Animation

6.1 Idle Function

To perform animation (e.g., rotating the shapes), you could register an `idle()` callback handler with GLUT, via `glutIdleFunc` command. The graphic system will call back the `idle()` function when there is no other event to be processed.

```
void glutIdleFunc(void (*func)(void))
```

In the `idle()` function, you could issue `glutPostRedisplay` command to post a window re-paint request, which in turn will activate `display()` function.

```
void idle() {
   glutPostRedisplay();   // Post a re-paint request to activate display()
}
```

Take note that the above is equivalent to registering `display()` as the `idle` function.

```
// main
glutIdleFunc(display);
```

6.2 Double Buffering

Double buffering uses two display buffers to smoothen animation. The next screen is prepared in a *back* buffer, while the current screen is held in a *front* buffer. Once the preparation is done, you can use `glutSwapBuffer` command to swap the front and back buffers.

To use double buffering, you need to make two changes:

1. In the `main()`, include this line before creating the window:

   ```
   glutInitDisplayMode(GLUT_DOUBLE);  // Set double buffered mode
   ```

2. In the `display()` function, replace `glFlush()` with `glutSwapBuffers()`, which swap the front and back buffers.

Double buffering should be used in animation. For static display, single buffering is sufficient. (Many graphics hardware always double buffered, so it is hard to see the differences.)

6.3 Example 5: Animation using Idle Function (GL05IdleFunc.cpp)

The following program rotates all the shapes created in our previous example using idle function with double buffering.

```
1/*
2 * GL05IdleFunc.cpp: Translation and Rotation
3 * Transform primitives from their model spaces to world space (Model Transform).
4 */
5#include <windows.h>  // for MS Windows
6#include <GL/glut.h>  // GLUT, include glu.h and gl.h
7
8// Global variable
```

```
GLfloat angle = 0.0f;  // Current rotational angle of the shapes

/* Initialize OpenGL Graphics */
void initGL() {
   // Set "clearing" or background color
   glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Black and opaque
}

/* Called back when there is no other event to be handled */
void idle() {
   glutPostRedisplay();   // Post a re-paint request to activate display()
}

/* Handler for window-repaint event. Call back when the window first appears and
   whenever the window needs to be re-painted. */
void display() {
   glClear(GL_COLOR_BUFFER_BIT);   // Clear the color buffer
   glMatrixMode(GL_MODELVIEW);     // To operate on Model-View matrix
   glLoadIdentity();               // Reset the model-view matrix

   glPushMatrix();                     // Save model-view matrix setting
   glTranslatef(-0.5f, 0.4f, 0.0f);    // Translate
   glRotatef(angle, 0.0f, 0.0f, 1.0f); // rotate by angle in degrees
   glBegin(GL_QUADS);                  // Each set of 4 vertices form a quad
      glColor3f(1.0f, 0.0f, 0.0f);     // Red
      glVertex2f(-0.3f, -0.3f);
      glVertex2f( 0.3f, -0.3f);
      glVertex2f( 0.3f,  0.3f);
      glVertex2f(-0.3f,  0.3f);
   glEnd();
   glPopMatrix();                      // Restore the model-view matrix

   glPushMatrix();                     // Save model-view matrix setting
   glTranslatef(-0.4f, -0.3f, 0.0f);   // Translate
   glRotatef(angle, 0.0f, 0.0f, 1.0f); // rotate by angle in degrees
   glBegin(GL_QUADS);
      glColor3f(0.0f, 1.0f, 0.0f); // Green
      glVertex2f(-0.3f, -0.3f);
      glVertex2f( 0.3f, -0.3f);
      glVertex2f( 0.3f,  0.3f);
      glVertex2f(-0.3f,  0.3f);
   glEnd();
   glPopMatrix();                      // Restore the model-view matrix

   glPushMatrix();                     // Save model-view matrix setting
```

```
54   glTranslatef(-0.7f, -0.5f, 0.0f);   // Translate
55   glRotatef(angle, 0.0f, 0.0f, 1.0f); // rotate by angle in degrees
56   glBegin(GL_QUADS);
57      glColor3f(0.2f, 0.2f, 0.2f); // Dark Gray
58      glVertex2f(-0.2f, -0.2f);
59      glColor3f(1.0f, 1.0f, 1.0f); // White
60      glVertex2f( 0.2f, -0.2f);
61      glColor3f(0.2f, 0.2f, 0.2f); // Dark Gray
62      glVertex2f( 0.2f,  0.2f);
63      glColor3f(1.0f, 1.0f, 1.0f); // White
64      glVertex2f(-0.2f,  0.2f);
65   glEnd();
66   glPopMatrix();                      // Restore the model-view matrix
67
68   glPushMatrix();                     // Save model-view matrix setting
69   glTranslatef(0.4f, -0.3f, 0.0f);    // Translate
70   glRotatef(angle, 0.0f, 0.0f, 1.0f); // rotate by angle in degrees
71   glBegin(GL_TRIANGLES);
72      glColor3f(0.0f, 0.0f, 1.0f); // Blue
73      glVertex2f(-0.3f, -0.2f);
74      glVertex2f( 0.3f, -0.2f);
75      glVertex2f( 0.0f,  0.3f);
76   glEnd();
77   glPopMatrix();                      // Restore the model-view matrix
78
79   glPushMatrix();                     // Save model-view matrix setting
80   glTranslatef(0.6f, -0.6f, 0.0f);    // Translate
81   glRotatef(180.0f + angle, 0.0f, 0.0f, 1.0f); // Rotate 180+angle degree
82   glBegin(GL_TRIANGLES);
83      glColor3f(1.0f, 0.0f, 0.0f); // Red
84      glVertex2f(-0.3f, -0.2f);
85      glColor3f(0.0f, 1.0f, 0.0f); // Green
86      glVertex2f( 0.3f, -0.2f);
87      glColor3f(0.0f, 0.0f, 1.0f); // Blue
88      glVertex2f( 0.0f,  0.3f);
89   glEnd();
90   glPopMatrix();                      // Restore the model-view matrix
91
92   glPushMatrix();                     // Save model-view matrix setting
93   glTranslatef(0.5f, 0.4f, 0.0f);     // Translate
94   glRotatef(angle, 0.0f, 0.0f, 1.0f); // rotate by angle in degrees
95   glBegin(GL_POLYGON);
96      glColor3f(1.0f, 1.0f, 0.0f); // Yellow
97      glVertex2f(-0.1f, -0.2f);
98      glVertex2f( 0.1f, -0.2f);
```

```
 99      glVertex2f( 0.2f,  0.0f);
100      glVertex2f( 0.1f,  0.2f);
101      glVertex2f(-0.1f,  0.2f);
102      glVertex2f(-0.2f,  0.0f);
103   glEnd();
104   glPopMatrix();                    // Restore the model-view matrix
105
106   glutSwapBuffers();   // Double buffered - swap the front and back buffers
107
108   // Change the rotational angle after each display()
109   angle += 0.2f;
110}
111
112/* Handler for window re-size event. Called back when the window first appears and
113   whenever the window is re-sized with its new width and height */
114void reshape(GLsizei width, GLsizei height) {  // GLsizei for non-negative integer
115   // Compute aspect ratio of the new window
116   if (height == 0) height = 1;                // To prevent divide by 0
117   GLfloat aspect = (GLfloat)width / (GLfloat)height;
118
119   // Set the viewport to cover the new window
120   glViewport(0, 0, width, height);
121
122   // Set the aspect ratio of the clipping area to match the viewport
123   glMatrixMode(GL_PROJECTION);  // To operate on the Projection matrix
124   glLoadIdentity();
125   if (width >= height) {
126     // aspect >= 1, set the height from -1 to 1, with larger width
127      gluOrtho2D(-1.0 * aspect, 1.0 * aspect, -1.0, 1.0);
128   } else {
129      // aspect < 1, set the width to -1 to 1, with larger height
130     gluOrtho2D(-1.0, 1.0, -1.0 / aspect, 1.0 / aspect);
131   }
132}
133
134/* Main function: GLUT runs as a console application starting at main() */
135int main(int argc, char** argv) {
136   glutInit(&argc, argv);          // Initialize GLUT
137   glutInitDisplayMode(GLUT_DOUBLE);  // Enable double buffered mode
138   glutInitWindowSize(640, 480);   // Set the window's initial width & height - non-squa
139   glutInitWindowPosition(50, 50); // Position the window's initial top-left corner
140   glutCreateWindow("Animation via Idle Function");  // Create window with the given tit
141   glutDisplayFunc(display);       // Register callback handler for window re-paint event
142   glutReshapeFunc(reshape);       // Register callback handler for window re-size event
143   glutIdleFunc(idle);             // Register callback handler if no other event
```

```
144   initGL();                        // Our own OpenGL initialization
145   glutMainLoop();                  // Enter the infinite event-processing loop
146   return 0;
147}
```

In the above example, instead of accumulating all the translations and undoing the rotations, we use `glPushMatrix` to save the current state, perform transformations, and restore the saved state via `glPopMatrix`. (In the above example, we can also use `glLoadIdentity` to reset the matrix before the next transformations.)

```
GLfloat angle = 0.0f;  // Current rotational angle of the shapes
```

We define a global variable called `angle` to keep track of the rotational angle of all the shapes. We will later use `glRotatef` to rotate all the shapes to this angle.

```
angle += 0.2f;
```

At the end of each refresh (in `display()`), we update the rotational angle of all the shapes.

```
glutSwapBuffers();                  // Swap front- and back framebuffer

glutInitDisplayMode(GLUT_DOUBLE);  // In main(), enable double buffered mode
```

Instead of `glFlush()` which flushes the framebuffer for display immediately, we enable double buffering and use `glutSwapBuffer()` to swap the front- and back-buffer during the VSync for smoother display.

```
void idle() {
   glutPostRedisplay();   // Post a re-paint request to activate display()
}

glutIdleFunc(idle);       // In main() - Register callback handler if no other event
```

We define an `idle()` function, which posts a re-paint request and invoke `display()`, if there is no event outstanding. We register this `idle()` function in `main()` via `glutIdleFunc()`.

6.4 Double Buffering & Refresh Rate

When double buffering is enabled, `glutSwapBuffers` synchronizes with the screen refresh interval (VSync). That is, the buffers will be swapped at the same time when the monitor is putting up a new frame. As the result, `idle()` function, at best, refreshes the animation at the same rate as the refresh rate of the monitor (60Hz for LCD/LED monitor). It may operates at half the monitor refresh rate (if the computations takes more than 1 refresh interval), one-third, one-fourth, and so on, because it need to wait for the VSync.

6.5 Timer Function

With `idle()`, we have no control to the refresh interval. We could register a `Timer()` function with GLUT via `glutTimerFunc`. The `Timer()` function will be called back at the specified fixed interval.

```
void glutTimerFunc(unsigned int millis, void (*func)(int value), value)
   // where millis is the delay in milliseconds, value will be passed to the timer function.
```

6.6 Example 6: Animation via Timer Function (GL06TimerFunc.cpp)

The following modifications rotate all the shapes created in the earlier example counter-clockwise by 2 degree per 30 milliseconds.

```
 1/*
 2 * GL06TimerFunc.cpp: Translation and Rotation
 3 * Transform primitives from their model spaces to world space (Model Transform).
 4 */
 5#include <windows.h>  // for MS Windows
 6#include <GL/glut.h>  // GLUT, include glu.h and gl.h
 7
 8// global variable
 9GLfloat angle = 0.0f;  // rotational angle of the shapes
10int refreshMills = 30; // refresh interval in milliseconds
11
12/* Initialize OpenGL Graphics */
13void initGL() {
14   // Set "clearing" or background color
15   glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Black and opaque
16}
17
18/* Called back when timer expired */
19void Timer(int value) {
20   glutPostRedisplay();      // Post re-paint request to activate display()
21   glutTimerFunc(refreshMills, Timer, 0); // next Timer call milliseconds later
22}
23
24/* Handler for window-repaint event. Call back when the window first appears and
25   whenever the window needs to be re-painted. */
26void display() {
27   glClear(GL_COLOR_BUFFER_BIT);   // Clear the color buffer
28   glMatrixMode(GL_MODELVIEW);     // To operate on Model-View matrix
29   glLoadIdentity();               // Reset the model-view matrix
30
31   glPushMatrix();                    // Save model-view matrix setting
32   glTranslatef(-0.5f, 0.4f, 0.0f);    // Translate
33   glRotatef(angle, 0.0f, 0.0f, 1.0f); // rotate by angle in degrees
34   glBegin(GL_QUADS);                  // Each set of 4 vertices form a quad
35      glColor3f(1.0f, 0.0f, 0.0f);     // Red
36      glVertex2f(-0.3f, -0.3f);
37      glVertex2f( 0.3f, -0.3f);
38      glVertex2f( 0.3f,  0.3f);
39      glVertex2f(-0.3f,  0.3f);
```

```
   glEnd();
   glPopMatrix();                      // Restore the model-view matrix

   glPushMatrix();                     // Save model-view matrix setting
   glTranslatef(-0.4f, -0.3f, 0.0f);   // Translate
   glRotatef(angle, 0.0f, 0.0f, 1.0f); // rotate by angle in degrees
   glBegin(GL_QUADS);
      glColor3f(0.0f, 1.0f, 0.0f); // Green
      glVertex2f(-0.3f, -0.3f);
      glVertex2f( 0.3f, -0.3f);
      glVertex2f( 0.3f,  0.3f);
      glVertex2f(-0.3f,  0.3f);
   glEnd();
   glPopMatrix();                      // Restore the model-view matrix

   glPushMatrix();                     // Save model-view matrix setting
   glTranslatef(-0.7f, -0.5f, 0.0f);   // Translate
   glRotatef(angle, 0.0f, 0.0f, 1.0f); // rotate by angle in degrees
   glBegin(GL_QUADS);
      glColor3f(0.2f, 0.2f, 0.2f); // Dark Gray
      glVertex2f(-0.2f, -0.2f);
      glColor3f(1.0f, 1.0f, 1.0f); // White
      glVertex2f( 0.2f, -0.2f);
      glColor3f(0.2f, 0.2f, 0.2f); // Dark Gray
      glVertex2f( 0.2f,  0.2f);
      glColor3f(1.0f, 1.0f, 1.0f); // White
      glVertex2f(-0.2f,  0.2f);
   glEnd();
   glPopMatrix();                      // Restore the model-view matrix

   glPushMatrix();                     // Save model-view matrix setting
   glTranslatef(0.4f, -0.3f, 0.0f);    // Translate
   glRotatef(angle, 0.0f, 0.0f, 1.0f); // rotate by angle in degrees
   glBegin(GL_TRIANGLES);
      glColor3f(0.0f, 0.0f, 1.0f); // Blue
      glVertex2f(-0.3f, -0.2f);
      glVertex2f( 0.3f, -0.2f);
      glVertex2f( 0.0f,  0.3f);
   glEnd();
   glPopMatrix();                      // Restore the model-view matrix

   glPushMatrix();                     // Save model-view matrix setting
   glTranslatef(0.6f, -0.6f, 0.0f);    // Translate
   glRotatef(180.0f + angle, 0.0f, 0.0f, 1.0f); // Rotate 180+angle degree
   glBegin(GL_TRIANGLES);
```

```
 85      glColor3f(1.0f, 0.0f, 0.0f); // Red
 86      glVertex2f(-0.3f, -0.2f);
 87      glColor3f(0.0f, 1.0f, 0.0f); // Green
 88      glVertex2f( 0.3f, -0.2f);
 89      glColor3f(0.0f, 0.0f, 1.0f); // Blue
 90      glVertex2f( 0.0f,  0.3f);
 91   glEnd();
 92   glPopMatrix();                     // Restore the model-view matrix
 93
 94   glPushMatrix();                    // Save model-view matrix setting
 95   glTranslatef(0.5f, 0.4f, 0.0f);    // Translate
 96   glRotatef(angle, 0.0f, 0.0f, 1.0f); // rotate by angle in degrees
 97   glBegin(GL_POLYGON);
 98      glColor3f(1.0f, 1.0f, 0.0f); // Yellow
 99      glVertex2f(-0.1f, -0.2f);
100      glVertex2f( 0.1f, -0.2f);
101      glVertex2f( 0.2f,  0.0f);
102      glVertex2f( 0.1f,  0.2f);
103      glVertex2f(-0.1f,  0.2f);
104      glVertex2f(-0.2f,  0.0f);
105   glEnd();
106   glPopMatrix();                     // Restore the model-view matrix
107
108   glutSwapBuffers();   // Double buffered - swap the front and back buffers
109
110   // Change the rotational angle after each display()
111   angle += 2.0f;
112}
113
114/* Handler for window re-size event. Called back when the window first appears and
115   whenever the window is re-sized with its new width and height */
116void reshape(GLsizei width, GLsizei height) {  // GLsizei for non-negative integer
117   // Compute aspect ratio of the new window
118   if (height == 0) height = 1;                // To prevent divide by 0
119   GLfloat aspect = (GLfloat)width / (GLfloat)height;
120
121   // Set the viewport to cover the new window
122   glViewport(0, 0, width, height);
123
124   // Set the aspect ratio of the clipping area to match the viewport
125   glMatrixMode(GL_PROJECTION);  // To operate on the Projection matrix
126   glLoadIdentity();
127   if (width >= height) {
128     // aspect >= 1, set the height from -1 to 1, with larger width
129      gluOrtho2D(-1.0 * aspect, 1.0 * aspect, -1.0, 1.0);
```

```
130   } else {
131      // aspect < 1, set the width to -1 to 1, with larger height
132     gluOrtho2D(-1.0, 1.0, -1.0 / aspect, 1.0 / aspect);
133   }
134}
135
136/* Main function: GLUT runs as a console application starting at main() */
137int main(int argc, char** argv) {
138   glutInit(&argc, argv);          // Initialize GLUT
139   glutInitDisplayMode(GLUT_DOUBLE);  // Enable double buffered mode
140   glutInitWindowSize(640, 480);   // Set the window's initial width & height - non-squa
141   glutInitWindowPosition(50, 50); // Position the window's initial top-left corner
142   glutCreateWindow("Animation via Idle Function");  // Create window with the given tit
143   glutDisplayFunc(display);       // Register callback handler for window re-paint even
144   glutReshapeFunc(reshape);       // Register callback handler for window re-size event
145   glutTimerFunc(0, Timer, 0);     // First timer call immediately
146   initGL();                       // Our own OpenGL initialization
147   glutMainLoop();                 // Enter the infinite event-processing loop
148   return 0;
149}
```

```
void Timer(int value) {
   glutPostRedisplay();                 // Post re-paint request to activate display()
   glutTimerFunc(refreshMills, Timer, 0); // next Timer call milliseconds later
}
```

We replace the `idle()` function by a `timer()` function, which post a re-paint request to invoke `display()`, after the timer expired.

```
glutTimerFunc(0, Timer, 0);     // First timer call immediately
```

In `main()`, we register the `timer()` function, and activate the `timer()` immediately (with initial timer = 0).

6.7 More GLUT functions

- `glutInitDisplayMode`: requests a display with the specified mode, such as color mode (`GLUT_RGB`, `GLUT_RGBA`, `GLUT_INDEX`), single/double buffering (`GLUT_SINGLE`, `GLUT_DOUBLE`), enable depth (`GLUT_DEPTH`), joined with a bit `OR` '|'.

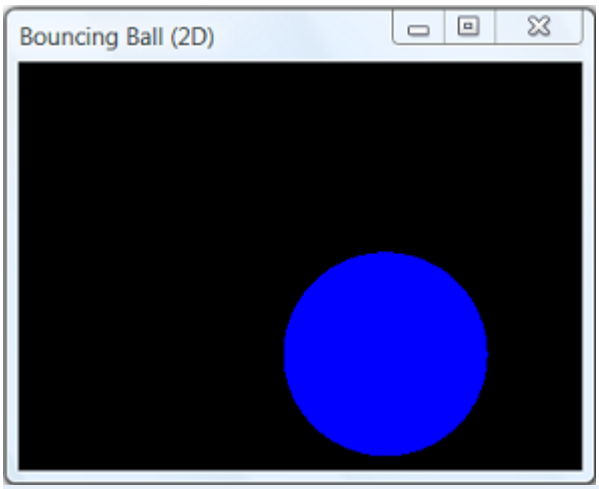
```
void glutInitDisplayMode(unsigned int displayMode)
```

For example,

```
glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
   // Use RGBA color, enable double buffering and enable depth buffer
```

6.8 Example 7: A Bouncing Ball (`GL07BouncingBall.cpp`)

This example shows a ball bouncing inside the window. Take note that circle is not a primitive geometric shape in OpenGL. This example uses `TRIANGLE_FAN` to compose a circle.




```
 1/*
 2 * GL07BouncingBall.cpp: A ball bouncing inside the window
 3 */
 4#include <windows.h>  // for MS Windows
 5#include <GL/glut.h>  // GLUT, includes glu.h and gl.h
 6#include <Math.h>     // Needed for sin, cos
 7#define PI 3.14159265f
 8
 9// Global variables
10char title[] = "Bouncing Ball (2D)";  // Windowed mode's title
11int windowWidth  = 640;     // Windowed mode's width
12int windowHeight = 480;     // Windowed mode's height
13int windowPosX   = 50;      // Windowed mode's top-left corner x
14int windowPosY   = 50;      // Windowed mode's top-left corner y
15
16GLfloat ballRadius = 0.5f;   // Radius of the bouncing ball
17GLfloat ballX = 0.0f;         // Ball's center (x, y) position
18GLfloat ballY = 0.0f;
19GLfloat ballXMax, ballXMin, ballYMax, ballYMin; // Ball's center (x, y) bounds
20GLfloat xSpeed = 0.02f;      // Ball's speed in x and y directions
21GLfloat ySpeed = 0.007f;
22int refreshMillis = 30;      // Refresh period in milliseconds
23
24// Projection clipping area
25GLdouble clipAreaXLeft, clipAreaXRight, clipAreaYBottom, clipAreaYTop;
26
27/* Initialize OpenGL Graphics */
28void initGL() {
29   glClearColor(0.0, 0.0, 0.0, 1.0); // Set background (clear) color to black
30}
31
32/* Callback handler for window re-paint event */
```

```
33void display() {
34   glClear(GL_COLOR_BUFFER_BIT);  // Clear the color buffer
35   glMatrixMode(GL_MODELVIEW);    // To operate on the model-view matrix
36   glLoadIdentity();              // Reset model-view matrix
37
38   glTranslatef(ballX, ballY, 0.0f);  // Translate to (xPos, yPos)
39   // Use triangular segments to form a circle
40   glBegin(GL_TRIANGLE_FAN);
41      glColor3f(0.0f, 0.0f, 1.0f);  // Blue
42      glVertex2f(0.0f, 0.0f);       // Center of circle
43      int numSegments = 100;
44      GLfloat angle;
45      for (int i = 0; i <= numSegments; i++) { // Last vertex same as first vertex
46         angle = i * 2.0f * PI / numSegments;  // 360 deg for all segments
47         glVertex2f(cos(angle) * ballRadius, sin(angle) * ballRadius);
48      }
49   glEnd();
50
51   glutSwapBuffers();  // Swap front and back buffers (of double buffered mode)
52
53   // Animation Control - compute the location for the next refresh
54   ballX += xSpeed;
55   ballY += ySpeed;
56   // Check if the ball exceeds the edges
57   if (ballX > ballXMax) {
58      ballX = ballXMax;
59      xSpeed = -xSpeed;
60   } else if (ballX < ballXMin) {
61      ballX = ballXMin;
62      xSpeed = -xSpeed;
63   }
64   if (ballY > ballYMax) {
65      ballY = ballYMax;
66      ySpeed = -ySpeed;
67   } else if (ballY < ballYMin) {
68      ballY = ballYMin;
69      ySpeed = -ySpeed;
70   }
71}
72
73/* Call back when the windows is re-sized */
74void reshape(GLsizei width, GLsizei height) {
75   // Compute aspect ratio of the new window
76   if (height == 0) height = 1;                // To prevent divide by 0
77   GLfloat aspect = (GLfloat)width / (GLfloat)height;
```

```
 78
 79   // Set the viewport to cover the new window
 80   glViewport(0, 0, width, height);
 81
 82   // Set the aspect ratio of the clipping area to match the viewport
 83   glMatrixMode(GL_PROJECTION);  // To operate on the Projection matrix
 84   glLoadIdentity();             // Reset the projection matrix
 85   if (width >= height) {
 86      clipAreaXLeft   = -1.0 * aspect;
 87      clipAreaXRight  = 1.0 * aspect;
 88      clipAreaYBottom = -1.0;
 89      clipAreaYTop    = 1.0;
 90   } else {
 91      clipAreaXLeft   = -1.0;
 92      clipAreaXRight  = 1.0;
 93      clipAreaYBottom = -1.0 / aspect;
 94      clipAreaYTop    = 1.0 / aspect;
 95   }
 96   gluOrtho2D(clipAreaXLeft, clipAreaXRight, clipAreaYBottom, clipAreaYTop);
 97   ballXMin = clipAreaXLeft + ballRadius;
 98   ballXMax = clipAreaXRight - ballRadius;
 99   ballYMin = clipAreaYBottom + ballRadius;
100   ballYMax = clipAreaYTop - ballRadius;
101}
102
103/* Called back when the timer expired */
104void Timer(int value) {
105   glutPostRedisplay();    // Post a paint request to activate display()
106   glutTimerFunc(refreshMillis, Timer, 0); // subsequent timer call at milliseconds
107}
108
109/* Main function: GLUT runs as a console application starting at main() */
110int main(int argc, char** argv) {
111   glutInit(&argc, argv);            // Initialize GLUT
112   glutInitDisplayMode(GLUT_DOUBLE); // Enable double buffered mode
113   glutInitWindowSize(windowWidth, windowHeight);  // Initial window width and height
114   glutInitWindowPosition(windowPosX, windowPosY); // Initial window top-left corner (x,
115   glutCreateWindow(title);      // Create window with given title
116   glutDisplayFunc(display);     // Register callback handler for window re-paint
117   glutReshapeFunc(reshape);     // Register callback handler for window re-shape
118   glutTimerFunc(0, Timer, 0);   // First timer call immediately
119   initGL();                     // Our own OpenGL initialization
120   glutMainLoop();               // Enter event-processing loop
121   return 0;
122}
```