Programming Exercise 1:Interactive 2d drawing

Introduction

In this exercise, you will implement interactive 2d drawing and get to see it work. Before starting on this programming exercise, we strongly recommend watching the lectures and glut tutorial.

To get started with the exercise, you will need to download the starter codes. There are two cpp files:

opengl.cpp--which just draw a triangle staticly.

opengl_2d_sraw.cpp--which draw a line dynamically using mouse clicking and moving.

You could run thse two programs first,then you should modify the second file "opengl_2d_sraw.cpp" to add more functions such as draw many lines,circles and polygons. You may define your data structure and variables to store your data.

To run these programs ,the glut enviroment should be setup.You can get the glut from glut-3.7.6-bin.zip (117 KB)(http://user.xmission.com/~nate/glut/glut-3.7.6-bin.zip) ,unzip it and copy three files to different locations on your window as following:

- glut.h – This is the file you'll have to include in your source code. The common place to put this file is in the gl folder which should be inside the include folder of your system. for example ,your vc2010 include folder "*c:\program files\Microsoft Visual Studio 10.0\VC\include\gl*" or "*c:\program files(x86)\Microsoft Visual Studio 10.0\VC\include\gl*" in(my conputer
- glut32.lib (Windows version) – This file must be linked to your application so make sure to put it your lib folder. example ,your vc2010 lib folder "*c:\program files\Microsoft Visual Studio 10.0\VC\lib*" or "*c:\program files(x86)\Microsoft Visual Studio 10.0\VC\lib*" in(my conputer
- glut32.dll (Windows) – You could place the dll file in your exe's folder.for example,the sustem folder: "*c:\windows\system32*" or "*c:\windows\sysWOW64*" in my conputer

Interactive 2d drawing

In this part of this exercise,you will implement an interactive 2d draw program based on the starter code " opengl_2d_sraw.cpp " to draw different 2d figures such as line,polygon,circle which are curves or filled regions.

To help you start on this exercise, we provided all nessary starter code. You could run " opengl.cpp " to make you familiar with glut basic. It will show a window as figure 1:

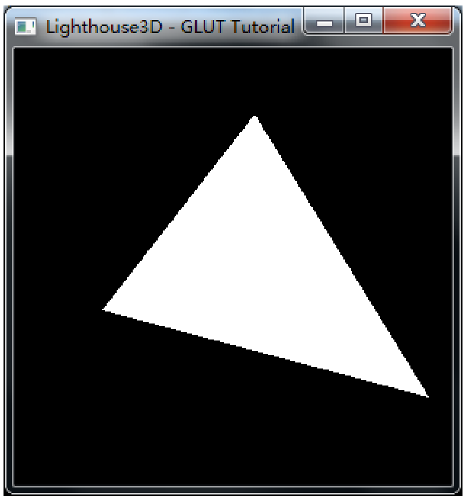

figure 1 a triangle in opengl

Then you can remove this " opengl.cpp " from your project and add " opengl_2d_sraw.cpp " to your project. In this code,first ,there is a main() function which inilize a window,then register some callback functions and finally enter into a mainloop to accept the user input and system events.

```
//---------------------------------------------------
int main(int argc, char **argv) {
        // init GLUT and create Window
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
        glutInitWindowPosition(100,100);
        glutInitWindowSize(view_width,view_height);
        glutCreateWindow("Lighthouse3D - GLUT Tutorial");


        glutDisplayFunc(renderScene);      // register dispaly callbacks
        glutReshapeFunc(resizeScene);      //register reshape callbacks
        glutMouseFunc(mouseClicked);     //register mouse click callbacks
        glutMotionFunc(mouseMotion);     //register mouse motion callbacks

        // enter GLUT event processing cycle
        glutMainLoop();

        return 1;
}
  //---------------------------------------------------
```

There are four callback functions are registered. The renderScene() is a registered display function to draw the scene.**The first task** of this exercise you should do is to draw a single line with a start point sP and an end point eP as the scene, you can copy the draw line code here into the renderScene. The final code should be like as following:

```
  //--------------- renderScene ---------------
```

```
   void renderScene(void) {
       glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

     // please fill the following code into the renderScene() function
       glBegin(GL_LINES);
       glVertex2f(sP[0],sP[1]);
       glVertex2f(eP[0],eP[1]);
       glEnd();

       glutSwapBuffers();
}
//----------------------------
```

The two points sP and eP is defined outside this function as global variables.

```
//--------------------------------------
     typedef float Point2D[2];
     Point2D sP = {0,0},eP = {30,50};
//--------------------------------------
```

The *glutReshapeFunc*() api of GLUT will be used to register a reshaping function which will be called to set the correct perspective when the window size or position changed! The code line " *glutReshapeFunc(resizeScene);*" to tell the glut when window changed, your *resizeScene*() will be responsible for setting the correct perspective.

*void glutReshapeFunc(void (*func)(int width, int height));*

Similarly,the mouse callback functions " mouseClicked" and " mousePassiveMotion"are registerd to be responsible for mouse clicking and mouse motion events using the following code:

```
glutMouseFunc(mouseClicked);                    //register mouse clicking callback
glutPassiveMotionFunc(mousePassiveMotion);    //register passive motion callback
```

There are three api functions in GLUT to register mouse callback functions as following:

```
void glutMouseFunc(void (*func)(int button, int state, int x, int y));
void glutMotionFunc(void (*func) (int x,int y));
void glutPassiveMotionFunc(void (*func) (int x, int y));
```

I implemented a simple rubble-line tool to visualize the line-drawing process.A golbal logistic variable *first* is used to distinguish the mouse clicked point is the start point or the end point of the to-be-drawing line. It is inilized to true:

```
bool first = true;
```

when the program starts,when a mouse clicking is detected, the screen position(x,y) is converted to a world coordinates (xf,yf). If the *first== true,* the point (xf,yf) is the start point of the line and the variable *first* is set to false, otherwise the point (xf,yf) is the end point of the line and the variable *first* is set to false so that we could start to draw a new line.

```
y = view_height-y;
```

```
float xf = (x-(float)view_width/2)/view_width*(right-left);
float yf = (y-(float)view_height/2)/view_height*(top-bottom);
if(first){
        sP[0] = xf; sP[1] = yf;
        first = false;
}
else{
        eP[0] = xf; eP[1] = yf;
        first = true;
        glutPostRedisplay();
}
```

To convert the screen coordinates in the viewport to world coordinates in the world window. We first to make the origin of the screen coordinates to the bottom-left corner,then we map it to world coordinates frames (see figure 2) .

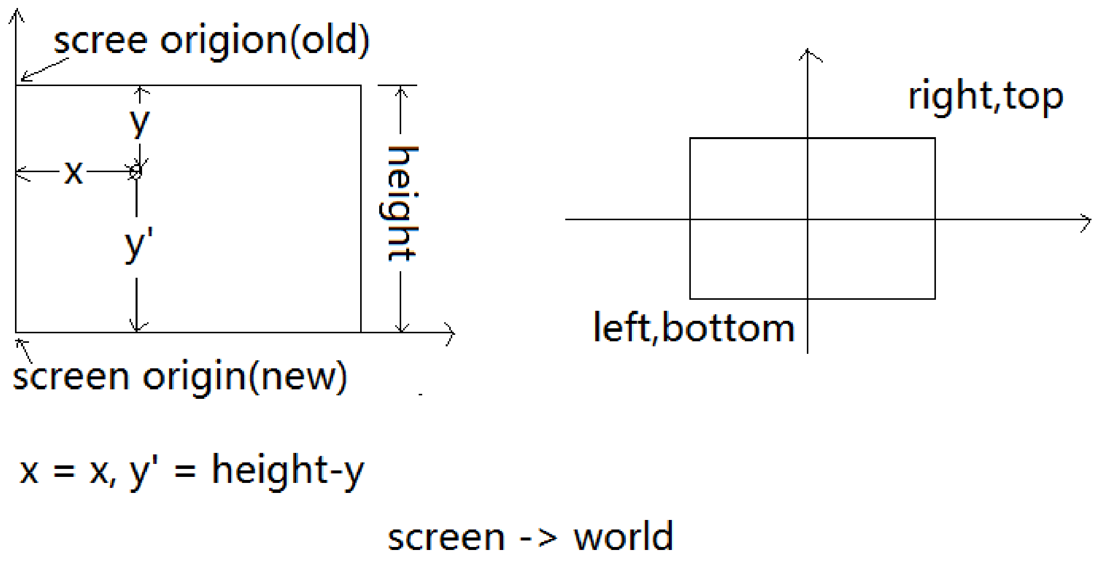

figure 2

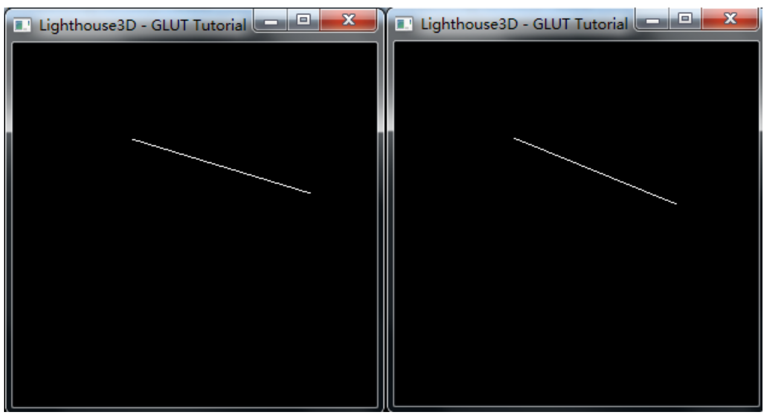

figure 3 Line moves following your mouse

After you finished the first task and run it, there will be a window and when you clicked your mouse,then you will see a line moves following your mouse until you clicked the mouse again,the line will determined.

Your second task is to draw any number of 2d curves and filled regions using rubble tools like subble-line tool above. To do this you should define your data structure to store those drawn entities such as lines,polygons and circles as you like.You may use the polymorphism in c++ language to define different entity classes such as Line2D or Polygon2D,... which are all derived from an base class Entity2D as following:

```
struct Entity2D{
        virtual void drawme();
   };
struct Line2D:public Entity2D{
     Point2D sP,eP;
     virtual void drawme(){
          glBegin(GL_LINES);
              glVertex2f(sP[0],sP[1]);
              glVertex2f(eP[0],eP[1]);
          glEnd();
     }
}
struct Polygon2D:public Entity2D{
     vector< Point2D >    points;
     virtual void drawme(){
          //....    please add your code to this polygon here
     }
}
```

You could use a C++ vector variable to store all the pointers of these entities as following:

```
   vector<Entity2D *> curvers.
```

Then you could modify the renderScene () to draw all these entities as following:

```
    for(int i = 0 ; i< curvers.szie();i++){
           curvers[i]->drawme();
    }
```

You may define different rubble tools as LineTool,PolygonTool,...which are all derived from a base class RubbleTool to help you to treat the mouse-drawing. Following is a sample code:

```
struct RubbleTool{
       int view_width,view_height;
      float left,right,bottom,top;
       RubbleTool(int w=400, int h = 300):view_width(w),view_height(h){}
      void set_window(float l,float r,float b,float t): left(l),right(r),bottom(b),top(t){}
      virtual void mouseClicked(int button, int state, int x, int y);
       virtual void glutMotionFunc(int x, int y);
       virtual void mousePassiveMotion(int x, int y);
  };
```

```
struct LineTool:public RubbleTool
 {
        Line2D    line;
        bool first;
        LineTool(){ first = true;}
        virtual void mouseClicked(int button, int state, int x, int y){
            y = view_height-y;
          float xf = (x-(float)view_width/2)/view_width*(right-left);
          float yf = (y-(float)view_height/2)/view_height*(top-bottom);
              if(first){
                   line.sP[0] = xf; line.sP[1] = yf;
                   first = false;
             }
             else{
                   line.eP[0] = xf; line. eP[1] = yf;
                   first = true;
             }
        }
        virtual void mousePassiveMotion(int x, int y){
              line.eP[0] = xf; line. eP[1] = yf;
         }
 }
```

To switch between different tools ,you may define a variable called drawing_mode to distinguish wheather you are drwaing a line,a polygon or other entity when you click or drag your mouse. You can switch bewteen different drawing mode using right-hand menu or key command. Suppose you register a keyboard callback to switch to a drawing mode. The keyboard callback register api is :

```
void glutKeyboardFunc(void (*func) (unsigned char key, int x, int y));
```

Here is an example of keyboard callback function which just exit the program when a user press "ESC" key of the keyboard.

```
void processNormalKeys(unsigned char key, int x, int y) {
  if (key == 27)
        exit(0);
}
```

Of course, there should a gobal variable pointing to the current Tool indicating your drawing mode. After that, the mouse events will transfer the current rubble Tool to treat these mouse events.

I provide a file " opengl_2d_tools.h " to help you start, you could make it more perfect as you like.

Hope you enjoy it and have a nice work!