Chapter 13

Computer Animation

Exercises for this chapter explore features of computer-animation programs, such as storyboards, raster methods, key-frame systems, morphing, and kinematic and dynamic motions, as well as animation implementations in OpenGL.

Exercises

13-1. The stick figure is to perform some type of motion, such as walking, running, jumping, climbing, swimming, or some acrobatic maneuver. Describe the motion in a storyboard layout (Section 13-2) and design several key frames, including some simple, line-drawing background for each scene.

13-2. Input for this program is a set of point positions for each key frame. These points define the positions of the figures joints and the end positions for each line segment (connecting link) in the figure. For each pair of key frames, generate two or three in-betweens, using linear interpolation to compute the successive placements of the point positions. The set of frames for the complete animation could be displayed using double buffering.

13-3. The storyboard layout and key frames for this exercise can include two or more stick figures, or one figure could be a rigid body, such as a truck or airplane, with no changes in the relative position of the objects connecting links as the object moves.

13-4. The program for this exercise is a modification of the program for Exercise 13-2, where each key frame now contains two or more objects.

13-5. Input for this program consists of two vertex lists, one for each of the two polygons, where one polygon has more vertices than the other. The morphing methods of Section 13-6 are then applied to the initial polygon to transform it into the second polygon, as illustrated in Fig. 13-10. Use linear interpolation to equalize either the number of vertices or the number of line segments in the generation of the in-betweens.

13-6. This exercise is a extension of the previous exercise to three-dimensions. Two input vertex lists again define the two objects, with the sphere specified as a set of surface patches, as in Exercise 8-1. The vertex positions on the sphere surface are then to be transformed into the polyhedron vertex positions, using linear interpolation calculations through the five in-betweens.

13-7. Use the methods of Section 13-6 to create an animation sequence involving an acceleration, where an object starts slowly and gradually moves faster and faster as the animation proceeds. The moving object could be a simple rigid object, or it could be a stick figure, as in Exercise 13-1.

13-8. For this exercise, the animation sequence could involve two objects, one accelerating and one gradually slowing to a stop. The acceleration is to be modeled with Eq. 13-7 and the decreasing speed is modeled with Eq. 13-8.

13-9. Modify the animation sequence from Exercise 13-7 so that the object first accelerates, then decelerates to a stop, and use Eq. 13-9 to model the animation sequence.

13-10. As an initial version of this animation program, a large hollow rectangle can be positioned in the center of the display area, with a small filled circle placed at a default position inside the rectangle boundaries. Then a keystroke can be used to initiate the motion of the circle, which is assigned a default initial velocity (direction and speed). The speed is specified as the distance (number of pixel positions) that the circle is to move along a linear path for each frame of the animation sequence, and the direction of travel can be defined with vector components specified relative to the coordinate origin, which could be defined at any convenient position, such as the lower-left corner of the viewing area.

When the action is initiated, the program first calculates the intersection position along the linear path of the circle with the rectangle boundary. This intersection calculation can be performed using the methods discussed for Exercise 3-35: Set up a parametric representation for the direction of travel and for each of the four rectangle boundaries, then locate the nearest edge intersection along the travel path.

At each stage of the animation, the distance from the circle center to the intersection point on the rectangle boundary is calculated. When this distance is less than or equal to the radius of the circle, the circle rebounds along the reflection direction, which makes an angle with the edge normal that is equal to the angle of incidence (as in ray-tracing calculations, Equation 10-63), and the angle of incidence can be calculated from the dot product of the unit velocity vector for the circle and the unit vector along the intersected edge. Intersection calculations are then repeated for the new path direction, and the circle continues to bounce around inside the rectangle for the duration of the animation sequence, which can be terminated after a fixed number of frames, a fixed number of circle rebounds, or using interactive keyboard input.

Once the basic program has been tested and debugged, options can be added, such as specifying circle parameters using interactive input from a keyboard, or from menu and mouse operations. These parameters include the radius of the circle, which should be small compared to the dimensions of the rectangle, and the velocity of the circle.

13-11. For this program, the rectangle can be oriented so that the width is somewhat less than the height, and then replace the lower horizontal edge with a moveable line segment (the "paddle") that is one-third the rectangle width. Interactive movement of this line segment can be controlled with the left-right cursor keys, which can move the line segment to any horizontal position between the left rectangle edge and the right edge.

Once the circle motion begins, the paddle can be moved back and forth to intercept the circle. If the circle arrives at a rebound position with the lower rectangle boundary and the paddle is not at that position, the circle escapes from the rectangle interior and the game is terminated. Each time the circle is intercepted at the lower rectangle boundary, the score is incremented by 1 and the updated score is displayed somewhere in the viewing area.

13-12. Modify the previous program to include motion acceleration by incrementing the speed of the circle after a set number of paddle interceptions. For example, the speed of circle could be slightly increased after each paddle interception, or a larger speed increment could be used after, say, three or four interceptions.

13-13. The program from Exercise 13-10 is to be modified so that a small sphere bounces around inside a box. Three-dimensional parameters are now needed to set the initial position and initial velocity of the sphere, and the sphere moves along a three-dimensional linear path between each "wall" intersection. Viewing parameters can be set up to display either a parallel projection or a perspective projection of the motion on a selected view plane, using a wire-frame representation for the box.

13-14. Input for this program can include parameters for the ball (radius and color), initial ball position $(x_0,\ y_0)$ relative to a ground plane and coordinate origin, the frequency value (number of cycles per second), the damping constant, the number of positions (for example 10 or 12) to be displayed per cycle, and the number of cycles to be displayed. In addition, keyboard input can be used to begin the animation, as well as to terminate it. The ball rebounds from the ground plane when the distance from the ball center to the ground plane is less than or equal to the ball radius (Exercise 13-10). As options, the movement of the ball could be terminated when the maximum amplitude per cycle falls below a preset value, and the ball and motion parameters could be selected from menu options.

13-15. Squash and stretch effects for a bouncing ball are illustrated in Fig. 13-4. As the ball drops, it can slowly stretch into an elliptical shape whose major axis is tangent to the motion path. When the center of the ball reaches the rebound position above the ground plane, the ellipse is rotated so that the major axis is parallel to the ground plane. Then, as the ball rebounds, the major axis of the ellipse is rotated so that it is tangent to the animation path, and the ellipse shape slowing evolves into a circle at the maximum rebound height. Termination conditions can be set as in the previous exercise.

13-16. For this program, the initial motion of the ball is along the parabolic path described by the parametric equations 3-53. Input parameters include ball color, initial ball position, initial ball velocity, and the gravity acceleration parameter g. As in Exercise 13-14, the ball is to rebound from the ground plane, but now the rebound path is the inverse of the parabolic descent shape, and the maximum rebound height is less than the original ball height because of ground friction. The inverse parabolic rebound path can be obtained as the reflection of the downward path about a vertical axis. One method for displaying the rebound motion is to plot the inverse parabolic path using a maximum height above the ground plane that is three-fourths that of the initial ball height. Thus, each time the ball bounces up, its rebound height decreases, and the motion can be terminated when the rebound height is below a selected minimum value.

13-17. A predefined scene and motion specification could be used for this program, or the object shapes, positions, and motion parameters could all be specified as input. Also, a simple two-dimensional scene could be specified, or the objects and motions could be three-dimensional. For example, a satellite orbiting animation could be displayed in two dimensions, with a circular moon revolving about a circular earth.

13-18. For this exercise, the rotating hexagon program in Section 13-10 can be modified by first displaying a menu of shapes, such as a triangle, rectangle, hexagon, and ellipse. Keyboard or mouse input (Section 11-6) can be used to select the object to be rotated.

13-19. Use an ellipse algorithm from Section 3-10 to calculate positions for the hexagon along the motion path. The hexagon program could also be modified to animate some other selected object, as in Exercise 13-18.

13-20. The speed of the hexagon could be controlled using preset increments and the up-down cursor keys on the keyboard, or the speed could modified by selecting $\pm$ speed increments from a displayed menu.