calculations determine whether any part of the ellipse is within the interior of the clipping window. After all ellipse intersections have been calculated, the portion of the ellipse within the clipping window can be displayed using the midpoint ellipse algorithm.

Alternatively, each pixel position along the ellipse boundary can be tested for a window intersection as it is generated, and only those pixels inside the window are displayed. For a filled ellipse, each scan line across the ellipse can be tested for window intersections. Sections of each scan line that are outside the window are then clipped.

6-17. All characters are the same width, and the coordinate extents of each character are compared to the clipping-window boundaries. If the rectangular limits of a character are completely inside the window, the character is saved for display. Otherwise, it is completely clipped. This procedure is applied to each character in a string by stepping from one character to the next using a fixed-width step size.

6-18. First check the coordinate extents of a character grid against the clipping-window boundaries. If the grid is completely inside the window, it is saved for display. If the grid is completely outside the window, the character is deleted from the display list. Otherwise, the display includes only those parts of the charter grid that are inside the window. All pixel positions that are not inside are clipped.

As a modification of this exercise, outline fonts could be clipped. In this case, clipping procedures are applied to the individual line segments for each character.