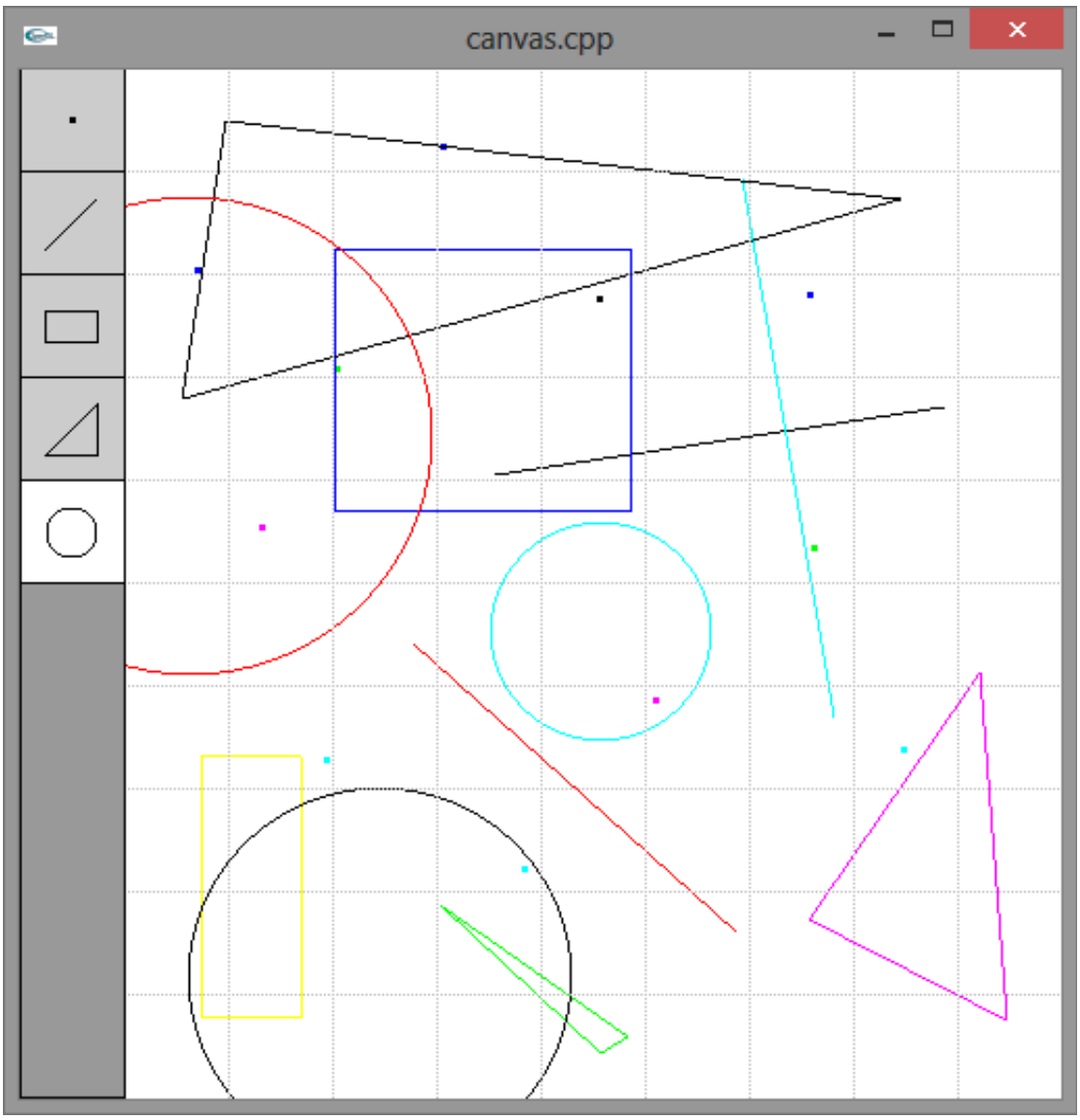
1. (20 pts) Code attached



2. Dr. Chandra Narayanaswami
   a. (1 pt) – Dr. W. Randolph Franklin
   b. (1 pt) – *Parallel Processing for Geometric Applications*

   c. (8 pts) – Two phase rendering for computer graphics (US Patent #6,177,944)

      This patent describes a technique by which the traditional geometry pipleline is split into two phases to improve overall graphics performance. The first phase computes the clipping status and immediately reports this before completing the actual clipping. The second phase performs the rest of the work. The advantage of this two phase method is that the wait time on the host processor is minimized, waiting only for the results of the clipping status from the first phase as opposed to waiting for the complete rendering.

```
////////////////////////////////////////////////////////////////////////////////
// canvas.cpp
//
// HW3 Solution - Problem 1
// Computer Graphics - Fall 2013
////////////////////////////////////////////////////////////////////////////////

#include <cstdlib>
#include <vector>
#include <iostream>
#include <cmath>

#ifdef __APPLE__
#  include <GLUT/glut.h>
#else
#  include <GL/glut.h>
#endif

using namespace std;

#define INACTIVE 0
#define POINT 1
#define LINE 2
#define RECTANGLE 3
#define TRIANGLE 4
#define CIRCLE 5
#define NUMBERPRIMITIVES 5

#define RED 0
#define YELLOW 1
#define GREEN 2
#define CYAN 3
#define BLUE 4
#define MAGENTA 5
#define BLACK 6

#define PI 3.14159

// Use the STL extension of C++.
using namespace std;

// Globals.
static GLsizei width, height; // OpenGL window size.
static float pointSize = 3.0; // Size of point
static int primitive = INACTIVE; // Current drawing primitive.
static int pointCount = 0; // Number of  specified points.
static int tempX, tempY, tempX2, tempY2; // Co-ordinates of clicked point.
static int isGrid = 1; // Is there grid?
static int color = BLACK; // The current draw color

// Set specified color
void setColor(int c)
{
   switch (c)
```

```
   {
   case RED:
      glColor3f(1.0,0.0,0.0);
      break;
   case YELLOW:
      glColor3f(1.0,1.0,0.0);
      break;
   case GREEN:
      glColor3f(0.0,1.0,0.0);
      break;
   case CYAN:
      glColor3f(0.0,1.0,1.0);
      break;
   case BLUE:
      glColor3f(0.0,0.0,1.0);
      break;
   case MAGENTA:
      glColor3f(1.0,0.0,1.0);
      break;
   case BLACK:
   default:
      glColor3f(0.0,0.0,0.0);
   }
}

// Point class.
class Point
{
public:
   Point(int xVal, int yVal, int cVal)
   {
         x = xVal; y = yVal, c = cVal;
   }
   void drawPoint(void); // Function to draw a point.
private:
   int x, y; // x and y co-ordinates of point.
   int c; // color
   static float size; // Size of point.
};

float Point::size = pointSize; // Set point size.

// Function to draw a point.
void Point::drawPoint()
{
   glPointSize(size);
   setColor(c);
   glBegin(GL_POINTS);
      glVertex3f(x, y, 0.0);
   glEnd();
}

// Vector of points.
vector<Point> points;
```

```
// Iterator to traverse a Point array.
vector<Point>::iterator pointsIterator;

// Function to draw all points in the points array.
void drawPoints(void)
{
   // Loop through the points array drawing each point.
   pointsIterator = points.begin();
   while(pointsIterator != points.end() )
   {
      pointsIterator->drawPoint();
         pointsIterator++;
   }
}

// Line class.
class Line
{
public:
   Line(int x1Val, int y1Val, int x2Val, int y2Val, int cVal)
   {
         x1 = x1Val; y1 = y1Val; x2 = x2Val; y2 = y2Val; c = cVal;
   }
   void drawLine();
private:
   int x1, y1, x2, y2, c; // x and y co-ordinates of endpoints.
};


// Function to draw a line.
void Line::drawLine()
{
   setColor(c);
   glBegin(GL_LINES);
      glVertex3f(x1, y1, 0.0);
      glVertex3f(x2, y2, 0.0);
   glEnd();
}

// Vector of lines.
vector<Line> lines;

// Iterator to traverse a Line array.
vector<Line>::iterator linesIterator;

// Function to draw all lines in the lines array.
void drawLines(void)
{
   // Loop through the lines array drawing each line.
   linesIterator = lines.begin();
   while(linesIterator != lines.end() )
   {
      linesIterator->drawLine();
```

```
        linesIterator++;
   }
}

// Rectangle class.
class Rectangle
{
public:
   Rectangle(int x1Val, int y1Val, int x2Val, int y2Val, int cVal)
   {
        x1 = x1Val; y1 = y1Val; x2 = x2Val; y2 = y2Val; c = cVal;
   }
   void drawRectangle();
private:
   int x1, y1, x2, y2, c; // x and y co-ordinates of diagonally opposite vertices.
};

// Function to draw a rectangle.
void Rectangle::drawRectangle()
{
   setColor(c);
   glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
   glRectf(x1, y1, x2, y2);
}

// Vector of rectangles.
vector<Rectangle> rectangles;

// Iterator to traverse a Rectangle array.
vector<Rectangle>::iterator rectanglesIterator;

// Function to draw all rectangles in the rectangles array.
void drawRectangles(void)
{
   // Loop through the rectangles array drawing each rectangle.
   rectanglesIterator = rectangles.begin();
   while(rectanglesIterator != rectangles.end() )
   {
      rectanglesIterator->drawRectangle();
        rectanglesIterator++;
   }
}

// Triangle class.
class Triangle
{
public:
   Triangle(int x1Val, int y1Val, int x2Val, int y2Val, int x3Val, int y3Val, int cVal)
   {
        x1 = x1Val; y1 = y1Val;
        x2 = x2Val; y2 = y2Val;
        x3 = x3Val; y3 = y3Val;
     c = cVal;
   }
```

```
   void drawTriangle();
private:
   int x1, y1, x2, y2, x3, y3, c; // x and y co-ordinates of vertices.
};


// Function to draw a triangle.
void Triangle::drawTriangle()
{
   setColor(c);
   glBegin(GL_TRIANGLES);
      glVertex3f(x1, y1, 0.0);
      glVertex3f(x2, y2, 0.0);
      glVertex3f(x3, y3, 0.0);
   glEnd();
}

// Vector of triangles.
vector<Triangle> triangles;

// Iterator to traverse a Triangle array.
vector<Triangle>::iterator trianglesIterator;

// Function to draw all triangles in the triangles array.
void drawTriangles(void)
{
   // Loop through the lines array drawing each line.
   trianglesIterator = triangles.begin();
   while(trianglesIterator != triangles.end() )
   {
      trianglesIterator->drawTriangle();
      trianglesIterator++;
   }
}

// Circle class.
class Circle
{
public:
   Circle(int x1Val, int y1Val, int x2Val, int y2Val, int cVal)
   {
         x1 = x1Val; y1 = y1Val; x2 = x2Val; y2 = y2Val, c = cVal;
   }
   void drawCircle();
private:
   int x1, y1, x2, y2, c; // x and y co-ordinates of endpoints.
};


// Function to draw a circle.
void Circle::drawCircle()
{
   setColor(c);
   double r = sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
```

```
   glBegin(GL_LINE_LOOP);
      for(int i=0; i < 100; i++) {
         double angle = 2*PI*i/100;
         double x = cos(angle);
         double y = sin(angle);
         glVertex3f(x1+x*r, y1+y*r, 0.0);
      }
   glEnd();
}

// Vector of circles.
vector<Circle> circles;

// Iterator to traverse a Circles array.
vector<Circle>::iterator circlesIterator;

// Function to draw all circles in the circles array.
void drawCircles(void)
{
   // Loop through the circles array drawing each circle.
   circlesIterator = circles.begin();
   while(circlesIterator != circles.end() )
   {
      circlesIterator->drawCircle();
         circlesIterator++;
   }
}

// Function to draw point selection box in left selection area.
void drawPointSelectionBox(void)
{
   if (primitive == POINT) glColor3f(1.0, 1.0, 1.0); // Highlight.
   else glColor3f(0.8, 0.8, 0.8); // No highlight.
   glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
   glRectf(0.0, 0.9*height, 0.1*width, height);

   // Draw black boundary.
   glColor3f(0.0, 0.0, 0.0);
   glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
   glRectf(0.0, 0.9*height, 0.1*width, height);

   // Draw point icon.
   glPointSize(pointSize);
   glColor3f(0.0, 0.0, 0.0);
   glBegin(GL_POINTS);
      glVertex3f(0.05*width, 0.95*height, 0.0);
   glEnd();
}

// Function to draw line selection box in left selection area.
void drawLineSelectionBox(void)
{
   if (primitive == LINE) glColor3f(1.0, 1.0, 1.0); // Highlight.
   else glColor3f(0.8, 0.8, 0.8); // No highlight.
```

```
   glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
   glRectf(0.0, 0.8*height, 0.1*width, 0.9*height);

   // Draw black boundary.
   glColor3f(0.0, 0.0, 0.0);
   glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
   glRectf(0.0, 0.8*height, 0.1*width, 0.9*height);

   // Draw line icon.
   glColor3f(0.0, 0.0, 0.0);
   glBegin(GL_LINES);
      glVertex3f(0.025*width, 0.825*height, 0.0);
      glVertex3f(0.075*width, 0.875*height, 0.0);
   glEnd();
}

// Function to draw rectangle selection box in left selection area.
void drawRectangleSelectionBox(void)
{
   if (primitive == RECTANGLE) glColor3f(1.0, 1.0, 1.0); // Highlight.
   else glColor3f(0.8, 0.8, 0.8); // No highlight.
   glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
   glRectf(0.0, 0.7*height, 0.1*width, 0.8*height);

   // Draw black boundary.
   glColor3f(0.0, 0.0, 0.0);
   glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
   glRectf(0.0, 0.7*height, 0.1*width, 0.8*height);

   // Draw rectangle icon.
   glColor3f(0.0, 0.0, 0.0);
   glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
   glRectf(0.025*width, 0.735*height, 0.075*width, 0.765*height);
   glEnd();
}

// Function to draw triangle selection box in left selection area.
void drawTriangleSelectionBox(void)
{
   if (primitive == TRIANGLE) glColor3f(1.0, 1.0, 1.0); // Highlight.
   else glColor3f(0.8, 0.8, 0.8); // No highlight.
   glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
   glRectf(0.0, 0.6*height, 0.1*width, 0.7*height);

   // Draw black boundary.
   glColor3f(0.0, 0.0, 0.0);
   glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
   glRectf(0.0, 0.6*height, 0.1*width, 0.7*height);

   // Draw line icon.
   glColor3f(0.0, 0.0, 0.0);
   glBegin(GL_LINES);
      glVertex3f(0.025*width, 0.825*height, 0.0);
      glVertex3f(0.075*width, 0.875*height, 0.0);
```

```
   glEnd();

   glBegin(GL_TRIANGLES);
      glVertex3f(0.025*width, 0.625*height, 0.0);
      glVertex3f(0.075*width, 0.625*height, 0.0);
         glVertex3f(0.075*width, 0.675*height, 0.0);
   glEnd();
}

// Function to draw circle selection box in left selection area.
void drawCircleSelectionBox(void)
{
   if (primitive == CIRCLE) glColor3f(1.0, 1.0, 1.0); // Highlight.
   else glColor3f(0.8, 0.8, 0.8); // No highlight.
   glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
   glRectf(0.0, 0.5*height, 0.1*width, 0.6*height);

   // Draw black boundary.
   glColor3f(0.0, 0.0, 0.0);
   glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
   glRectf(0.0, 0.5*height, 0.1*width, 0.6*height);

   // Draw line icon.
   glColor3f(0.0, 0.0, 0.0);
   glBegin(GL_LINE_LOOP);
      for(int i=0; i < 100; i++) {
         double angle = 2*PI*i/100;
         double x = cos(angle);
         double y = sin(angle);
         glVertex3f(0.05*width+x*12.0, 0.55*height+y*12.0, 0.0);
      }
   glEnd();
}

// Function to draw unused part of left selection area.
void drawInactiveArea(void)
{
   glColor3f(0.6, 0.6, 0.6);
   glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
   glRectf(0.0, 0.0, 0.1*width, (1 - NUMBERPRIMITIVES*0.1)*height);

   glColor3f(0.0, 0.0, 0.0);
   glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
   glRectf(0.0, 0.0, 0.1*width, (1 - NUMBERPRIMITIVES*0.1)*height);
}

// Function to draw temporary point.
void drawTempPoint(void)
{
   glColor3f(1.0, 0.0, 0.0);
   glPointSize(pointSize);
   glBegin(GL_POINTS);
      glVertex3f(tempX, tempY, 0.0);
      if (tempX != 0.0) glVertex3f(tempX2, tempY2, 0.0); // draw second point for triangle
```

```
   glEnd();
}

// Function to draw a grid.
void drawGrid(void)
{
   int i;

   glEnable(GL_LINE_STIPPLE);
   glLineStipple(1, 0x5555);
   glColor3f(0.75, 0.75, 0.75);

   glBegin(GL_LINES);
         for (i=2; i <=9; i++)
         {
         glVertex3f(i*0.1*width, 0.0, 0.0);
         glVertex3f(i*0.1*width, height, 0.0);
         }
         for (i=1; i <=9; i++)
         {
         glVertex3f(0.1*width, i*0.1*height, 0.0);
         glVertex3f(width, i*0.1*height, 0.0);
         }
   glEnd();
   glDisable(GL_LINE_STIPPLE);
}

// Drawing routine.
void drawScene(void)
{
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(0.0, 0.0, 0.0);

   drawPoints();
   drawLines();
   drawRectangles();
   drawTriangles();
   drawCircles();
   if ( ((primitive == LINE) || (primitive == RECTANGLE) || (primitive == CIRCLE)) &&
          (pointCount == 1) ) drawTempPoint();
   if ((primitive == TRIANGLE) && ((pointCount == 1) || (pointCount == 2))) drawTempPoint();
   if (isGrid) drawGrid();

   drawPointSelectionBox();
   drawLineSelectionBox();
   drawRectangleSelectionBox();
   drawTriangleSelectionBox();
   drawCircleSelectionBox();
   drawInactiveArea();

   glutSwapBuffers();
}

// Function to pick primitive if click is in left selection area.
```

```
void pickPrimitive(int y)
{
   if ( y < (1- NUMBERPRIMITIVES*0.1)*height ) primitive = INACTIVE;
   else if ( y < (1 - 4*0.1)*height ) primitive = CIRCLE;
   else if ( y < (1 - 3*0.1)*height ) primitive = TRIANGLE;
   else if ( y < (1 - 2*0.1)*height ) primitive = RECTANGLE;
   else if ( y < (1 - 1*0.1)*height ) primitive = LINE;
   else primitive = POINT;
}

// The mouse callback routine.
void mouseControl(int button, int state, int x, int y)
{
   if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
   {
      y = height - y; // Correct from mouse to OpenGL co-ordinates.

         // Click outside canvas - do nothing.
      if ( x < 0 || x > width || y < 0 || y > height ) ;

         // Click in left selection area.
      else if ( x < 0.1*width )
         {
            pickPrimitive(y);
               pointCount = 0;
         }

         // Click in canvas.
      else
         {
            if (primitive == POINT) points.push_back( Point(x,y,color) );
         else if (primitive == LINE)
               {
               if (pointCount == 0)
                     {
               tempX = x; tempY = y;
                     pointCount++;
                     }
               else
                     {
               lines.push_back( Line(tempX, tempY, x, y, color) );
                     pointCount = 0;
                     }
               }
         else if (primitive == RECTANGLE)
               {
               if (pointCount == 0)
                     {
               tempX = x; tempY = y;
                     pointCount++;
                     }
               else
                     {
               rectangles.push_back( Rectangle(tempX, tempY, x, y, color) );
```

```
                    pointCount = 0;
                    }
                }
        else if (primitive == TRIANGLE)
                {
                if (pointCount == 0)
                    {
                tempX = x; tempY = y;
                    pointCount++;
                    }
          else if (pointCount == 1)
          {
             tempX2 = tempX; tempY2 = tempY;
             tempX = x; tempY = y;
             pointCount++;
          }
              else
                    {
                triangles.push_back( Triangle(tempX2, tempY2, tempX, tempY, x, y, color) );
                    pointCount = 0;
            tempX2 = 0.0; // reset for drawTempPoint
                    }
                }
        else if (primitive == CIRCLE)
                {
                if (pointCount == 0)
                    {
                tempX = x; tempY = y;
                    pointCount++;
                    }
                else
                    {
                circles.push_back( Circle(tempX, tempY, x, y, color) );
                    pointCount = 0;
                    }
      }
        }
   }
   glutPostRedisplay();
}

// Initialization routine.
void setup(void)
{
   glClearColor(1.0, 1.0, 1.0, 0.0);
}

// OpenGL window reshape routine.
void resize(int w, int h)
{
   glViewport(0, 0, (GLsizei)w, (GLsizei)h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
```

```
   // Set viewing box dimensions equal to window dimensions.
   glOrtho(0.0, (float)w, 0.0, (float)h, -1.0, 1.0);

   // Pass the size of the OpenGL window to globals.
   width = w;
   height = h;

   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}

// Keyboard input processing routine.
void keyInput(unsigned char key, int x, int y)
{
   switch (key)
   {
      case 27:
         exit(0);
         break;
      default:
         break;
   }
}

// Clear the canvas and reset for fresh drawing.
void clearAll(void)
{
   points.clear();
   lines.clear();
   rectangles.clear();
   triangles.clear();
   circles.clear();
   primitive = INACTIVE;
   color = BLACK;
   pointCount = 0;
}

// The right button menu callback function.
void rightMenu(int id)
{
   if (id==1)
   {
         clearAll();
         glutPostRedisplay();
   }
   if (id==2) exit(0);
}

// The grid sub-menu callback function.
void grid_menu(int id)
{
   if (id==3) isGrid = 1;
   if (id==4) isGrid = 0;
   glutPostRedisplay();
```

```
}

// The color sub-medu callback function.
void color_menu(int id)
{
   if (id==5) color = RED;
   if (id==6) color = YELLOW;
   if (id==7) color = GREEN;
   if (id==8) color = CYAN;
   if (id==9) color = BLUE;
   if (id==10) color = MAGENTA;
   if (id==11) color = BLACK;
}

// Function to create menu.
void makeMenu(void)
{
   int grid_sub_menu;
   grid_sub_menu = glutCreateMenu(grid_menu);
   glutAddMenuEntry("On", 3);
   glutAddMenuEntry("Off",4);

   int color_sub_menu;
   color_sub_menu = glutCreateMenu(color_menu);
   glutAddMenuEntry("Red",5);
   glutAddMenuEntry("Yellow",6);
   glutAddMenuEntry("Green",7);
   glutAddMenuEntry("Cyan",8);
   glutAddMenuEntry("Blue",9);
   glutAddMenuEntry("Magenta",10);
   glutAddMenuEntry("Black",11);

   glutCreateMenu(rightMenu);
   glutAddSubMenu("Grid", grid_sub_menu);
   glutAddSubMenu("Color", color_sub_menu);
   glutAddMenuEntry("Clear",1);
   glutAddMenuEntry("Quit",2);
   glutAttachMenu(GLUT_RIGHT_BUTTON);
}

// Routine to output interaction instructions to the C++ window.
void printInteraction(void)
{
   cout << "Interaction:" << endl;
   cout << "Left click on a box on the left to select a primitive." << endl
        << "Then left click on the drawing area: once for point, twice for line or rectangle."
<< endl
        << "Right click for menu options." <<  endl;
}

// Main routine.
int main(int argc, char **argv)
{
   printInteraction();
```

```
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_DOUBLE);
   glutInitWindowSize(500, 500);
   glutInitWindowPosition(100, 100);
   glutCreateWindow("canvas.cpp");
   setup();
   glutDisplayFunc(drawScene);
   glutReshapeFunc(resize);
   glutKeyboardFunc(keyInput);
   glutMouseFunc(mouseControl);

   makeMenu(); // Create menu.

   glutMainLoop();

   return 0;
}
```