Chapter 11

Interactive Input Methods and Graphical User Interfaces

Problems and projects for this chapter explore input methods and user-interface design for graphics packages, as well as OpenGL input functions and their applications.

Exercises

11-1. A set of objects, either two dimensional or three dimensional, can be defined with respect to the coordinate origin. Object names, such as red circle, circle3, or large triangle, can be assigned and listed in the menu. Or, the objects could be shown along the upper or lower part of a display window along with their names. A mouse (or other locator device) can then be used to select an object either from the menu listing or the object display. Then a locator device selects a coordinate position within the display area, and the selected object is translated to that position.

The reference position for any object can be the center (or centroid) point. Alternatively, some other position, such as a vertex of a polygon or polyhedron, could be used as the object reference position. And the reference positions could be shown on the object display in some way, such as marking the position with a crosshair or red dot.

11-2. A second menu listing transformation options is to be created for the algorithm of Exercise 11-1, and submenus can be used to list the transformation parameters. After selecting an object and a coordinate position, the user can then access the transformation menus. Once a particular transformation is selected, submenus can list options for the parameters. For example, if a user selects a scaling transformation, a list of fixed-point positions, such as the coordinate origin and the object centroid, can be presented in a pop-up menu. Similarly, another pop-up menu can list options for each of the coordinate scaling parameters.

Instead of presenting a fixed set of parameter options, an interactive dialogue could be used to obtain any values for the transformation parameters. The system can query the user for a value of a particular parameter, and the parameter value could be entered from the keyboard. Another option is to display a graphical representation of a slider or other valuator device for entering numerical values (Fig. 11-1).

11-3. For this procedure, a menu can be set up, as in Fig. 4-7, so that various brush styles and line options can be selected. Moving an activated stroke device then generates a drawing in the selected style. A procedure must then be devised for generating each stroke option as the input device is moved.

Each time the stroke device is deactivated, another menu option can be selected. The menu can be provided as a permanent part of the display area, or a pop-up menu could be designed.

11-4. A pattern can be characterized according to its attributes and geometric features, such as color, the number of horizontal and vertical lines in the pattern, and the thickness of each line. A pattern table then contains the description for each possible input character. If a character is formed as a pixel grid, the individual pixels in an input pattern can be tested to determine their colors and relationship to each other. For example, a standard thickness horizontal line is stored as a row of color pixels (same y value) with no pixels on the grid lines immediately above and below that line.

11-5. The numeric scale can be horizontal or vertical, and the slider can be moved with either a locator or a stroke device. If a locator device is employed, the position of the slider is controlled with a point-and-click operation. If a stroke device is employed, the slider can be dragged so that the position of the slider is shown as the device moves.

One end of the linear scale represents the minimum scalar value, s_{min}, and the position at the other end corresponds to the maximum scalar value s_{max}, with the slider initially positioned at the minimum value. Labels should be placed at intervals along the slider to indicate the values represented at those positions.

The linear scale can be displayed as a thin rectangle, and the slider can be displayed as a small triangular arrow beneath the scale, as in Fig. 11-1. Instead of an arrow pointer, a narrow rectangle on the scale could be used to represent the slider. Slider position is constrained as a translation along the linear scale according to relative movement of the interactive input device. And scale positions are determined by the position of some point on the slider representation, such as the tip of an arrow pointer.

As the slider is moved, the scale position is converted to a scalar value s within the range represented by the scale. As an example, the scalar value at an x-coordinate position along a horizontal scale is computed as

$$s = s_{min} + (x - x_{min})\frac{s_{max} - s_{min}}{x_{max} - x_{min}}$$

The computed scalar value s is then displayed in the echo box near the scale.

To ensure that each input coordinate position $(x,\ y)$ is near the scale line, the slider can be activated only when a selected input position is within a bounding rectangle specified for the graphical slider representation. Otherwise, the device input can be ignored.

11-6. Procedures for this routine are similar to those for a linear scale (Exercise 11-5). Either a locator or a stroke device can be used to move the scale slider. A circular scale can be represented using a thick circular line, with several choices for the slider representation. A radial arrow from the circle center position $(x_c,\ y_c)$ to the scale could be used to represent the slider. Alternatively, the slider display could be a small triangle next to the circular scale or a small rectangle on the circular scale.

Labels should be placed at intervals around the circular scale to indicate the angular degree values represented by the interval positions. The circle position $(r,\ 0)$ corresponds to $0°$, with increasing angular values around the scale in a counterclockwise direction.

As in the routine for a linear scale, a bounding rectangle can be defined around the slider, so that the slider is activated only when the input position is within the bounding rectangle. And scale positions are determined by the position of some point on the slider representation, such as the tip of an arrow pointer.

An input position $(x,\ y)$ on the circular scale is converted to an angle value with the calculation

$$\theta \;=\; \tan^{-1} \frac{y - y_c}{x - x_c}$$

This radian value is then converted to degrees

$$\theta_{deg} \;=\; \frac{\theta \cdot 360}{2\,\pi}$$

and displayed in the echo box.

The circular scale could also be used to select values other than angles. In that case, the scale labeling changes and an angular position on the scale must be converted to the corresponding data value represented by that position on the circle. For example, positions around the circle could represent a percentage from position $(r,\ 0)$. And two input positions on the scale could be used to obtain the circle percentage of the selected circular arc, which could represent a section of a pie chart.

11-7. A user interface could be developed for this program so that a user could choose an action, such as displaying a line segment or terminating the program. Options could be presented in a menu, or an interactive dialogue could be used to query a user for an action. For example, if a line segment is to be displayed, the user can be asked to input the coordinates for the first endpoint, then the second endpoint. As each pair of endpoint coordinates is selected, a line-drawing routine is invoked to display the line segment.

Drawing options could also be included in this program to allow a user to select a previously displayed line so that in can be modified in some way. A selected line could be deleted or one of its endpoints moved to a new position. Line selection can be accomplished by using the input device to perform a picking operation with a specified pick window (Section 11-2).

11-8. Modify the program in the preceding exercise so that gravity-field regions are defined for each line segment as it is generated. A gravity field around each of the two line endpoints can be a small circle or square. And a gravity field for points along the line between the two endpoints can be specified using a rectangular region. Processing is simplified if line connections are restricted to endpoint positions. In that case, only the two endpoint gravity-field regions are needed for each line segment. A gravity-field table can be set up to store an identifying label for each region, along with the region parameters. The gravity-field parameters, such as the radius of a circle, edge length of a square, and a perpendicular line segment distance, could be specified as input or as menu options.

After the first line is generated, successive input positions are checked against existing gravity fields. If an input position $(x,\ y)$ is within some gravity field, the position is moved to a point on the line segment associated with the gravity field. Otherwise, position $(x,\ y)$ is an endpoint for the new line.

An input position $(x,\ y)$ is within a circular gravity field with center coordinates $(x_c,\ y_c)$ and radius r if

$$(x - x_c)^2 + (y - y_c)^2 \ \le\ r^2$$

And an input position $(x,\ y)$ is within a square gravity field with center coordinates $(x_{sq},\ y_{sq})$ and edge length l_{sq} if

$$x_{sq} - \frac{l_{sq}}{2} \ \le\ x \ \le\ x_{sq} + \frac{l_{sq}}{2}, \qquad\qquad y_{sq} - \frac{l_{sq}}{2} \ \le\ y \ \le\ y_{sq} + \frac{l_{sq}}{2}$$

If an input position is not within an endpoint gravity field, it can next be checked against any rectangular gravity fields defined for line segments. The vertices for a rectangular gravity field around a line segment can be computed using the methods for creating thick line segments (Section 4-5), and inside-outside tests (Section 3-15) can be used to determine if an input position should be moved onto a line segment. Alternatively, the perpendicular distance from an input point to a line segment (Eq. 11-2) can be used to test the position of the input point relative to a rectangular gravity field region and to calculate a connection point at a position along the line path.

11-9. Set up a menu for the previous exercise that allows a user to activate and deactivate a line constraint that generates either a horizontal or a vertical line, given two endpoint postions. When the line-constraint option is selected, the horizontal and vertical separations of the endpoints are compared. For two endpoint positions $(x_1,\ y_1)$ and $(x_2,\ y_2)$, a vertical line is displayed from $(x_1,\ y_1)$ to $(x_1,\ y_2)$ if $|x_2 - x_1| < |y_2 - y_1|$. Otherwise, a horizontal line is displayed from $(x_1,\ y_1)$ to $(x_2,\ y_1)$. The constraints are applied after the gravity-field checks.

11-10. A menu can be used to allow a user to select and deselect a grid constraint. In addition, other possible menu options are the size of the grid cells (assuming the cells are squares), a “show grid” option that can be toggled on and off, the color of the grid lines, and the position of the grid within the display area. By default, the grid could be displayed over the entire display area, with a preset gird size and line color.

Each input coordinate position is moved to the nearest grid intersection point, and a straight-line segment is displayed after each two positions have been selected. A table of grid intersection coordinates can be set up to facilitate the search for the nearest grid intersection.

11-11. A stroke device can be used to generate coordinate positions continuously as the input device is moved. The first position selected is the starting endpoint for a line segment, and, as the input device moves, the line segment is displayed from the starting position to each successive coordinate position. At each step, the previously displayed line is erased before the next line is displayed. A button activation or a "double click" can be used to end the rubber-band process and display the line segment from the initial point to the final point.

11-12. The procedure from the previous exercise is expanded to include the rubber-band display of rectangles and circles by providing a menu of object choices and associated routines for generating each of the objects (Section 11-4). Once an object type has been selected, the appropriate routine is invoked to generate the object from an initial position as the input device moves.

For a straight-line segment, the initial input position specifies the coordinates for the first endpoint. For a rectangle, the initial input position specifies the coordinates of a vertex, and subsequent input positions give coordinates for the diagonally opposite vertex. For a circle, the initial input position specifies the coordinates for the circle center, and subsequent input positions are points on concentric circle boundaries.

11-13. Objects in the two-dimensional scene can be limited to simple fill areas, such as circles and polygons, and a picking operation (Section 11-2) is used to select an object, given an input position within the scene. The picking operation can be initiated from a menu or with a button or "double click" input. And picking ambiguities could be resolved using an interactive dialogue. When the input position is within the coordinate extents of two or more objects, one of the objects can be highlighted by changing its color and a confirmation can be requested from the user. If the user rejects the object, another overlapping object is highlighted and the confirmation request is repeated. If the user rejects all objects, the picking operation is terminated. Alternatively, picking ambiguities can be resolved by locating the nearest object to the input position, using the methods discussed in Section 11-2. A picked object can be identified in some way, such as changing color or blinking on and off.

11-14. In some region of the display area, generate a number of nonoverlapping shapes, such as a circle, a triangle, and a rectangle. Initiate a dragging operation by selecting a position within the coordinate extents of one of the displayed objects, then moving the activated input device to another position. A series of translation operations are then performed to display the object along the path of the moving input device until the dragging operation is terminated. A button or "double click" input can be used to set the final position of the object. Any number of dragging operations can be performed to construct the scene.

11-15. A set of request-mode functions can include an input function for each of the six logical device classes. Also, a device code could be assigned to each input device, such as a mouse and a keyboard, and an additional function can be used to activate those device codes that are to supply data in request mode. The arguments for an input-device function can include a device code and data parameters, such as an array for stroke input and the size of the array. Similarly, the arguments for a request-mode string function would include a parameter for the string and another parameter for the number of characters in the string. Also, an error message should be generated if invalid data is entered or if a specified device has not been activated. An input device that has been activated in request mode does not generate data until an input command is issued by the graphics program.

11-16. Sample-mode functions for each of the six logical device classes can be similar to those for request mode (Exercise 11-15). However, in sample mode, a device can generate and store data without waiting for a request from the application program. When an input function is specified in the program, the stored value (or, values, depending on the type of input) is retrieved. Also, a device that has been activated in sample mode can continually update input data values by replacing previous values with currently sampled values.

11-17. Input devices operating in event mode collect and store data values in an event queue. Thus, the functions for event mode are the same as those for sample mode (Exercise 11-16) except that all input data are saved, so that previously stored values are not replaced with current values.

11-18. The function for Exercises 11-15, 11-16, and 11-17 are to be combined into one package. This implementation could be limited to a subclass of logical devices, such as locator, stroke, and, string devices. Also, an input device can be activated in more than one mode. For example, a mouse could be activated to input stroke data as well as locator data.

11-19. For this exercise, two menus could be set up, one for color choices and the other for point-size choices, or a single menu with two submenus could be devised. The color choices can include a range of values for the *RGB* components, and the point-size choices can include integer values from 1 to 5, for example. Then the `glColor` and `glPointSize` statements in procedure `displayFcn` are replaced with user input from the menus.

11-20. The example polyline program in Section 11-6 displays line segments using a default style, color, and width. Menus can be set up for these parameters similar to those for the point-plotting parameters in Exercise 11-19. A range of values can be provided for the *RGB* color components and the line width, and style options could be limited to two or three choices, such as solid, dashed, or dotted. The various line styles can be generated with the OpenGL line-style function discussed in Section 4-8.

11-21. Options in the style menu from Exercise 11-20 can be modified to include various line-texture patterns, such as those shown in Fig. 10-114. And the example line-texture program from Section 10-21 can be modified and incorporated into the program from the preceding exercise.

11-22. A menu of values for the *RGB* components of a color can be set up for this program. Using a mouse, a user selects a coordinate position $(x,\ y)$ and a set of *RGB* values. A square fill area, with vertices at positions $(x \pm 50,\ y \pm 50)$, is then displayed in the selected color. Instead of displaying a fixed-size square, the program can include input statements so that the width w and height h for a rectangular area can be specified.

11-23. For each input position, calculate the coordinates for the four vertices of the rectangle and compare the vertex coordinates to the coordinate extents of the display window. If any x or y vertex coordinate is outside the limits of the display window, the input point is rejected.

11-24. Any surface or linear texture pattern could be mapped to the rectangle, such as a black-and-white checkerboard pattern or the multicolored lines shown in Fig. 10-114.

11-25. An input character string can be displayed at a selected position using the OpenGL character functions of Section 3-21 and the mouse functions from Section 11-6.

11-26. For this program the dragging operations discussed for Exercise 11-14 are to be implemented using the OpenGL mouse functions (Section 11-6) to position a single selected object within the display area.

11-27. Expand the preceding program to construct a scene by dragging any number of selected objects into position within the display area. For example, a panel truck or a fruit tree could be constructed from a menu of rectangle and circle shapes.

11-28. Parameter values for a selected scaling or rotation transformation could be entered from the keyboard, or the selected transformation could be applied interactively. A scaling transformation, for instance, could be applied by selecting and dragging some point on the object, such as a rectangle vertex, relative to the centroid of the object, so that all points on the object would expand or contract relative to the centroid position. And a rotation could be applied by dragging some object point around the centroid of the object. The angular displacement for the point is calculated, and all other object positions are then rotated by this angular amount to produce the rotated display of the object.

11-29. For this exercise, the program of Exercise 11-26 is to be modified so that a set of three-dimensional GLUT wire-frame shapes is displayed and one of the shapes can be dragged into a position within the display area. Example programs for displaying the GLUT wire-frame shapes are given in Sections 8-2 and 8-6.

11-30. The GLUT functions for displaying three-dimensional shapes are given in Sections 8-2 and 8-6. And the OpenGL illumination and surface-shading functions are discussed in Section 10-20.

11-31. Modify the example picking program in Section 11-6 for interactive selection of three-dimensional objects. First, construct a three-dimensional scene containing a few basic objects, such as a box and a sphere, and assign an integer pick identifier to each object. Next, set up the viewing parameters for the scene, using either an orthogonal projection or a perspective projection, and display the scene on a selected view plane. Implement the OpenGL picking operations, with the pick-window dimensions predefined or specified as input. Then pick a distant object in the display window, translate the picked object to the foreground, and redisplay the scene.

11-32. Combine the interactive polyline-drawing program from Section 11-6 with the Bézier curve-drawing program in either Section 8-10 or Section 8-18. For the combined program, four control-point positions are to be selected in the display window using mouse input. The four input positions are then passed to the Bézier routine to display the curve. Optionally, the control points and control graph could also be displayed, and curve parameters, such as style, color, and width, could be specified as input.

11-33. To display a fourth-degree Bézier curve, enter five control-point coordinate positions. To display a fifth-degree Bézier curve, enter six control-point positions. An interactive dialogue could be used to request the coordinate positions until a termination signal is entered, allowing a maximum of six input positions and a minimum of three input positions. Or the number of control points could be specified as keyboard input.

11-34. Combine the interactive polyline-drawing program from Section 11-6 with a cubic B-spline curve-drawing program, using the routines from Section 8-18. The degree of the curve and the number of control points could be specified as keyboard input, or these parameters could be predefined. For the combined program, the control-point positions are to be selected in the display window using mouse input. The input positions are then passed to the B-spline routines to display the curve. Optionally, the control points and control graph could also be displayed, and curve parameters, such as style, color, and width, could be specified as input.

11-35. The program from Exercise 11-32 can be modified to display a surface patch, using the OpenGL Bézier-spline surface functions from Section 8-18. Various viewing parameters could be specified for the program so that the surface is viewed from different positions. Either an orthogonal projection or a perspective view can be generated for the patch as a wire-frame or solid surface section.

11-36. A statement of application objectives should be given as the first step in the design process. This statement defines the scope and intent of the package. For example, the model might be defined for a facility planning system, which allows the placement of equipment and furniture into areas of a room or a set of rooms. Some options might also be available to the user, such as defining the relative sizes of the furniture items or introducing additional items. Once the objectives have been stated, detailed information for the system can be given. This includes definition of the symbols (icons) to be used for the objects in the model, the information and terms presented to the user, and the operations available to the user.

11-37. Possible help facilities (Section 11-8) include tutorials, example applications, prompts, and backups. More prompts and explanations are valuable for someone learning to use the system. Less prompting allows an experienced user to operate more effectively.

11-38. Methods for dealing with backup and errors are discussed in Section 11-8. More sophisticated methods with many options are better for an experienced user, and simple, easily understood methods are better for beginners.

11-39. User dialogues and menu options (Section 11-8) can be designed with options for accommodating different skill levels. Brief dialogues and menus allow experienced users to develop an application quickly, while more extensive and detailed menus allow a beginner to use the system more effectively and with less frustration.

11-40. As with other aspects of a user interface, feedback methods (Section 11-8) can be presented with several options. This allows an experienced user to eliminate detailed feedback dialogues and to work faster, and an inexperienced user can select more detailed feedback options.

11-41. Methods for designing screen layouts and window managers are given in Section 11-8. Basically, screen layouts are designed so that multiple windows, menus, and other items can be displayed without interfering with each other and the work area. For example, icons, sliders, and pop-up menus should be placed to one side of an interactive display window so that they do not interfere with one another. Window managers are responsible for opening, closing, positioning, and resizing display windows.

11-42. Select a particular application area, and design a window manager that can process multiple display areas, menus, and other items. The window manager should be able to handle overlapping display areas, moving windows to the background and foreground, and the other operations discussed in Section 11-8.

11-43. The user model for a paint program describes the functions and capabilities of the program. Methods for input are to be explained, along with the types of allowed input. For instance, the program could allow a user to draw lines and paint color areas. Feedback, error messages, and other relevant information should be explained. Menus can be provided for choosing colors, brush size, and brush type. Menu placement should be considered relative to the work area. Also, "tear-off" menus and other positioning options can be provided for the user. Input includes options such as line drawing or painting color regions. In addition, options can be provided for dealing with multiple pictures and for saving pictures. And, an example application can be given to demonstrate the features of the paint system.

11-44. A two-level hierarchical model can be described as a tree structure with two levels below the root node. For example, a two-level description of an electrical circuit can consist of circuit sections and the components in each section. Thus, a user interface for a two-level hierarchical modeling package can present a set of icons for items such as resistors, capacitors, and conductors, and processing options, such as the methods for placing and arranging electrical components into various labeled circuit sections to construct amplifiers and other electrical devices.