OpenGL Tutorial

An Introduction on OpenGL with 2D Graphics

TABLE OF CONTENTS (HIDE)

1. Setting Up OpenGL

To set up OpenGL, depending on your programming platform, read:

- How to write OpenGL programs in C/C++.
- How to write OpenGL programs in Java: JOGL or LWJGL.
- How to write OpenGL|ES programs in Android.

1.1 Example 1: Setting Up OpenGL and GLUT (`GL01Hello.cpp`)

Make sure that you can run the "`GL01Hello.cpp`" described in "How to write OpenGL programs in C/C++", reproduced below:

```
/*
 * GL01Hello.cpp: Test OpenGL/GLUT C/C++ Setup
 * Tested under Eclipse CDT with MinGW/Cygwin and CodeBlocks with MinGW
 * To compile with -lfreeglut -lglu32 -lopengl32
 */
#include <windows.h>  // for MS Windows
#include <GL/glut.h>  // GLUT, include glu.h and gl.h

/* Handler for window-repaint event. Call back when the window first appears and
   whenever the window needs to be re-painted. */
void display() {
   glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set background color to black and opaque
   glClear(GL_COLOR_BUFFER_BIT);         // Clear the color buffer (background)

   // Draw a Red 1x1 Square centered at origin
   glBegin(GL_QUADS);              // Each set of 4 vertices form a quad
      glColor3f(1.0f, 0.0f, 0.0f); // Red
      glVertex2f(-0.5f, -0.5f);    // x, y
      glVertex2f( 0.5f, -0.5f);
      glVertex2f( 0.5f,  0.5f);
      glVertex2f(-0.5f,  0.5f);
   glEnd();

   glFlush();  // Render now
}

```

```
/* Main function: GLUT runs as a console application starting at main()  */
int main(int argc, char** argv) {
   glutInit(&argc, argv);                 // Initialize GLUT
   glutCreateWindow("OpenGL Setup Test"); // Create a window with the given title
   glutInitWindowSize(320, 320);   // Set the window's initial width & height
   glutInitWindowPosition(50, 50); // Position the window's initial top-left corner
   glutDisplayFunc(display); // Register display callback handler for window re-paint
   glutMainLoop();           // Enter the event-processing loop
   return 0;
}
```

```
#include <windows.h>
```

The header "`windows.h`" is needed for the Windows platform only.

```
#include <GL/glut.h>
```

We also included the GLUT header, which is guaranteed to include "`glu.h`" (for GL Utility) and "`gl.h`" (for Core OpenGL).

The rest of the program will be explained in due course.

2. Introduction

OpenGL (Open Graphics Library) is a cross-platform, hardware-accelerated, language-independent, industrial standard API for producing 3D (including 2D) graphics. Modern computers have dedicated GPU (Graphics Processing Unit) with its own memory to speed up graphics rendering. OpenGL is the software interface to graphics hardware. In other words, OpenGL graphic rendering commands issued by your applications could be directed to the graphic hardware and accelerated.

We use 3 sets of libraries in our OpenGL programs:

1. **Core OpenGL (GL)**: consists of hundreds of commands, which begin with a prefix "`gl`" (e.g., `glColor`, `glVertex`, `glTranslate`, `glRotate`). The Core OpenGL models an object via a set of geometric primitives such as point, line and polygon.
2. **OpenGL Utility Library (GLU)**: built on-top of the core OpenGL to provide important utilities (such as setting camera view and projection) and more building models (such as qradric surfaces and polygon tessellation). GLU commands start with a prefix "`glu`" (e.g., `gluLookAt`, `gluPerspective`).

3. **OpenGL Utilities Toolkit (GLUT)**: OpenGL is designed to be independent of the windowing system or operating system. GLUT is needed to interact with the Operating System (such as creating a window, handling key and mouse inputs); it also provides more building models (such as sphere and torus). GLUT commands start with a prefix of "`glut`" (e.g., `glutCreatewindow`, `glutMouseFunc`). GLUT is platform independent, which is built on top of platform-specific OpenGL extension such as GLX for X Window System, WGL for Microsoft Window, and AGL, CGL or Cocoa for Mac OS.
   Quoting from the opengl.org: "GLUT is designed for constructing small to medium sized OpenGL programs. While GLUT is well-suited to learning OpenGL and developing simple OpenGL applications, GLUT is not a full-featured toolkit so large applications requiring sophisticated user interfaces are better off using native window system toolkits. *GLUT is simple, easy, and small.*"
   Alternative of GLUT includes SDL, ...
4. **OpenGL Extension Wrangler Library (GLEW)**: "GLEW is a cross-platform open-source C/C++ extension loading library. GLEW provides efficient run-time mechanisms for determining which OpenGL extensions are supported on the target platform." Source and pre-build binary available at http://glew.sourceforge.net/. A standalone utility called "`glewinfo.exe`" (under the "`bin`" directory) can be used to produce the list of OpenGL functions supported by your graphics system.
5. Others.

3. Vertex, Primitive and Color

3.1 Example 2: Vertex, Primitive and Color (`GL02Primitive.cpp`)

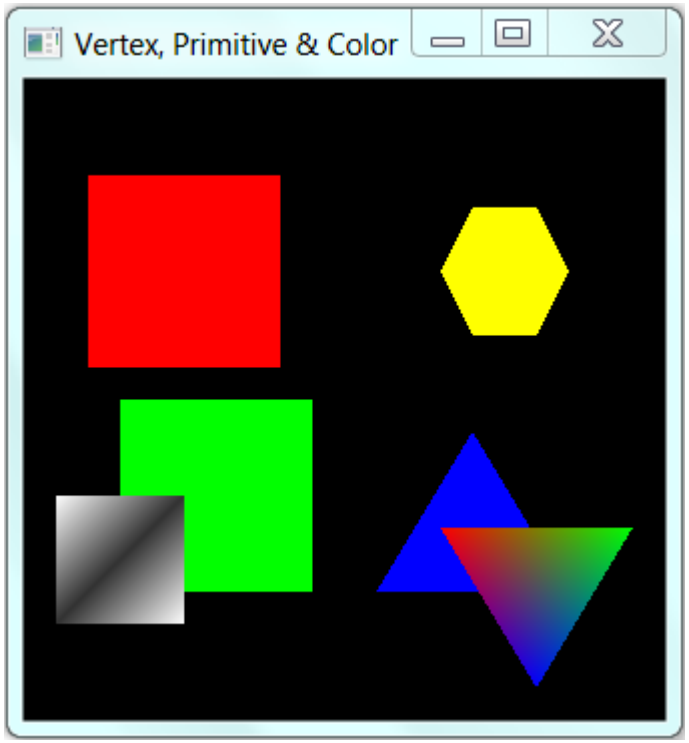
Try building and runnng this OpenGL C/C++ program:

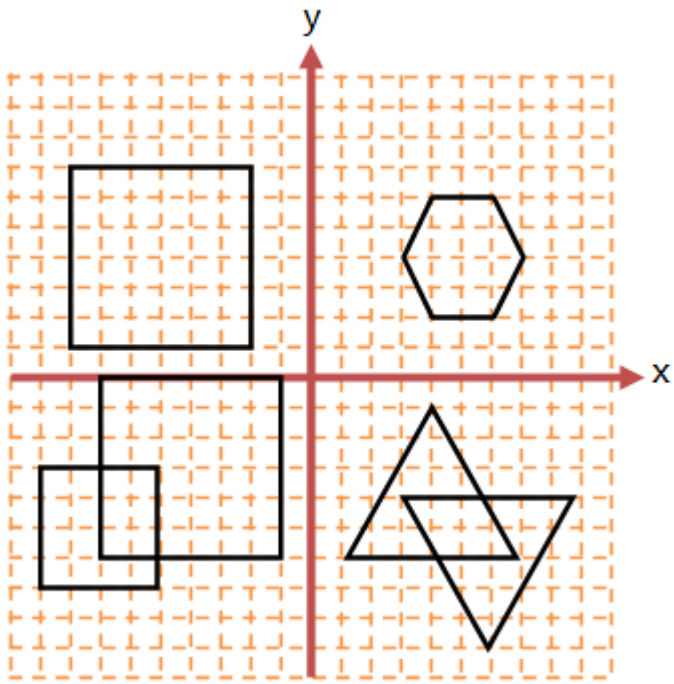
```
/*
 * GL02Primitive.cpp: Vertex, Primitive and Color
 * Draw Simple 2D colored Shapes: quad, triangle and polygon.
 */
#include <windows.h>  // for MS Windows
#include <GL/glut.h>  // GLUT, include glu.h and gl.h

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

```
12  }
13
14  /* Handler for window-repaint event. Call back when the window first appears and
15     whenever the window needs to be re-painted. */
16  void display() {
17     glClear(GL_COLOR_BUFFER_BIT);   // Clear the color buffer with current clearing color
18
19     // Define shapes enclosed within a pair of glBegin and glEnd
20     glBegin(GL_QUADS);              // Each set of 4 vertices form a quad
21        glColor3f(1.0f, 0.0f, 0.0f); // Red
22        glVertex2f(-0.8f, 0.1f);     // Define vertices in counter-clockwise (CCW) order
23        glVertex2f(-0.2f, 0.1f);     //  so that the normal (front-face) is facing you
24        glVertex2f(-0.2f, 0.7f);
25        glVertex2f(-0.8f, 0.7f);
26
27        glColor3f(0.0f, 1.0f, 0.0f); // Green
28        glVertex2f(-0.7f, -0.6f);
29        glVertex2f(-0.1f, -0.6f);
30        glVertex2f(-0.1f,  0.0f);
31        glVertex2f(-0.7f,  0.0f);
32
33        glColor3f(0.2f, 0.2f, 0.2f); // Dark Gray
34        glVertex2f(-0.9f, -0.7f);
35        glColor3f(1.0f, 1.0f, 1.0f); // White
36        glVertex2f(-0.5f, -0.7f);
37        glColor3f(0.2f, 0.2f, 0.2f); // Dark Gray
38        glVertex2f(-0.5f, -0.3f);
39        glColor3f(1.0f, 1.0f, 1.0f); // White
40        glVertex2f(-0.9f, -0.3f);
41     glEnd();
42
43     glBegin(GL_TRIANGLES);          // Each set of 3 vertices form a triangle
44        glColor3f(0.0f, 0.0f, 1.0f); // Blue
45        glVertex2f(0.1f, -0.6f);
46        glVertex2f(0.7f, -0.6f);
47        glVertex2f(0.4f, -0.1f);
```

```
48
49          glColor3f(1.0f, 0.0f, 0.0f); // Red
50          glVertex2f(0.3f, -0.4f);
51          glColor3f(0.0f, 1.0f, 0.0f); // Green
52          glVertex2f(0.9f, -0.4f);
53          glColor3f(0.0f, 0.0f, 1.0f); // Blue
54          glVertex2f(0.6f, -0.9f);
55       glEnd();
56
57       glBegin(GL_POLYGON);            // These vertices form a closed polygon
58          glColor3f(1.0f, 1.0f, 0.0f); // Yellow
59          glVertex2f(0.4f, 0.2f);
60          glVertex2f(0.6f, 0.2f);
61          glVertex2f(0.7f, 0.4f);
62          glVertex2f(0.6f, 0.6f);
63          glVertex2f(0.4f, 0.6f);
64          glVertex2f(0.3f, 0.4f);
65       glEnd();
66
67       glFlush();  // Render now
68    }
69
70    /* Main function: GLUT runs as a console application starting at main()  */
71    int main(int argc, char** argv) {
72       glutInit(&argc, argv);          // Initialize GLUT
73       glutCreateWindow("Vertex, Primitive & Color");  // Create window with the given title
74       glutInitWindowSize(320, 320);   // Set the window's initial width & height
75       glutInitWindowPosition(50, 50); // Position the window's initial top-left corner
76       glutDisplayFunc(display);       // Register callback handler for window re-paint event
77       initGL();                       // Our own OpenGL initialization
78       glutMainLoop();                 // Enter the event-processing loop
79       return 0;
80    }
```

The expected output and the coordinates are as follows. Take note that 4 shapes have pure color, and 2 shapes have color blending from their vertices.





I shall explain the program in the following sections.

3.2 OpenGL as a State Machine

OpenGL operates as a *state machine*, and maintain a set of *state variables* (such as the foreground color, background color, and many more). In a state machine, once the value of a state variable is set, the value persists until a new value is given.

For example, we set the "clearing" (background) color to black *once* in `initGL()`. We use this setting to clear the window in the `display()` *repeatedly* (`display()` is called back whenever there is a window re-paint request) - the clearing color is not changed in the entire program.

```
// In initGL(), set the "clearing" or background color
glClearColor(0.0f, 0.0f, 0.0f, 1.0f);  // black and opaque

// In display(), clear the color buffer (i.e., set background) with the current "clearing" color
glClear(GL_COLOR_BUFFER_BIT);
```

Another example: If we use `glColor` function to set the current foreground color to "red", then "red" will be used for all the subsequent vertices, until we use another `glColor` function to change the foreground color.

In a state machine, everything shall remain until you explicitly change it!

3.3 Naming Convention for OpenGL Functions

An OpenGL functions:

- begins with lowercase `gl` (for core OpenGL), `glu` (for OpenGL Utility) or `glut` (for OpenGL Utility Toolkit).
- followed by the purpose of the function, in *camel case* (initial-capitalized), e.g., `glColor` to specify the drawing color, `glVertex` to define the position of a vertex.
- followed by specifications for the parameters, e.g., `glColor3f` takes three `float` parameters. `glVectex2i` takes two `int` parameters.
  (This is needed as C Language does not support function overloading. Different versions of the function need to be written for different parameter lists.)

The convention can be expressed as follows:

```
returnType glFunction[234][sifd] (type value, ...);  // 2, 3 or 4 parameters
returnType glFunction[234][sifd]v (type *value);     // an array parameter
```

The function may take 2, 3, or 4 parameters, in type of `s` (`GLshort`), `i` (`GLint`), `f` (`GLfloat`) or `d` (`GLdouble`). The 'v' (for vector) denotes that the parameters are kept in an array of 2, 3, or 4 elements, and pass into the function as an array pointer.

OpenGL defines its own *data types*:

- Signed Integers: `GLbyte` (8-bit), `GLshort` (16-bit), `GLint` (32-bit).
- Unsigned Integers: `GLubyte` (8-bit), `GLushort` (16-bit), `GLuint` (32-bit).
- Floating-point numbers: `GLfloat` (32-bit), `GLdouble` (64-bit), `GLclampf` and `GLclampd` (between 0.0 and 1.0).
- `GLboolean` (unsigned char with 0 for false and non-0 for true).
- `GLsizei` (32-bit non-negative integers).
- `GLenum` (32-bit enumerated integers).

The OpenGL types are defined via `typedef` in "`gl.h`" as follows:

```
typedef unsigned int    GLenum;
typedef unsigned char   GLboolean;
typedef unsigned int    GLbitfield;
typedef void            GLvoid;
typedef signed char     GLbyte;         /* 1-byte signed */
typedef short           GLshort;        /* 2-byte signed */
typedef int             GLint;          /* 4-byte signed */
typedef unsigned char   GLubyte;        /* 1-byte unsigned */
typedef unsigned short  GLushort;       /* 2-byte unsigned */
typedef unsigned int    GLuint;         /* 4-byte unsigned */
typedef int             GLsizei;        /* 4-byte signed */
typedef float           GLfloat;        /* single precision float */
typedef float           GLclampf;       /* single precision float in [0,1] */
typedef double          GLdouble;       /* double precision float */
typedef double          GLclampd;       /* double precision float in [0,1] */
```

OpenGL's *constants* begins with "`GL_`", "`GLU_`" or "`GLUT_`", in uppercase separated with underscores, e.g., `GL_COLOR_BUFFER_BIT`.

For examples,

```
glVertex3f(1.1f, 2.2f, 3.3f);        // 3 GLfloat parameters
glVertex2i(4, 5);                    // 2 GLint paramaters
glColor4f(0.0f, 0.0f, 0.0f, 1.0f);   // 4 GLfloat parameters

GLdouble aVertex[] = {1.1, 2.2, 3.3};
glVertex3fv(aVertex);        // an array of 3 GLfloat values
```

3.4 One-time Initialization `initGL()`

The `initGL()` is meant for carrying out one-time OpenGL initialization tasks, such as setting the clearing color. `initGL()` is invoked once (and only once) in `main()`.

3.5 Callback Handler `display()`

The function `display()` is known as a *callback event handler*. An event handler provides the *response* to a particular *event* (such as key-press, mouse-click, window-paint). The function `display()` is meant to be the handler for *window-paint* event. The OpenGL graphics system calls back `display()` in

response to a window-paint request to re-paint the window (e.g., window first appears, window is restored after minimized, and window is resized). Callback means that the function is invoked by the system, instead of called by the your program.

The `Display()` runs when the window first appears and once per subsequent re-paint request. Observe that we included OpenGL graphics rendering code inside the `display()` function, so as to re-draw the entire window when the window first appears and upon each re-paint request.

3.6 Setting up GLUT - `main()`

GLUT provides high-level utilities to simplify OpenGL programming, especially in interacting with the Operating System (such as creating a window, handling key and mouse inputs). The following GLUT functions were used in the above program:

- `glutInit`: initializes GLUT, must be called before other GL/GLUT functions. It takes the same arguments as the `main()`.

```
void glutInit(int *argc, char **argv)
```

- `glutCreateWindow`: creates a window with the given title.

```
int glutCreateWindow(char *title)
```

- `glutInitWindowSize`: specifies the initial window width and height, in pixels.

```
void glutInitWindowSize(int width, int height)
```

- `glutInitWindowPosition`: positions the top-left corner of the initial window at (*x*, *y*). The coordinates (*x*, *y*), in term of pixels, is measured in window coordinates, i.e., origin (0, 0) is at the top-left corner of the screen; x-axis pointing right and y-axis pointing down.

```
void glutInitWindowPosition(int x, int y)
```

- `glutDisplayFunc`: registers the callback function (or event handler) for handling window-paint event. The OpenGL graphic system calls back this handler when it receives a window re-paint request. In the example, we register the function `display()` as the handler.

```
void glutDisplayFunc(void (*func)(void))
```

- `glutMainLoop`: enters the infinite event-processing loop, i.e, put the OpenGL graphics system to wait for events (such as re-paint), and trigger respective event handlers (such as `display()`).

```
void glutMainLoop()
```

In the `main()` function of the example:

```
glutInit(&argc, argv);
glutCreateWindow("Vertex, Primitive & Color");
glutInitWindowSize(320, 320);
glutInitWindowPosition(50, 50);
```

We initialize the GLUT and create a window with a title, an initial size and position.

```
glutDisplayFunc(display);
```

We register `display()` function as the callback handler for window-paint event. That is, `display()` runs when the window first appears and whenever there is a request to re-paint the window.

```
initGL();
```

We call the `initGL()` to perform all the one-time initialization operations. In this example, we set the clearing (background) color once, and use it repeatably in the `display()` function.

```
glutMainLoop();
```

We then put the program into the event-handling loop, awaiting for events (such as window-paint request) to trigger off the respective event handlers (such as `display()`).

3.7 Color

We use `glColor` function to set the *foreground color*, and `glClearColor` function to set the *background* (or *clearing*) color.

```
void glColor3f(GLfloat red, GLfloat green, GLfloat blue)
void glColor3fv(GLfloat *colorRGB)
void glColor4f(GLfloat red, GLfloat green, GLfloat blue, GLfloat alpha)
void glColor4fv(GLfloat *colorRGBA)

void glClearColor(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha)
   // GLclampf in the range of 0.0f to 1.0f
```

Notes:

- Color is typically specified in `float` in the range `0.0f` and `1.0f`.

- Color can be specified using RGB (Red-Green-Blue) or RGBA (Red-Green-Blue-Alpha) components. The 'A' (or alpha) specifies the transparency (or opacity) index, with value of 1 denotes opaque (non-transparent and cannot see-thru) and value of 0 denotes total transparent. We shall discuss alpha later.

In the above example, we set the background color via `glClearColor` in `initGL()`, with R=0, G=0, B=0 (black) and A=1 (opaque and cannot see through).

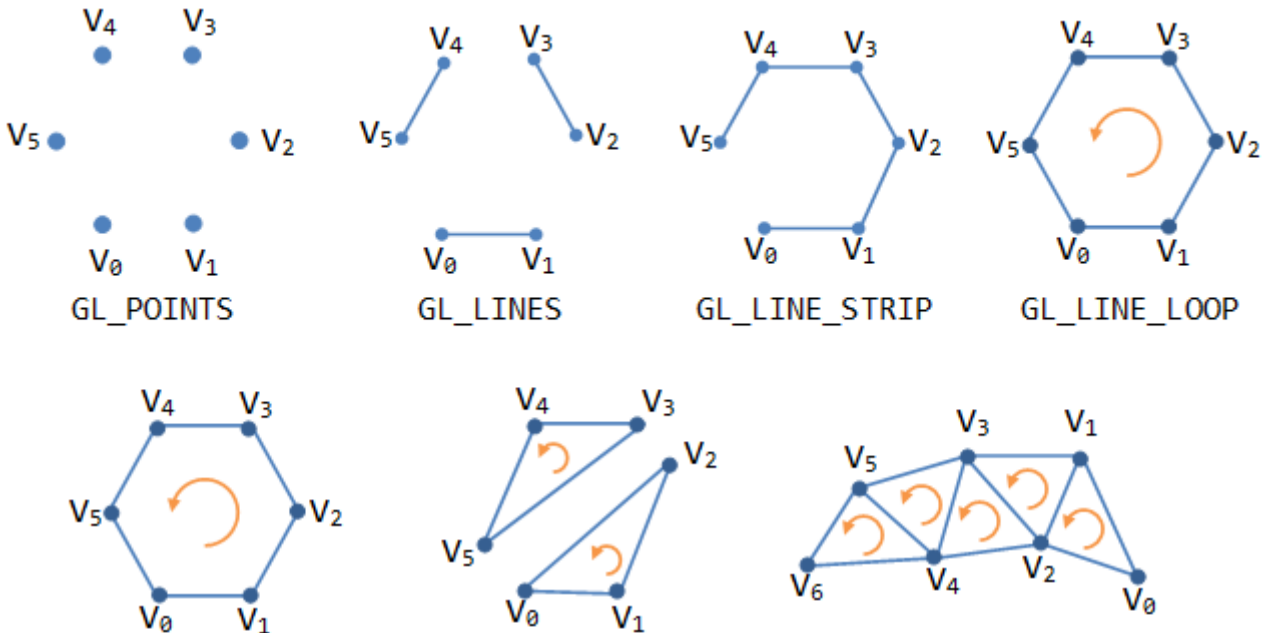
```
// In initGL(), set the "clearing" or background color
glClearColor(0.0f, 0.0f, 0.0f, 1.0f);  // Black and opague
```

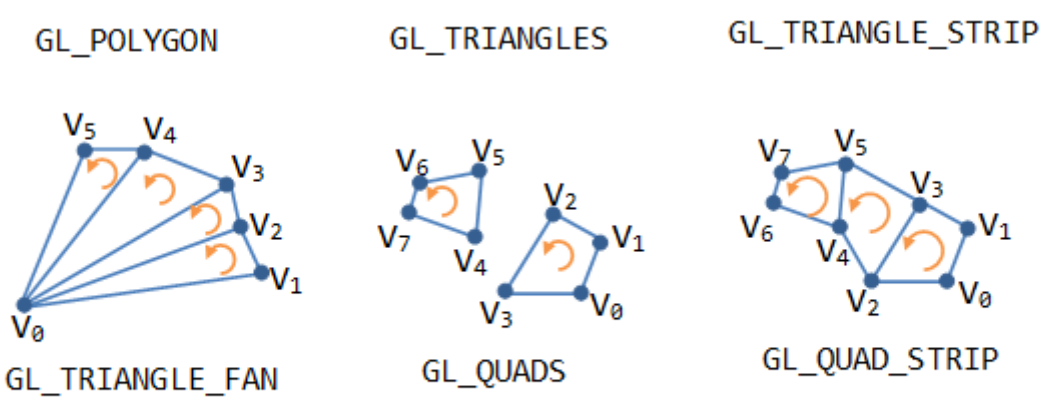
In `display()`, we set the vertex color via `glColor3f` for subsequent vertices. For example, R=1, G=0, B=0 (red).

```
// In display(), set the foreground color of the pixel
glColor3f(1.0f, 0.0f, 0.0f);  // Red
```

3.8 Geometric Primitives

In OpenGL, an object is made up of geometric primitives such as triangle, quad, line segment and point. A primitive is made up of one or more vertices. OpenGL supports the following primitives:





OpenGL Primitives

A geometric primitive is defined by specifying its vertices via `glVertex` function, enclosed within a pair `glBegin` and `glEnd`.

```
void glBegin(GLenum shape)
   void glVertex[234][sifd] (type x, type y, type z, ...)
   void glVertex[234][sifd]v (type *coords)
void glEnd()
```

`glBegin` specifies the type of geometric object, such as `GL_POINTS`, `GL_LINES`, `GL_QUADS`, `GL_TRIANGLES`, and `GL_POLYGON`. For types that end with 'S', you can define multiple objects of the same type in each `glBegin`/`glEnd` pair. For example, for `GL_TRIANGLES`, each set of three `glVertex`'s defines a triangle.

The vertices are usually specified in `float` precision. It is because integer is not suitable for trigonometric operations (needed to carry out transformations such as rotation). Precision of `float` is sufficient for carrying out intermediate operations, and render the objects finally into pixels on screen (with resolution of says 800x600, integral precision). `double` precision is often not necessary.

In the above example:

```
glBegin(GL_QUADS);
   .... 4 quads with 12x glVertex() ....
glEnd();
```

we define 3 color quads (`GL_QUADS`) with 12x `glVertex()` functions.

```
glColor3f(1.0f, 0.0f, 0.0f);
```

```
glVertex2f(-0.8f, 0.1f);
glVertex2f(-0.2f, 0.1f);
glVertex2f(-0.2f, 0.7f);
glVertex2f(-0.8f, 0.7f);
```

We set the color to red (R=1, G=0, B=0). All subsequent vertices will have the color of red. Take note that in OpenGL, color (and many properties) is applied to vertices rather than primitive shapes. The color of the a primitive shape is *interpolated* from its vertices.

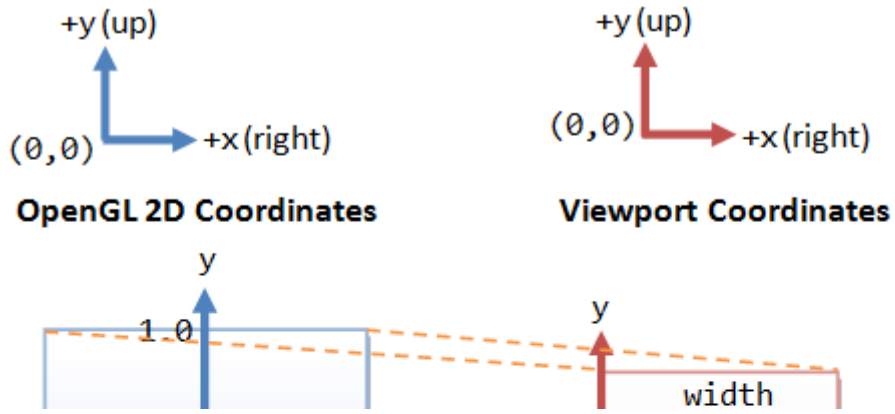
We similarly define a second quad in green.

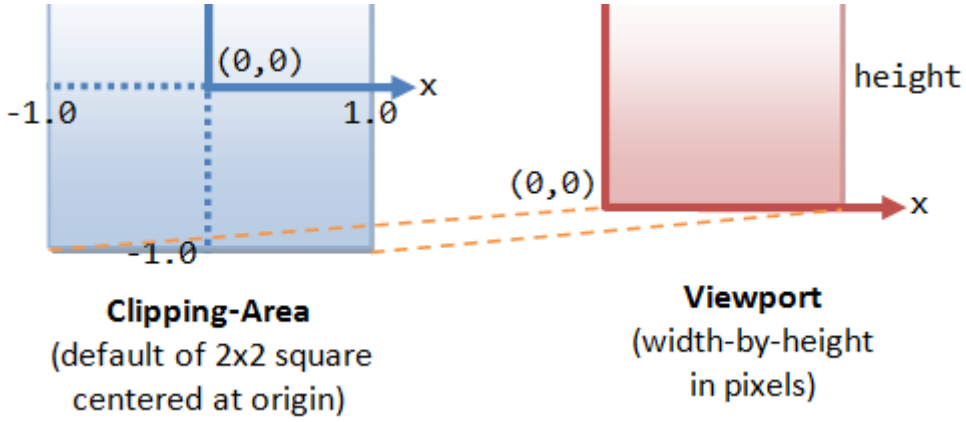
For the third quad (as follows), the vertices have different color. The color of the quad surface is interpolated from its vertices, resulting in a shades of white to dark gray, as shown in the output.

```
glColor3f(0.2f, 0.2f, 0.2f);  // Dark Gray
glVertex2f(-0.9f, -0.7f);
glColor3f(1.0f, 1.0f, 1.0f);  // White
glVertex2f(-0.5f, -0.7f);
glColor3f(0.2f, 0.2f, 0.2f);  // Dark Gray
glVertex2f(-0.5f, -0.3f);
glColor3f(1.0f, 1.0f, 1.0f);  // White
glVertex2f(-0.9f, -0.3f);
```

3.9 2D Coordinate System and the Default View

The following diagram shows the OpenGL 2D Coordinate System, which corresponds to the everyday 2D Cartesian coordinates with origin located at the bottom-left corner.





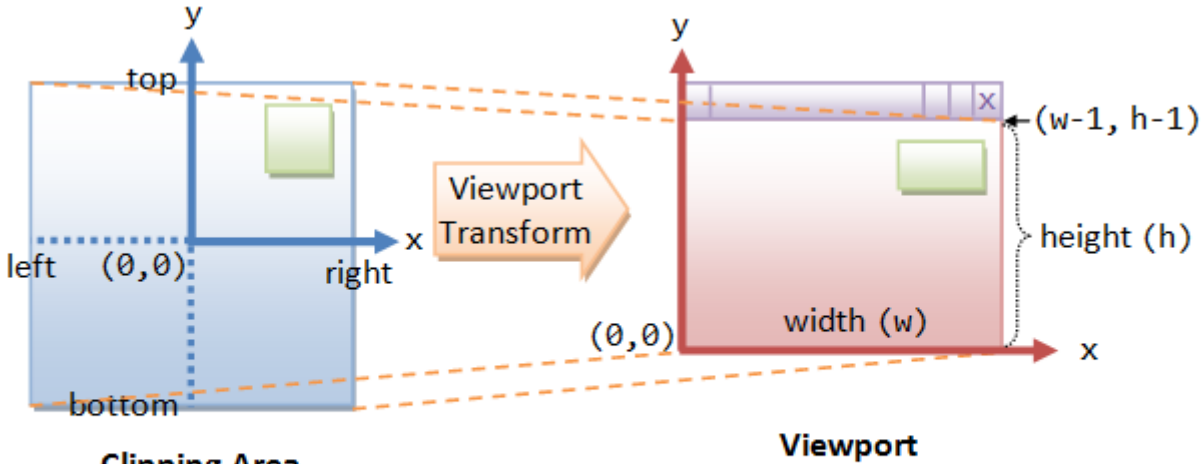
The default OpenGL 2D *clipping-area* (i.e., what is captured by the camera) is an orthographic view with x and y in the range of -1.0 and 1.0, i.e., a 2x2 square with centered at the origin. This clipping-area is mapped to the *viewport* on the screen. Viewport is measured in pixels.

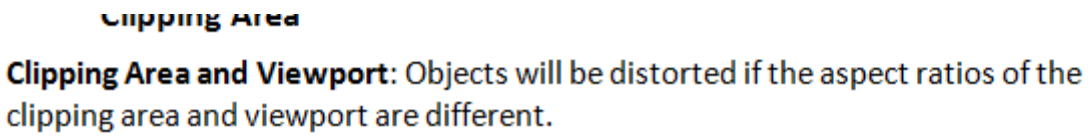
Study the above example to convince yourself that the 2D shapes created are positioned correctly on the screen.

4. Clipping-Area & Viewport

Try dragging the corner of the window to make it bigger or smaller. Observe that all the shapes are distorted.

We can handle the re-sizing of window via a callback handler `reshape()`, which can be programmed to adjust the OpenGL clipping-area according to the window's aspect ratio.





Clipping Area: *Clipping area* refers to the area that can be seen (i.e., captured by the camera), measured in OpenGL coordinates.

The function `gluOrtho2D` can be used to set the clipping area of 2D orthographic view. Objects outside the clipping area will be *clipped* away and cannot be seen.

```
void gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top)
   // The default clipping area is (-1.0, 1.0, -1.0, 1.0) in OpenGL coordinates,
   // i.e., 2x2 square centered at the origin.
```

To set the clipping area, we need to issue a series of commands as follows: we first select the so-called *projection matrix* for operation, and reset the projection matrix to identity. We then choose the 2D orthographic view with the desired clipping area, via `gluOrtho2D()`.

```
// Set to 2D orthographic projection with the specified clipping area
glMatrixMode(GL_PROJECTION);      // Select the Projection matrix for operation
glLoadIdentity();                 // Reset Projection matrix
gluOrtho2D(-1.0, 1.0, -1.0, 1.0); // Set clipping area's left, right, bottom, top
```

Viewport: *Viewport* refers to the display area on the window (screen), which is measured in pixels in screen coordinates (excluding the title bar).

The clipping area is mapped to the viewport. We can use `glViewport` function to configure the viewport.

```
void glViewport(GLint xTopLeft, GLint yTopLeft, GLsizei width, GLsizei height)
```

Suppose the the clipping area's (left, right, bottom, top) is (-1.0, 1.0, -1.0, 1.0) (in OpenGL coordinates) and the viewport's (xTopLeft, xTopRight, width, height) is (0, 0, 640, 480) (in screen coordinates in pixels), then the bottom-left corner (-1.0, -1.0) maps to (0, 0) in the viewport, the top-right corner (1.0, 1.0) maps to (639, 479). It is obvious that if the *aspect ratios* for the clipping area and the viewport are not the same, the shapes will be distorted.

Take note that in the earlier example, the windows' size of 320x320 has a square shape, with a aspect ratio consistent with the default 2x2 squarish clipping-area.

4.1 Example 3: Clipping-area and Viewport (`GL03Viewport.cpp`)

```
1  /*
2   * GL03Viewport.cpp: Clipping-area and Viewport
```

```
 * Implementing reshape to ensure same aspect ratio between the
 * clipping-area and the viewport.
 */
#include <windows.h>  // for MS Windows
#include <GL/glut.h>  // GLUT, include glu.h and gl.h

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

void display() {
   glClear(GL_COLOR_BUFFER_BIT);   // Clear the color buffer with current clearing color

   // Define shapes enclosed within a pair of glBegin and glEnd
   glBegin(GL_QUADS);              // Each set of 4 vertices form a quad
      glColor3f(1.0f, 0.0f, 0.0f); // Red
      glVertex2f(-0.8f, 0.1f);     // Define vertices in counter-clockwise (CCW) order
      glVertex2f(-0.2f, 0.1f);     //  so that the normal (front-face) is facing you
      glVertex2f(-0.2f, 0.7f);
      glVertex2f(-0.8f, 0.7f);

      glColor3f(0.0f, 1.0f, 0.0f); // Green
      glVertex2f(-0.7f, -0.6f);
      glVertex2f(-0.1f, -0.6f);
      glVertex2f(-0.1f,  0.0f);
      glVertex2f(-0.7f,  0.0f);

      glColor3f(0.2f, 0.2f, 0.2f); // Dark Gray
      glVertex2f(-0.9f, -0.7f);
      glColor3f(1.0f, 1.0f, 1.0f); // White
      glVertex2f(-0.5f, -0.7f);
      glColor3f(0.2f, 0.2f, 0.2f); // Dark Gray
      glVertex2f(-0.5f, -0.3f);
      glColor3f(1.0f, 1.0f, 1.0f); // White
```

```
      glVertex2f(-0.9f, -0.3f);
   glEnd();

   glBegin(GL_TRIANGLES);          // Each set of 3 vertices form a triangle
      glColor3f(0.0f, 0.0f, 1.0f); // Blue
      glVertex2f(0.1f, -0.6f);
      glVertex2f(0.7f, -0.6f);
      glVertex2f(0.4f, -0.1f);

      glColor3f(1.0f, 0.0f, 0.0f); // Red
      glVertex2f(0.3f, -0.4f);
      glColor3f(0.0f, 1.0f, 0.0f); // Green
      glVertex2f(0.9f, -0.4f);
      glColor3f(0.0f, 0.0f, 1.0f); // Blue
      glVertex2f(0.6f, -0.9f);
   glEnd();

   glBegin(GL_POLYGON);            // These vertices form a closed polygon
      glColor3f(1.0f, 1.0f, 0.0f); // Yellow
      glVertex2f(0.4f, 0.2f);
      glVertex2f(0.6f, 0.2f);
      glVertex2f(0.7f, 0.4f);
      glVertex2f(0.6f, 0.6f);
      glVertex2f(0.4f, 0.6f);
      glVertex2f(0.3f, 0.4f);
   glEnd();

   glFlush();  // Render now
}

/* Handler for window re-size event. Called back when the window first appears and
   whenever the window is re-sized with its new width and height */
void reshape(GLsizei width, GLsizei height) {  // GLsizei for non-negative integer
   // Compute aspect ratio of the new window
   if (height == 0) height = 1;                // To prevent divide by 0
   GLfloat aspect = (GLfloat)width / (GLfloat)height;
```

```
75
76      // Set the viewport to cover the new window
77      glViewport(0, 0, width, height);
78
79      // Set the aspect ratio of the clipping area to match the viewport
80      glMatrixMode(GL_PROJECTION);  // To operate on the Projection matrix
81      glLoadIdentity();             // Reset the projection matrix
82      if (width >= height) {
83        // aspect >= 1, set the height from -1 to 1, with larger width
84         gluOrtho2D(-1.0 * aspect, 1.0 * aspect, -1.0, 1.0);
85      } else {
86         // aspect < 1, set the width to -1 to 1, with larger height
87        gluOrtho2D(-1.0, 1.0, -1.0 / aspect, 1.0 / aspect);
88      }
89   }
90
91   /* Main function: GLUT runs as a console application starting at main() */
92   int main(int argc, char** argv) {
93      glutInit(&argc, argv);          // Initialize GLUT
94      glutInitWindowSize(640, 480);   // Set the window's initial width & height - non-square
95      glutInitWindowPosition(50, 50); // Position the window's initial top-left corner
96      glutCreateWindow("Viewport Transform");  // Create window with the given title
97      glutDisplayFunc(display);       // Register callback handler for window re-paint event
98      glutReshapeFunc(reshape);       // Register callback handler for window re-size event
99      initGL();                       // Our own OpenGL initialization
100     glutMainLoop();                 // Enter the infinite event-processing loop
101     return 0;
102  }
```

A `reshape()` function, which is called back when the window first appears and whenever the window is re-sized, can be used to ensure consistent aspect ratio between clipping-area and viewport, as shown in the above example. The graphics sub-system passes the window's width and height, in pixels, into the `reshape()`.

```
GLfloat aspect = (GLfloat)width / (GLfloat)height;
```

We compute the aspect ratio of the new re-sized window, given its new `width` and `height` provided by the graphics sub-system to the callback function

`reshape()`.

```
glViewport(0, 0, width, height);
```

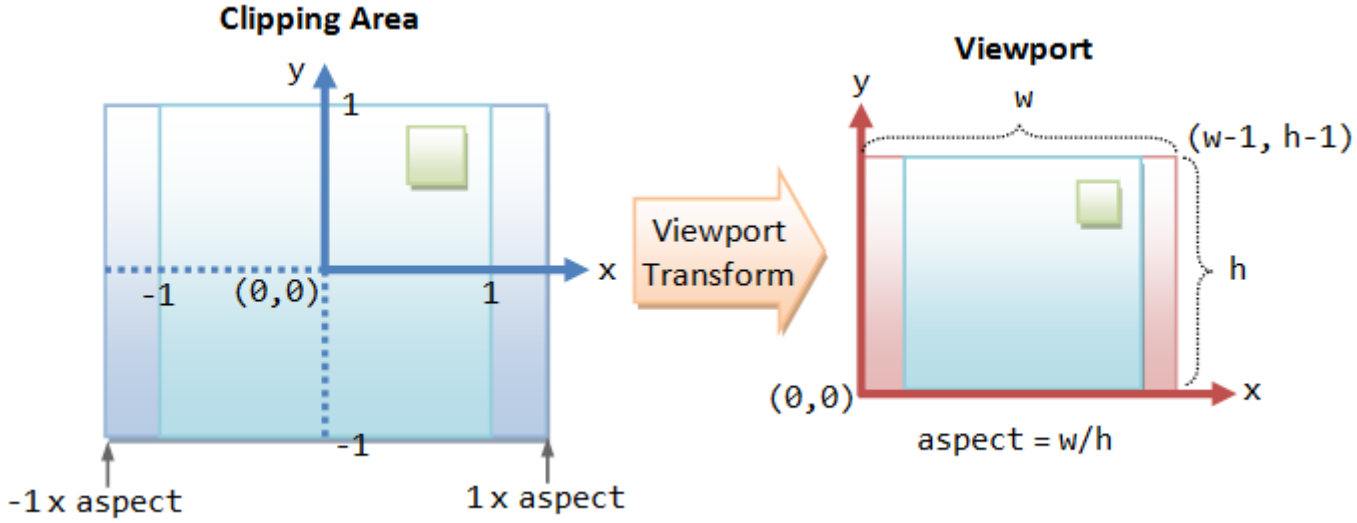
We set the viewport to cover the entire new re-sized window, in pixels.

Try setting the viewport to cover only a quarter (lower-right qradrant) of the window via `glViewport(0, 0, width/2, height/2)`.

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
if (width >= height) {
   gluOrtho2D(-1.0 * aspect, 1.0 * aspect, -1.0, 1.0);
} else {
   gluOrtho2D(-1.0, 1.0, -1.0 / aspect, 1.0 / aspect);
}
```

We set the aspect ratio of the clipping area to match the viewport. To set the clipping area, we first choose the operate on the projection matrix via `glMatrixMode(GL_PROJECTION)`. OpenGL has two matrices, a projection matrix (which deals with camera projection such as setting the clipping area) and a model-view matrix (for transforming the objects from their local spaces to the common world space). We reset the projection matrix via `glLoadIdentity()`.

Finally, we invoke `gluOrtho2D()` to set the clipping area with an aspect ratio matching the viewport. The shorter side has the range from -1 to +1, as illustrated below:



Clipping Area and Viewport: same aspect ratio for the clipping area and viewport to ensure that the objects are not distorted.

We need to register the `reshape()` callback handler with GLUT via `glutReshapeFunc()` in the `main()` as follows:

```
int main(int argc, char** argv) {
   glutInitWindowSize(640, 480);
   ......
   glutReshapeFunc(reshape);
}
```

In the above `main()` function, we specify the initial window size to `640x480`, which is non-squarish. Try re-sizing the window and observe the changes.

Note that the `reshape()` runs at least *once* when the window first appears. It is then called back whenever the window is re-shaped. On the other hand, the `initGL()` runs once (and only once); and the `display()` runs in response to window re-paint request (e.g., after the window is re-sized).

5. Translation & Rotation

In the above sample, we positioned each of the shapes by defining their vertices with respective to the *same* origin (called *world space*). It took me quite a while to figure out the absolute coordinates of these vertices.

Instead, we could position each of the shapes by defining their vertices with respective to their own center (called *model space* or *local space*). We can then use translation and/or rotation to position the shapes at the desired locations in the world space, as shown in the following revised `display()` function.

5.1 Example 4: Translation and Rotation (`GL04ModelTransform.cpp`)

```
1  /*
2   * GL04ModelTransform.cpp: Model Transform - Translation and Rotation
3   * Transform primitives from their model spaces to world space.
4   */
5  #include <windows.h>  // for MS Windows
6  #include <GL/glut.h>  // GLUT, include glu.h and gl.h
7
8  /* Initialize OpenGL Graphics */
```

```
 9  void initGL() {
10     // Set "clearing" or background color
11     glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Black and opaque
12  }
13
14  /* Handler for window-repaint event. Call back when the window first appears and
15     whenever the window needs to be re-painted. */
16  void display() {
17     glClear(GL_COLOR_BUFFER_BIT);    // Clear the color buffer
18     glMatrixMode(GL_MODELVIEW);      // To operate on Model-View matrix
19     glLoadIdentity();                // Reset the model-view matrix
20
21     glTranslatef(-0.5f, 0.4f, 0.0f); // Translate left and up
22     glBegin(GL_QUADS);               // Each set of 4 vertices form a quad
23        glColor3f(1.0f, 0.0f, 0.0f);  // Red
24        glVertex2f(-0.3f, -0.3f);     // Define vertices in counter-clockwise (CCW) order
25        glVertex2f( 0.3f, -0.3f);     //  so that the normal (front-face) is facing you
26        glVertex2f( 0.3f,  0.3f);
27        glVertex2f(-0.3f,  0.3f);
28     glEnd();
29
30     glTranslatef(0.1f, -0.7f, 0.0f); // Translate right and down
31     glBegin(GL_QUADS);               // Each set of 4 vertices form a quad
32        glColor3f(0.0f, 1.0f, 0.0f); // Green
33        glVertex2f(-0.3f, -0.3f);
34        glVertex2f( 0.3f, -0.3f);
35        glVertex2f( 0.3f,  0.3f);
36        glVertex2f(-0.3f,  0.3f);
37     glEnd();
38
39     glTranslatef(-0.3f, -0.2f, 0.0f); // Translate left and down
40     glBegin(GL_QUADS);                 // Each set of 4 vertices form a quad
41        glColor3f(0.2f, 0.2f, 0.2f); // Dark Gray
42        glVertex2f(-0.2f, -0.2f);
43        glColor3f(1.0f, 1.0f, 1.0f); // White
44        glVertex2f( 0.2f, -0.2f);
```

```
45          glColor3f(0.2f, 0.2f, 0.2f); // Dark Gray
46          glVertex2f( 0.2f,  0.2f);
47          glColor3f(1.0f, 1.0f, 1.0f); // White
48          glVertex2f(-0.2f,  0.2f);
49       glEnd();
50
51       glTranslatef(1.1f, 0.2f, 0.0f); // Translate right and up
52       glBegin(GL_TRIANGLES);          // Each set of 3 vertices form a triangle
53          glColor3f(0.0f, 0.0f, 1.0f); // Blue
54          glVertex2f(-0.3f, -0.2f);
55          glVertex2f( 0.3f, -0.2f);
56          glVertex2f( 0.0f,  0.3f);
57       glEnd();
58
59       glTranslatef(0.2f, -0.3f, 0.0f);     // Translate right and down
60       glRotatef(180.0f, 0.0f, 0.0f, 1.0f); // Rotate 180 degree
61          glBegin(GL_TRIANGLES);               // Each set of 3 vertices form a triangle
62          glColor3f(1.0f, 0.0f, 0.0f); // Red
63          glVertex2f(-0.3f, -0.2f);
64          glColor3f(0.0f, 1.0f, 0.0f); // Green
65          glVertex2f( 0.3f, -0.2f);
66          glColor3f(0.0f, 0.0f, 1.0f); // Blue
67          glVertex2f( 0.0f,  0.3f);
68       glEnd();
69
70       glRotatef(-180.0f, 0.0f, 0.0f, 1.0f); // Undo previous rotate
71       glTranslatef(-0.1f, 1.0f, 0.0f);      // Translate right and down
72       glBegin(GL_POLYGON);                  // The vertices form one closed polygon
73          glColor3f(1.0f, 1.0f, 0.0f); // Yellow
74          glVertex2f(-0.1f, -0.2f);
75          glVertex2f( 0.1f, -0.2f);
76          glVertex2f( 0.2f,  0.0f);
77          glVertex2f( 0.1f,  0.2f);
78          glVertex2f(-0.1f,  0.2f);
79          glVertex2f(-0.2f,  0.0f);
80       glEnd();
```

```
81
82     glFlush();   // Render now
83  }
84
85  /* Handler for window re-size event. Called back when the window first appears and
86     whenever the window is re-sized with its new width and height */
87  void reshape(GLsizei width, GLsizei height) {  // GLsizei for non-negative integer
88     // Compute aspect ratio of the new window
89     if (height == 0) height = 1;                // To prevent divide by 0
90     GLfloat aspect = (GLfloat)width / (GLfloat)height;
91
92     // Set the viewport to cover the new window
93     glViewport(0, 0, width, height);
94
95     // Set the aspect ratio of the clipping area to match the viewport
96     glMatrixMode(GL_PROJECTION);  // To operate on the Projection matrix
97     glLoadIdentity();
98     if (width >= height) {
99       // aspect >= 1, set the height from -1 to 1, with larger width
100       gluOrtho2D(-1.0 * aspect, 1.0 * aspect, -1.0, 1.0);
101    } else {
102       // aspect < 1, set the width to -1 to 1, with larger height
103     gluOrtho2D(-1.0, 1.0, -1.0 / aspect, 1.0 / aspect);
104    }
105 }
106
107 /* Main function: GLUT runs as a console application starting at main() */
108 int main(int argc, char** argv) {
109    glutInit(&argc, argv);          // Initialize GLUT
110    glutInitWindowSize(640, 480);   // Set the window's initial width & height - non-square
111    glutInitWindowPosition(50, 50); // Position the window's initial top-left corner
112    glutCreateWindow("Model Transform");  // Create window with the given title
113    glutDisplayFunc(display);       // Register callback handler for window re-paint event
114    glutReshapeFunc(reshape);       // Register callback handler for window re-size event
115    initGL();                       // Our own OpenGL initialization
116    glutMainLoop();                 // Enter the infinite event-processing loop
```

```
117     return 0;
118  }
```

```
glMatrixMode(GL_MODELVIEW); // To operate on model-view matrix
glLoadIdentity();           // Reset
```

Translation and rotation are parts of so-called *model transform*, which transform from the objects from the local space (or model space) to the common world space. To carry out model transform, we set the matrix mode to mode-view matrix (`GL_MODELVIEW`) and reset the matrix. (Recall that in the previous example, we set the matrix mode to projection matrix (`GL_PROJECTION`) to set the clipping area.)

OpenGL is operating as a state machine. That is, once a state is set, the value of the state persists until it is changed. In other words, once the coordinates are translated or rotated, all the subsequent operations will be based on this coordinates.

Translation is done via `glTranslate` function:

```
void gltranslatef (GLfloat x, GLfloat y, GLfloat z)
   // where (x, y, z) is the translational vector
```

Take note that `glTranslatef` function must be placed outside the `glBegin`/`glEnd`, where as `glColor` can be placed inside `glBegin`/`glEnd`.

Rotation is done via `glRotatef` function:

```
void glRotatef (GLfloat angle, GLfloat x, GLfloat y, GLfloat z)
   // where angle specifies the rotation in degree, (x, y, z) forms the axis of rotation.
```

Take note that the rotational angle is measured in degrees (instead of radians) in OpenGL.

In the above example, we translate within the x-y plane (z=0) and rotate about the z-axis (which is normal to the x-y plane).

6. 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).
```

```
 */
#include <windows.h>  // for MS Windows
#include <GL/glut.h>  // GLUT, include glu.h and gl.h

// 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
```

```
40
41      glPushMatrix();                     // Save model-view matrix setting
42      glTranslatef(-0.4f, -0.3f, 0.0f);   // Translate
43      glRotatef(angle, 0.0f, 0.0f, 1.0f); // rotate by angle in degrees
44      glBegin(GL_QUADS);
45         glColor3f(0.0f, 1.0f, 0.0f); // Green
46         glVertex2f(-0.3f, -0.3f);
47         glVertex2f( 0.3f, -0.3f);
48         glVertex2f( 0.3f,  0.3f);
49         glVertex2f(-0.3f,  0.3f);
50      glEnd();
51      glPopMatrix();                      // Restore the model-view matrix
52
53      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 */
114 void 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() */
135 int 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-square
139    glutInitWindowPosition(50, 50); // Position the window's initial top-left corner
140    glutCreateWindow("Animation via Idle Function");  // Create window with the given title
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
9  GLfloat angle = 0.0f;  // rotational angle of the shapes
10 int refreshMills = 30; // refresh interval in milliseconds
11
12 /* Initialize OpenGL Graphics */
13 void 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 */
19 void 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. */
26 void 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);
40    glEnd();
41    glPopMatrix();                      // Restore the model-view matrix
42
43    glPushMatrix();                     // Save model-view matrix setting
44    glTranslatef(-0.4f, -0.3f, 0.0f);   // Translate
45    glRotatef(angle, 0.0f, 0.0f, 1.0f); // rotate by angle in degrees
46    glBegin(GL_QUADS);
47       glColor3f(0.0f, 1.0f, 0.0f); // Green
48       glVertex2f(-0.3f, -0.3f);
49       glVertex2f( 0.3f, -0.3f);
50       glVertex2f( 0.3f,  0.3f);
51       glVertex2f(-0.3f,  0.3f);
52    glEnd();
53    glPopMatrix();                      // Restore the model-view matrix
54
55    glPushMatrix();                     // Save model-view matrix setting
56    glTranslatef(-0.7f, -0.5f, 0.0f);   // Translate
57    glRotatef(angle, 0.0f, 0.0f, 1.0f); // rotate by angle in degrees
58    glBegin(GL_QUADS);
59       glColor3f(0.2f, 0.2f, 0.2f); // Dark Gray
```

```
60          glVertex2f(-0.2f, -0.2f);
61          glColor3f(1.0f, 1.0f, 1.0f); // White
62          glVertex2f( 0.2f, -0.2f);
63          glColor3f(0.2f, 0.2f, 0.2f); // Dark Gray
64          glVertex2f( 0.2f,  0.2f);
65          glColor3f(1.0f, 1.0f, 1.0f); // White
66          glVertex2f(-0.2f,  0.2f);
67       glEnd();
68       glPopMatrix();                     // Restore the model-view matrix
69
70       glPushMatrix();                    // Save model-view matrix setting
71       glTranslatef(0.4f, -0.3f, 0.0f);   // Translate
72       glRotatef(angle, 0.0f, 0.0f, 1.0f); // rotate by angle in degrees
73       glBegin(GL_TRIANGLES);
74          glColor3f(0.0f, 0.0f, 1.0f); // Blue
75          glVertex2f(-0.3f, -0.2f);
76          glVertex2f( 0.3f, -0.2f);
77          glVertex2f( 0.0f,  0.3f);
78       glEnd();
79       glPopMatrix();                     // Restore the model-view matrix
80
81       glPushMatrix();                    // Save model-view matrix setting
82       glTranslatef(0.6f, -0.6f, 0.0f);   // Translate
83       glRotatef(180.0f + angle, 0.0f, 0.0f, 1.0f); // Rotate 180+angle degree
84       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 */
116   void 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() */
137 int 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-square
141    glutInitWindowPosition(50, 50); // Position the window's initial top-left corner
142    glutCreateWindow("Animation via Idle Function");  // Create window with the given title
143    glutDisplayFunc(display);       // Register callback handler for window re-paint event
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 '`|`'.

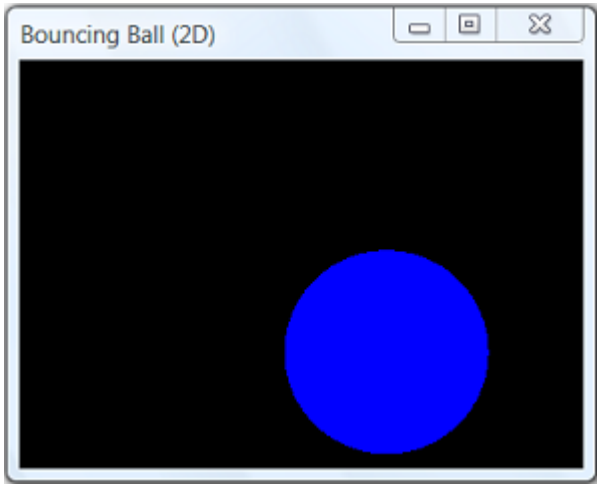
  ```
  void glutInitDisplayMode(unsigned int displayMode)
  ```

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
10 char title[] = "Bouncing Ball (2D)";  // Windowed mode's title
11 int windowWidth  = 640;     // Windowed mode's width
12 int windowHeight = 480;     // Windowed mode's height
13 int windowPosX   = 50;      // Windowed mode's top-left corner x
```

```
14  int windowPosY   = 50;      // Windowed mode's top-left corner y
15
16  GLfloat ballRadius = 0.5f;   // Radius of the bouncing ball
17  GLfloat ballX = 0.0f;         // Ball's center (x, y) position
18  GLfloat ballY = 0.0f;
19  GLfloat ballXMax, ballXMin, ballYMax, ballYMin; // Ball's center (x, y) bounds
20  GLfloat xSpeed = 0.02f;      // Ball's speed in x and y directions
21  GLfloat ySpeed = 0.007f;
22  int refreshMillis = 30;      // Refresh period in milliseconds
23
24  // Projection clipping area
25  GLdouble clipAreaXLeft, clipAreaXRight, clipAreaYBottom, clipAreaYTop;
26
27  /* Initialize OpenGL Graphics */
28  void 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 */
33  void 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 */
74   void 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 */
104    void 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() */
110    int 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, y)
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  }
```

[TODO] Explanation

7. Handling Keyboard Inputs with GLUT

We can register callback functions to handle keyboard inputs for normal and special keys, respectively.

- `glutKeyboardFunc`: registers callback handler for keyboard event.

```
void glutKeyboardFunc (void (*func)(unsigned char key, int x, int y)
   // key is the char pressed, e.g., 'a' or 27 for ESC
   // (x, y) is the mouse location in Windows' coordinates
```

- `glutSpecialFunc`: registers callback handler for special key (such as arrow keys and function keys).

```
void glutSpecialFunc (void (*func)(int specialKey, int x, int y)
   // specialKey: GLUT_KEY_* (* for LEFT, RIGHT, UP, DOWN, HOME, END, PAGE_UP, PAGE_DOWN, F1,...F12).
   // (x, y) is the mouse location in Windows' coordinates
```

7.1 Example 8: Switching between Full-Screen and Windowed-mode (GL08FullScreen.cpp)

For the bouncing ball program, the following special-key handler toggles between *full-screen* and *windowed modes* using F1 key.

```
1  /*
2   * GL08FullScreen.cpp: Switching between full-screen mode and windowed-mode
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
10 char title[] = "Full-Screen & Windowed Mode";  // Windowed mode's title
11 int windowWidth  = 640;     // Windowed mode's width
12 int windowHeight = 480;     // Windowed mode's height
```

```
13  int windowPosX   = 50;      // Windowed mode's top-left corner x
14  int windowPosY   = 50;      // Windowed mode's top-left corner y
15
16  GLfloat ballRadius = 0.5f;   // Radius of the bouncing ball
17  GLfloat ballX = 0.0f;         // Ball's center (x, y) position
18  GLfloat ballY = 0.0f;
19  GLfloat ballXMax, ballXMin, ballYMax, ballYMin; // Ball's center (x, y) bounds
20  GLfloat xSpeed = 0.02f;      // Ball's speed in x and y directions
21  GLfloat ySpeed = 0.007f;
22  int refreshMillis = 30;      // Refresh period in milliseconds
23
24  // Projection clipping area
25  GLdouble clipAreaXLeft, clipAreaXRight, clipAreaYBottom, clipAreaYTop;
26
27  bool fullScreenMode = true; // Full-screen or windowed mode?
28
29  /* Initialize OpenGL Graphics */
30  void initGL() {
31     glClearColor(0.0, 0.0, 0.0, 1.0); // Set background (clear) color to black
32  }
33
34  /* Callback handler for window re-paint event */
35  void display() {
36     glClear(GL_COLOR_BUFFER_BIT);  // Clear the color buffer
37     glMatrixMode(GL_MODELVIEW);    // To operate on the model-view matrix
38     glLoadIdentity();              // Reset model-view matrix
39
40     glTranslatef(ballX, ballY, 0.0f);  // Translate to (xPos, yPos)
41     // Use triangular segments to form a circle
42     glBegin(GL_TRIANGLE_FAN);
43        glColor3f(0.0f, 0.0f, 1.0f);  // Blue
44        glVertex2f(0.0f, 0.0f);       // Center of circle
45        int numSegments = 100;
46        GLfloat angle;
47        for (int i = 0; i <= numSegments; i++) { // Last vertex same as first vertex
48           angle = i * 2.0f * PI / numSegments;  // 360 deg for all segments
```

```
49          glVertex2f(cos(angle) * ballRadius, sin(angle) * ballRadius);
50       }
51    glEnd();
52
53    glutSwapBuffers();  // Swap front and back buffers (of double buffered mode)
54
55    // Animation Control - compute the location for the next refresh
56    ballX += xSpeed;
57    ballY += ySpeed;
58    // Check if the ball exceeds the edges
59    if (ballX > ballXMax) {
60       ballX = ballXMax;
61       xSpeed = -xSpeed;
62    } else if (ballX < ballXMin) {
63       ballX = ballXMin;
64       xSpeed = -xSpeed;
65    }
66    if (ballY > ballYMax) {
67       ballY = ballYMax;
68       ySpeed = -ySpeed;
69    } else if (ballY < ballYMin) {
70       ballY = ballYMin;
71       ySpeed = -ySpeed;
72    }
73 }
74
75 /* Call back when the windows is re-sized */
76 void reshape(GLsizei width, GLsizei height) {
77    // Compute aspect ratio of the new window
78    if (height == 0) height = 1;                // To prevent divide by 0
79    GLfloat aspect = (GLfloat)width / (GLfloat)height;
80
81    // Set the viewport to cover the new window
82    glViewport(0, 0, width, height);
83
84    // Set the aspect ratio of the clipping area to match the viewport
```

```
 85      glMatrixMode(GL_PROJECTION);  // To operate on the Projection matrix
 86      glLoadIdentity();             // Reset the projection matrix
 87      if (width >= height) {
 88         clipAreaXLeft   = -1.0 * aspect;
 89         clipAreaXRight  = 1.0 * aspect;
 90         clipAreaYBottom = -1.0;
 91         clipAreaYTop    = 1.0;
 92      } else {
 93         clipAreaXLeft   = -1.0;
 94         clipAreaXRight  = 1.0;
 95         clipAreaYBottom = -1.0 / aspect;
 96         clipAreaYTop    = 1.0 / aspect;
 97      }
 98      gluOrtho2D(clipAreaXLeft, clipAreaXRight, clipAreaYBottom, clipAreaYTop);
 99      ballXMin = clipAreaXLeft + ballRadius;
100      ballXMax = clipAreaXRight - ballRadius;
101      ballYMin = clipAreaYBottom + ballRadius;
102      ballYMax = clipAreaYTop - ballRadius;
103   }
104
105   /* Called back when the timer expired */
106   void Timer(int value) {
107      glutPostRedisplay();    // Post a paint request to activate display()
108      glutTimerFunc(refreshMillis, Timer, 0); // subsequent timer call at milliseconds
109   }
110
111   /* Callback handler for special-key event */
112   void specialKeys(int key, int x, int y) {
113      switch (key) {
114         case GLUT_KEY_F1:    // F1: Toggle between full-screen and windowed mode
115            fullScreenMode = !fullScreenMode;         // Toggle state
116            if (fullScreenMode) {                     // Full-screen mode
117               windowPosX   = glutGet(GLUT_WINDOW_X); // Save parameters for restoring later
118               windowPosY   = glutGet(GLUT_WINDOW_Y);
119               windowWidth  = glutGet(GLUT_WINDOW_WIDTH);
120               windowHeight = glutGet(GLUT_WINDOW_HEIGHT);
```

```
            glutFullScreen();                       // Switch into full screen
         } else {                                          // Windowed mode
            glutReshapeWindow(windowWidth, windowHeight); // Switch into windowed mode
            glutPositionWindow(windowPosX, windowPosX);   // Position top-left corner
         }
         break;
   }
}

/* Main function: GLUT runs as a console application starting at main() */
int main(int argc, char** argv) {
   glutInit(&argc, argv);            // Initialize GLUT
   glutInitDisplayMode(GLUT_DOUBLE); // Enable double buffered mode
   glutInitWindowSize(windowWidth, windowHeight);  // Initial window width and height
   glutInitWindowPosition(windowPosX, windowPosY); // Initial window top-left corner (x, y)
   glutCreateWindow(title);      // Create window with given title
   glutDisplayFunc(display);     // Register callback handler for window re-paint
   glutReshapeFunc(reshape);     // Register callback handler for window re-shape
   glutTimerFunc(0, Timer, 0);   // First timer call immediately
   glutSpecialFunc(specialKeys); // Register callback handler for special-key event
   glutFullScreen();             // Put into full screen
   initGL();                     // Our own OpenGL initialization
   glutMainLoop();               // Enter event-processing loop
   return 0;
}
```

[TODO] Explanation

[TODO] Using `glVertex` to draw a Circle is inefficient (due to the compute-intensive `sin()` and `cos()` functions). Try using GLU's quadric.

7.2 Example 9: Key-Controlled (GL09KeyControl.cpp)

For the bouncing ball program, the following key and special-key handlers provide exits with ESC (27), increase/decrease y speed with up-/down-arrow key, increase/decrease x speed with left-/right-arrow key, increase/decrease ball's radius with PageUp/PageDown key.

```
/*
 * GL09KeyControl.cpp: A key-controlled bouncing ball
```

```
 */
#include <windows.h>  // for MS Windows
#include <GL/glut.h>  // GLUT, include glu.h and gl.h
#include <Math.h>     // Needed for sin, cos
#define PI 3.14159265f

// Global variables
char title[] = "Full-Screen & Windowed Mode";  // Windowed mode's title
int windowWidth  = 640;     // Windowed mode's width
int windowHeight = 480;     // Windowed mode's height
int windowPosX   = 50;      // Windowed mode's top-left corner x
int windowPosY   = 50;      // Windowed mode's top-left corner y

GLfloat ballRadius = 0.5f;   // Radius of the bouncing ball
GLfloat ballX = 0.0f;        // Ball's center (x, y) position
GLfloat ballY = 0.0f;
GLfloat ballXMax, ballXMin, ballYMax, ballYMin; // Ball's center (x, y) bounds
GLfloat xSpeed = 0.02f;      // Ball's speed in x and y directions
GLfloat ySpeed = 0.007f;
int refreshMillis = 30;      // Refresh period in milliseconds

// Projection clipping area
GLdouble clipAreaXLeft, clipAreaXRight, clipAreaYBottom, clipAreaYTop;

bool fullScreenMode = true; // Full-screen or windowed mode?

/* Initialize OpenGL Graphics */
void initGL() {
   glClearColor(0.0, 0.0, 0.0, 1.0); // Set background (clear) color to black
}

/* Callback handler for window re-paint event */
void display() {
   glClear(GL_COLOR_BUFFER_BIT);  // Clear the color buffer
   glMatrixMode(GL_MODELVIEW);    // To operate on the model-view matrix
   glLoadIdentity();              // Reset model-view matrix
```

```
39
40      glTranslatef(ballX, ballY, 0.0f);  // Translate to (xPos, yPos)
41      // Use triangular segments to form a circle
42      glBegin(GL_TRIANGLE_FAN);
43         glColor3f(0.0f, 0.0f, 1.0f);  // Blue
44         glVertex2f(0.0f, 0.0f);       // Center of circle
45         int numSegments = 100;
46         GLfloat angle;
47         for (int i = 0; i <= numSegments; i++) { // Last vertex same as first vertex
48            angle = i * 2.0f * PI / numSegments;  // 360 deg for all segments
49            glVertex2f(cos(angle) * ballRadius, sin(angle) * ballRadius);
50         }
51      glEnd();
52
53      glutSwapBuffers();  // Swap front and back buffers (of double buffered mode)
54
55      // Animation Control - compute the location for the next refresh
56      ballX += xSpeed;
57      ballY += ySpeed;
58      // Check if the ball exceeds the edges
59      if (ballX > ballXMax) {
60         ballX = ballXMax;
61         xSpeed = -xSpeed;
62      } else if (ballX < ballXMin) {
63         ballX = ballXMin;
64         xSpeed = -xSpeed;
65      }
66      if (ballY > ballYMax) {
67         ballY = ballYMax;
68         ySpeed = -ySpeed;
69      } else if (ballY < ballYMin) {
70         ballY = ballYMin;
71         ySpeed = -ySpeed;
72      }
73   }
74
```

```
 75 /* Call back when the windows is re-sized */
 76 void reshape(GLsizei width, GLsizei height) {
 77    // Compute aspect ratio of the new window
 78    if (height == 0) height = 1;                // To prevent divide by 0
 79    GLfloat aspect = (GLfloat)width / (GLfloat)height;
 80
 81    // Set the viewport to cover the new window
 82    glViewport(0, 0, width, height);
 83
 84    // Set the aspect ratio of the clipping area to match the viewport
 85    glMatrixMode(GL_PROJECTION);  // To operate on the Projection matrix
 86    glLoadIdentity();             // Reset the projection matrix
 87    if (width >= height) {
 88       clipAreaXLeft   = -1.0 * aspect;
 89       clipAreaXRight  = 1.0 * aspect;
 90       clipAreaYBottom = -1.0;
 91       clipAreaYTop    = 1.0;
 92    } else {
 93       clipAreaXLeft   = -1.0;
 94       clipAreaXRight  = 1.0;
 95       clipAreaYBottom = -1.0 / aspect;
 96       clipAreaYTop    = 1.0 / aspect;
 97    }
 98    gluOrtho2D(clipAreaXLeft, clipAreaXRight, clipAreaYBottom, clipAreaYTop);
 99    ballXMin = clipAreaXLeft + ballRadius;
100    ballXMax = clipAreaXRight - ballRadius;
101    ballYMin = clipAreaYBottom + ballRadius;
102    ballYMax = clipAreaYTop - ballRadius;
103 }
104
105 /* Called back when the timer expired */
106 void Timer(int value) {
107    glutPostRedisplay();    // Post a paint request to activate display()
108    glutTimerFunc(refreshMillis, Timer, 0); // subsequent timer call at milliseconds
109 }
110
```

```
/* Callback handler for normal-key event */
void keyboard(unsigned char key, int x, int y) {
   switch (key) {
      case 27:     // ESC key
         exit(0);
         break;
   }
}

/* Callback handler for special-key event */
void specialKeys(int key, int x, int y) {
   switch (key) {
      case GLUT_KEY_F1:    // F1: Toggle between full-screen and windowed mode
         fullScreenMode = !fullScreenMode;         // Toggle state
         if (fullScreenMode) {                     // Full-screen mode
            windowPosX   = glutGet(GLUT_WINDOW_X); // Save parameters for restoring later
            windowPosY   = glutGet(GLUT_WINDOW_Y);
            windowWidth  = glutGet(GLUT_WINDOW_WIDTH);
            windowHeight = glutGet(GLUT_WINDOW_HEIGHT);
            glutFullScreen();                      // Switch into full screen
         } else {                                            // Windowed mode
            glutReshapeWindow(windowWidth, windowHeight); // Switch into windowed mode
            glutPositionWindow(windowPosX, windowPosX);   // Position top-left corner
         }
         break;
      case GLUT_KEY_RIGHT:    // Right: increase x speed
         xSpeed *= 1.05f; break;
      case GLUT_KEY_LEFT:     // Left: decrease x speed
         xSpeed *= 0.95f; break;
      case GLUT_KEY_UP:       // Up: increase y speed
         ySpeed *= 1.05f; break;
      case GLUT_KEY_DOWN:     // Down: decrease y speed
         ySpeed *= 0.95f; break;
      case GLUT_KEY_PAGE_UP:  // Page-Up: increase ball's radius
         ballRadius *= 1.05f;
         ballXMin = clipAreaXLeft + ballRadius;
```

```
147             ballXMax = clipAreaXRight - ballRadius;
148             ballYMin = clipAreaYBottom + ballRadius;
149             ballYMax = clipAreaYTop - ballRadius;
150             break;
151          case GLUT_KEY_PAGE_DOWN: // Page-Down: decrease ball's radius
152             ballRadius *= 0.95f;
153             ballXMin = clipAreaXLeft + ballRadius;
154             ballXMax = clipAreaXRight - ballRadius;
155             ballYMin = clipAreaYBottom + ballRadius;
156             ballYMax = clipAreaYTop - ballRadius;
157             break;
158       }
159    }
160
161    /* Main function: GLUT runs as a console application starting at main() */
162    int main(int argc, char** argv) {
163       glutInit(&argc, argv);            // Initialize GLUT
164       glutInitDisplayMode(GLUT_DOUBLE); // Enable double buffered mode
165       glutInitWindowSize(windowWidth, windowHeight);  // Initial window width and height
166       glutInitWindowPosition(windowPosX, windowPosY); // Initial window top-left corner (x, y)
167       glutCreateWindow(title);      // Create window with given title
168       glutDisplayFunc(display);     // Register callback handler for window re-paint
169       glutReshapeFunc(reshape);     // Register callback handler for window re-shape
170       glutTimerFunc(0, Timer, 0);   // First timer call immediately
171       glutSpecialFunc(specialKeys); // Register callback handler for special-key event
172       glutKeyboardFunc(keyboard);   // Register callback handler for special-key event
173       glutFullScreen();             // Put into full screen
174       initGL();                     // Our own OpenGL initialization
175       glutMainLoop();               // Enter event-processing loop
176       return 0;
177    }
```

[TODO] Explanation

8. Handling Mouse Inputs with GLUT

Similarly, we can register callback function to handle mouse-click and mouse-motion.

- `glutMouseFunc`: registers callback handler for mouse click.

```
void glutMouseFunc(void (*func)(int button, int state, int x, int y)
   // (x, y) is the mouse-click location.
   // button: GLUT_LEFT_BUTTON, GLUT_RIGHT_BUTTON, GLUT_MIDDLE_BUTTON
   // state: GLUT_UP, GLUT_DOWN
```

- `glutMotionFunc`: registers callback handler for mouse motion (when the mouse is clicked and moved).

```
void glutMotionFunc(void (*func)(int x, int y)
   // where (x, y) is the mouse location in Window's coordinates
```

8.1 Example 10: Mouse-Controlled (GL10MouseControl.cpp)

For the bouncing ball program, the following mouse handler pause the movement with left-mouse click, and resume with right-mouse click.

```
1  /*
2   * GL10MouseControl.cpp: A mouse-controlled bouncing ball
3   */
4  #include <windows.h>  // for MS Windows
5  #include <GL/glut.h>  // GLUT, include glu.h and gl.h
6  #include <Math.h>     // Needed for sin, cos
7  #define PI 3.14159265f
8
9  // Global variables
10 char title[] = "Full-Screen & Windowed Mode";  // Windowed mode's title
11 int windowWidth  = 640;     // Windowed mode's width
12 int windowHeight = 480;     // Windowed mode's height
13 int windowPosX   = 50;      // Windowed mode's top-left corner x
14 int windowPosY   = 50;      // Windowed mode's top-left corner y
15
16 GLfloat ballRadius = 0.5f;   // Radius of the bouncing ball
17 GLfloat ballX = 0.0f;        // Ball's center (x, y) position
18 GLfloat ballY = 0.0f;
19 GLfloat ballXMax, ballXMin, ballYMax, ballYMin; // Ball's center (x, y) bounds
```

```
20  GLfloat xSpeed = 0.02f;      // Ball's speed in x and y directions
21  GLfloat ySpeed = 0.007f;
22  int refreshMillis = 30;      // Refresh period in milliseconds
23
24  // Projection clipping area
25  GLdouble clipAreaXLeft, clipAreaXRight, clipAreaYBottom, clipAreaYTop;
26
27  bool fullScreenMode = true; // Full-screen or windowed mode?
28  bool paused = false;         // Movement paused or resumed
29  GLfloat xSpeedSaved, ySpeedSaved;  // To support resume
30
31  /* Initialize OpenGL Graphics */
32  void initGL() {
33     glClearColor(0.0, 0.0, 0.0, 1.0); // Set background (clear) color to black
34  }
35
36  /* Callback handler for window re-paint event */
37  void display() {
38     glClear(GL_COLOR_BUFFER_BIT);  // Clear the color buffer
39     glMatrixMode(GL_MODELVIEW);    // To operate on the model-view matrix
40     glLoadIdentity();              // Reset model-view matrix
41
42     glTranslatef(ballX, ballY, 0.0f);  // Translate to (xPos, yPos)
43     // Use triangular segments to form a circle
44     glBegin(GL_TRIANGLE_FAN);
45        glColor3f(0.0f, 0.0f, 1.0f);  // Blue
46        glVertex2f(0.0f, 0.0f);       // Center of circle
47        int numSegments = 100;
48        GLfloat angle;
49        for (int i = 0; i <= numSegments; i++) { // Last vertex same as first vertex
50           angle = i * 2.0f * PI / numSegments;  // 360 deg for all segments
51           glVertex2f(cos(angle) * ballRadius, sin(angle) * ballRadius);
52        }
53     glEnd();
54
55     glutSwapBuffers();  // Swap front and back buffers (of double buffered mode)
```

```
56
57    // Animation Control - compute the location for the next refresh
58    ballX += xSpeed;
59    ballY += ySpeed;
60    // Check if the ball exceeds the edges
61    if (ballX > ballXMax) {
62       ballX = ballXMax;
63       xSpeed = -xSpeed;
64    } else if (ballX < ballXMin) {
65       ballX = ballXMin;
66       xSpeed = -xSpeed;
67    }
68    if (ballY > ballYMax) {
69       ballY = ballYMax;
70       ySpeed = -ySpeed;
71    } else if (ballY < ballYMin) {
72       ballY = ballYMin;
73       ySpeed = -ySpeed;
74    }
75 }
76
77 /* Call back when the windows is re-sized */
78 void reshape(GLsizei width, GLsizei height) {
79    // Compute aspect ratio of the new window
80    if (height == 0) height = 1;                // To prevent divide by 0
81    GLfloat aspect = (GLfloat)width / (GLfloat)height;
82
83    // Set the viewport to cover the new window
84    glViewport(0, 0, width, height);
85
86    // Set the aspect ratio of the clipping area to match the viewport
87    glMatrixMode(GL_PROJECTION);  // To operate on the Projection matrix
88    glLoadIdentity();             // Reset the projection matrix
89    if (width >= height) {
90       clipAreaXLeft   = -1.0 * aspect;
91       clipAreaXRight  = 1.0 * aspect;
```

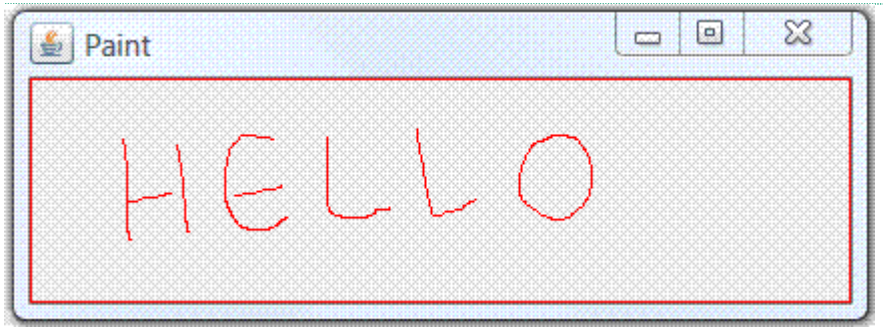
```
92          clipAreaYBottom = -1.0;
93          clipAreaYTop    = 1.0;
94       } else {
95          clipAreaXLeft   = -1.0;
96          clipAreaXRight  = 1.0;
97          clipAreaYBottom = -1.0 / aspect;
98          clipAreaYTop    = 1.0 / aspect;
99       }
100      gluOrtho2D(clipAreaXLeft, clipAreaXRight, clipAreaYBottom, clipAreaYTop);
101      ballXMin = clipAreaXLeft + ballRadius;
102      ballXMax = clipAreaXRight - ballRadius;
103      ballYMin = clipAreaYBottom + ballRadius;
104      ballYMax = clipAreaYTop - ballRadius;
105   }
106
107   /* Called back when the timer expired */
108   void Timer(int value) {
109      glutPostRedisplay();    // Post a paint request to activate display()
110      glutTimerFunc(refreshMillis, Timer, 0); // subsequent timer call at milliseconds
111   }
112
113   /* Callback handler for normal-key event */
114   void keyboard(unsigned char key, int x, int y) {
115      switch (key) {
116         case 27:     // ESC key
117            exit(0);
118            break;
119      }
120   }
121
122   /* Callback handler for special-key event */
123   void specialKeys(int key, int x, int y) {
124      switch (key) {
125         case GLUT_KEY_F1:    // F1: Toggle between full-screen and windowed mode
126            fullScreenMode = !fullScreenMode;         // Toggle state
127            if (fullScreenMode) {                     // Full-screen mode
```

```
128             windowPosX   = glutGet(GLUT_WINDOW_X); // Save parameters for restoring later
129             windowPosY   = glutGet(GLUT_WINDOW_Y);
130             windowWidth  = glutGet(GLUT_WINDOW_WIDTH);
131             windowHeight = glutGet(GLUT_WINDOW_HEIGHT);
132             glutFullScreen();                       // Switch into full screen
133          } else {                                          // Windowed mode
134             glutReshapeWindow(windowWidth, windowHeight); // Switch into windowed mode
135             glutPositionWindow(windowPosX, windowPosX);   // Position top-left corner
136          }
137          break;
138       case GLUT_KEY_RIGHT:    // Right: increase x speed
139          xSpeed *= 1.05f; break;
140       case GLUT_KEY_LEFT:     // Left: decrease x speed
141          xSpeed *= 0.95f; break;
142       case GLUT_KEY_UP:       // Up: increase y speed
143          ySpeed *= 1.05f; break;
144       case GLUT_KEY_DOWN:     // Down: decrease y speed
145          ySpeed *= 0.95f; break;
146       case GLUT_KEY_PAGE_UP:  // Page-Up: increase ball's radius
147          ballRadius *= 1.05f;
148          ballXMin = clipAreaXLeft + ballRadius;
149          ballXMax = clipAreaXRight - ballRadius;
150          ballYMin = clipAreaYBottom + ballRadius;
151          ballYMax = clipAreaYTop - ballRadius;
152          break;
153       case GLUT_KEY_PAGE_DOWN: // Page-Down: decrease ball's radius
154          ballRadius *= 0.95f;
155          ballXMin = clipAreaXLeft + ballRadius;
156          ballXMax = clipAreaXRight - ballRadius;
157          ballYMin = clipAreaYBottom + ballRadius;
158          ballYMax = clipAreaYTop - ballRadius;
159          break;
160    }
161 }
162
163 /* Callback handler for mouse event */
```

```
164 void mouse(int button, int state, int x, int y) {
165    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) { // Pause/resume
166       paused = !paused;          // Toggle state
167       if (paused) {
168          xSpeedSaved = xSpeed;  // Save parameters for restore later
169          ySpeedSaved = ySpeed;
170          xSpeed = 0;            // Stop movement
171          ySpeed = 0;
172       } else {
173          xSpeed = xSpeedSaved;  // Restore parameters
174          ySpeed = ySpeedSaved;
175       }
176    }
177 }
178
179 /* Main function: GLUT runs as a console application starting at main() */
180 int main(int argc, char** argv) {
181    glutInit(&argc, argv);            // Initialize GLUT
182    glutInitDisplayMode(GLUT_DOUBLE); // Enable double buffered mode
183    glutInitWindowSize(windowWidth, windowHeight);  // Initial window width and height
184    glutInitWindowPosition(windowPosX, windowPosY); // Initial window top-left corner (x, y)
185    glutCreateWindow(title);      // Create window with given title
186    glutDisplayFunc(display);     // Register callback handler for window re-paint
187    glutReshapeFunc(reshape);     // Register callback handler for window re-shape
188    glutTimerFunc(0, Timer, 0);   // First timer call immediately
189    glutSpecialFunc(specialKeys); // Register callback handler for special-key event
190    glutKeyboardFunc(keyboard);   // Register callback handler for special-key event
191    glutFullScreen();             // Put into full screen
192    glutMouseFunc(mouse);   // Register callback handler for mouse event
193    initGL();                     // Our own OpenGL initialization
194    glutMainLoop();               // Enter event-processing loop
195    return 0;
196 }
```

[TODO] Explanation

8.2 Example 11: A Simple Paint program



[TODO] Use mouse-motion and `GL_LINE_STRIP`.

Link to OpenGL/Computer Graphics References and Resources

Latest version tested: Eclipse CDT /MinGW
Last modified: July, 2012