Tutorial 5

At times we need to draw few transparent objects like glass, while others as opaque. In this tutorial we will learn about alpha compositing to achieve the above objective.

Alpha Compositing

To do alpha compositing OpenGL provides us with the function glBlendFunc(). But to use this function, we must first enable the compositing feature by notifying OpenGL to enable alpha compositing. To do so we call the glEnable() function with GL_BLEND as parameter.

```
//Enable Alpha Compositing
glEnable(GL_BLEND);
```

Another thing we need to specify to OpenGL, is the algorithm for alpha compositing. This is done by invoking glBlendFunc() function. The syntax for glBlendFunc() function is as follows:

void glBlendFunc(GLenum sfactor, GLenum dfactor);
Parameters:

- sfactor - Specifies how the red, green, blue, and alpha source blending factors are computed.
- dfactor - Specifies how the red, green, blue, and alpha destination (colour buffer) blending factors are computed.

In our code we are using GL_SRC_ALPHA as source factor and GL_ONE_MINUS_SRC_ALPHA as destination factor. This makes the OpenGL to draw the current object with it's $colour * \alpha$ and the current colour buffer (destination) with $colour * (1 - \alpha)$ and updates the current colour buffer. For more details please refer https://www.opengl.org/sdk/docs/man/html/glBlendFunc.xhtml.

Here in this tutorial we are drawing 4 different triangles with different depth and opacity (alpha value).

```
//{x,y,z,r,g,b,a}
float triangles[][7] = {
        //Triangle 1 colour=red alpha=0.65 depth=0.2
        {-0.5f, 0.0f, 0.2f, 1.0f, 0.0f, 0.0f, 0.65f},
        //Triangle 2 colour=yellow alpha=1(opaque) depth=0.1(farthest)
        { 0.0f, 0.0f, 0.1f, 1.0f, 1.0f, 0.0f, 1.0f},
        //Triangle 3 colour=light blue alpha=0.5 depth=0.3
        { 0.5f, 0.0f, 0.3f, 0.0f, 1.0f, 1.0f, 0.5f},
        //Triangle 4 colour=green alpha=0.25 depth=0.4 (nearest)
        { 0.0f,-0.5f, 0.4f, 0.0f, 1.0f, 0.0f, 0.25f},
};
```

Now try to run the code, and press the number keys 1,2,3 and 4 in any order, press 'r' to reset the screen. Vector draw_sequence stores the sequence in which we want the triangles to be drawn and the array notDrawn stores whether the triangle is already drawn or not. Pressing the keys viz. 1, 2, 3,and 4 sets the sequence in which the triangles will be drawn. Try drawing the triangles with different sequences. You will note that, even though we have set the correct alpha values, we are not getting the correct output for any sequence other than 2,1,3,4. Since this is the correct Z-order of drawing the triangles. This is because the destination is always set to current colour buffer and depth buffer only stores the z value of front most object for the given pixel. So we don't have enough information about the objects already drawn on the screen and thus we can't get the correct order to draw the objects. Hence it is necessary to draw the objects in their correct z-order, starting from farthest to nearest.

```
void processNormalKeys(unsigned char key, int x, int y) {
 switch(key) {
   case '1':
   //if triangle 1 is not already drawn then add it to
   //the sequence
       if(notDrawn[0]){
               draw_sequence.push_back(0);
               notDrawn[0]=false;
       }
     break;
   ...
  }
```

```
void processNormalKeys(unsigned char key, int x, int y) {
 switch(key) {
  ...
  //Resets the sequence
   case 'r':
       draw_sequence.clear();
       for(int i=0;i<4;i++) {
                notDrawn[i]=true;
       }
   break;

   //Sets the view to Orthographic mode
   case 'o':
       glMatrixMode(GL_PROJECTION);
       glLoadIdentity();
       glOrtho(-1.0, 1.0, -1.0, 1.0, -2.0, 2.0);
       glMatrixMode(GL_MODELVIEW);
       break;

   //Sets the view to Perspective mode
   case 'p':
       glMatrixMode(GL_PROJECTION);
       glLoadIdentity();
       gluPerspective(80.0, 1.0, 1.0, 20.0);
       glMatrixMode(GL_MODELVIEW);
       break;
   ...
 }
```

We have used a helper function drawTriangle to draw similar sized triangles at different positions and with different colours. It takes in x, y & z as offset from its original position and r, g, b & a as red, green, blue and alpha value as it's parameters.

```
void drawTriangle(float x,float y, float z,
                  float r,float g,float b,float a){
                  glBegin(GL_TRIANGLES);
                  glColor4f(r,g,b,a);
                  glVertex3f( 0.0f+x,  0.5f+y, 0.0f+z);
                  glVertex3f( 0.5f+x, -0.5f+y, 0.0f+z);
                  glVertex3f(-0.5f+x, -0.5f+y, 0.0f+z);
```

```
                glEnd();
}
```

Render function just draws the triangles in the sequence given by the vector draw_sequence.

```
void renderFunction(void) {
  //Clear the color and depth buffer
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  for(int i=0;i<draw_sequence.size();i++){
        int curr=draw_sequence[i]; //index of current triangle
        drawTriangle(triangles[curr][0],triangles[curr][1],
                     triangles[curr][2],triangles[curr][3],
                     triangles[curr][4],triangles[curr][5],
                     triangles[curr][6]);
  }

  // Swap front and back buffers
  glutSwapBuffers();
}
```