OpenGL|ES Tutorials (7): Blending & Text

Blending

Blending is a technique often used to merge two objects together. There are various types of blending. The important thing you have to remember with blending is that there are two parts to it. The Source Colour and the Destination Colour. The Source Colour is the underlying colour, and the destination colour is the one placed above. Normally, if no blending was used, you would only see the top colour. However, with blending there are various outcomes. The factors which decide what the final colour is depend on the properties of the blending that you are using. Below are the flags which you can use to for the blending factors.

Source Colour:

- GL_ZERO
- GL_ONE
- GL_SRC_COLOR
- GL_ONE_MINUS_SRC_COLOR
- GL_SRC_ALPHA
- GL_ONE_MINUS_SRC_ALPHA
- GL_DST_ALPHA
- GL_ONE_MINUS_DST_ALPHA

Destination Colour:

- GL_ZERO
- GL_ONE
- GL_DST_COLOR
- GL_ONE_MINUS_DST_COLOR
- GL_SRC_ALPHA_SATURATE
- GL_SRC_ALPHA
- GL_ONE_MINUS_SRC_ALPHA
- GL_DST_ALPHA
- GL_ONE_MINUS_DST_ALPHA

Text

As part of the framework, a function has been created for you which greatly simplifies writing text to the screen. This function is Framework2D_DrawText. It takes four parameters. The first is the text you wish to write to the screen, the second is the x-coordinate of where you want it on the screen, the third is the y-coordinate of where you want it on the screen, and the final parameter is the colour of the text in the form of an integer. Don't worry for now about converting colours to a single integer, as this will be discussed later on. To use this function, you will need to include the Framework2D header file, and two libraries: dxguid.lib and ddraw.lib

Introduction

The entire class is going to be re-written in this tutorial. However, only those lines which are new will be analysed. By now, you should be able to understand what the other lines do by yourself.

Declarations

```
1:    #include <windows.h>
2:    #include <GLES\gl.h>
3:    #include <GLES\egl.h>
4:    #include <KGSDK\framework.h>
5:    #include <KGSDK\frameworkGL.h>
6:    #include <KGSDK\framework2D.h>      //Header file for 2D operations
7:
8:    #pragma comment(lib, "Framework.lib")
9:    #pragma comment(lib, "libGLES_CM.lib")
10:   #pragma comment(lib, "dxguid.lib")  //Library required to draw text
11:   #pragma comment(lib, "ddraw.lib")   //Library required to draw text
12:
13:   static float rot[3];
14:   static float loc[3];
15:
16:   //Blending Factor Source
17:   GLenum blendSrc[] =
18:   {
19:       GL_ONE,
20:       GL_SRC_ALPHA
21:   };
22:
23:   //Blending Factor Destination
24:   GLenum blendDst[] =
25:   {
26:       GL_ZERO,
27:       GL_ONE,
28:       GL_DST_ALPHA,
29:       GL_ONE_MINUS_SRC_ALPHA
30:   };
31:
32:   int BlendSrcNumber = 0;       //The Current Blending Factor Source
33:   int BlendDstNumber = 0;       //The Current Blending Factor Destination
34:   bool TextDisplayed = false;   //Is there text on the screen?
35:
36:   bool RButton = false;         //Is Right Shoulder Button being held down?
37:   bool LButton = false;         //Is Left  Shoulder Button being held down?
38:
```

Analysis: A new header file is added on line 6. This header file is required so that we can draw text to the screen. In addition to this, an extra two libraries have been linked which are also required for drawing text. The next modifications to the declarations are the added enumerations on lines 16 – 30. The first is an enumeration for the blending source factor, and the second is an enumeration for the blending destination factor. Two integer variables are created on lines 32 and 33. One to keep record of which blending source factor we're using; and for which blending destination factor we're using. Line 34 declares a Boolean variable which will keep record of whether or not there is text displayed on the screen. Finally, the last two Boolean variables make sure that the Right Shoulder Button, and Left Shoulder button, are not counted if they are held down.

Init

```
1:      void Init()
2:      {
3:         glLoadIdentity();
4:
5:         GLfixed mat[4][4];
6:
7:         float nearz, farz;
8:
9:         farz  = 300.0f;
10:        nearz = 0.01f;
11:
12:        memset(mat, 0, sizeof(mat));
13:        mat[0][0] = (int) (65536.0f * 2.4f);
14:        mat[1][1] = (int) (65536.0f * 3.2f);
15:        mat[2][2] = (int) (65536.0f * (farz / (farz - nearz)));
16:        mat[2][3] = (int) (65536.0f * 1.0f);
17:        mat[3][2] = (int) (65536.0f * ((-farz * nearz) / (farz - nearz)));
18:
19:        glMatrixMode(GL_PROJECTION);
20:        glLoadMatrixx(&mat[0][0]);
21:
22:        rot[0] = 0;
23:        rot[1] = 0;
24:        rot[2] = 0;
25:        loc[0] = 0;
26:        loc[1] = 0;
27:        loc[2] = 3;
28:
29:        glEnable(GL_BLEND);              //Enable Blending
30:     }
```

Analysis: The only addition to the initialisation is line 29, where we enable blending. By now, you should be familiar with this process of enabling capabilities before using them.

Function: DrawStrips()

```
1:    void DrawStrips()
2:    {
3:          GLfloat TemplateStrip[] =
```

```
4:          {
5:          -0.1f, -0.3f, 0.0f,
6:          -0.1f,  0.3f, 0.0f,
7:           0.1f, -0.3f, 0.0f,
8:           0.1f,  0.3f, 0.0f
9:          };
10:
11:         glVertexPointer(3, GL_FLOAT, 0, TemplateStrip);
12:         glEnableClientState(GL_VERTEX_ARRAY);
13:
14:         glBlendFunc(blendSrc[BlendSrcNumber], blendDst[BlendDstNumber]);
15:         //Set Blending Factors (Properties)
16:
17:         //Top Left Cross
18:         glPushMatrix();
19:             glTranslatef(-0.6f, 0.5f, 0.0f);
20:             glColor4f(1.0f, 0.0f, 0.0f, 0.25f);   //Red
21:             glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
22:         glPopMatrix();
23:         glPushMatrix();
24:             glTranslatef(-0.6f, 0.5f, 0.0f);
25:             glRotatef(90, 0.0f, 0.0f, 1.0f);
26:             glColor4f(0.0f, 1.0f, 0.0f, 1.0f);    //Green
27:             glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
28:         glPopMatrix();
29:         //Top Right Cross
30:         glPushMatrix();
31:             glTranslatef(0.6f, 0.5f, 0.0f);
32:             glColor4f(0.0f, 1.0f, 0.0f, 0.5f);    //Green
33:             glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
34:         glPopMatrix();
35:         glPushMatrix();
36:             glTranslatef(0.6f, 0.5f, 0.0f);
37:             glRotatef(90, 0.0f, 0.0f, 1.0f);
38:             glColor4f(0.0f, 0.0f, 1.0f, 0.5f);    //Blue
39:             glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
40:         glPopMatrix();
41:         //Bottom Left Cross
42:         glPushMatrix();
43:             glTranslatef(-0.6f, -0.5f, 0.0f);
44:             glColor4f(0.0f, 0.0f, 1.0f, 0.5f);    //Blue
45:             glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
46:         glPopMatrix();
47:         glPushMatrix();
48:             glTranslatef(-0.6f, -0.5f, 0.0f);
49:             glRotatef(90, 0.0f, 0.0f, 1.0f);
50:             glColor4f(1.0f, 1.0f, 0.0f, 0.5f);    //Yellow
51:             glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
52:         glPopMatrix();
53:         //Bottom Right Cross
54:         glPushMatrix();
55:             glTranslatef(0.6f, -0.5f, 0.0f);
56:             glColor4f(1.0f, 1.0f, 0.0f, 0.5f);    //Yellow
57:             glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
58:         glPopMatrix();
59:         glPushMatrix();
60:             glTranslatef(0.6f, -0.5f, 0.0f);
61:             glRotatef(90, 0.0f, 0.0f, 1.0f);      //Green
62:             glColor4f(0.0f, 1.0f, 0.0f, 0.5f);
63:             glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
64:         glPopMatrix();
65:         //Center Cross
66:         glPushMatrix();
67:             glColor4f(1.0f, 0.0f, 0.0f, 0.25f);   //Red
68:             glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
```

```
69:         glPopMatrix();
70:         glPushMatrix();
71:             glRotatef(90, 0.0f, 0.0f, 1.0f);      //Blue
72:             glColor4f(0.0f, 0.0f, 1.0f, 0.75f);
73:             glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
74:         glPopMatrix();
75:    }
```

Analysis: We have replaced the DrawCube Function with a DrawStrips Function. This function demonstrates an important OpenGL|ES function: The glPushMatrix and the glPopMatrix function. glPushMatrix pushes the entire matrix on the stack, and the glPopMatrix does the opposite: it retrieves the matrix off the stack. You can imagine it as if a "checkpoint" is created when we use glPushMatrix. Then when we use glPopMatrix, we return back to that checkpoint (Hence, Matrix is reloaded). If you don't understand what these two functions do, don't worry as it will become evident later on.

Lines 3 – 9 define a vertex array for a simple rectangle. As with any vertex array, it is recommended you read through it, so that you know exactly what type of rectangle we are creating. Line 14 sets the Blending Factors. glBlendFunc is used for this, and takes two parameters. The first is the source blending factor, and the second is the destination blending factor. We retrieve these values from the arrays we created in the declarations section (blendSrc and blendDst).

Next we start drawing the actual strips on line 18 – 74. We are going to create 5 crosses (made from two rectangle strips – one vertical and one rotated 90 degrees so that it is horizontal (perpendicular) to the vertical one).

For every strip we draw, it is surrounded by a glPushMatrix and a glPopMatrix function call. This is why we only have to create one vertex array (TemplateStrip). The first strip (Top Left cross) we have to move the pen to the top left section of the screen. To do this we use the glTranslatef function. This takes three parameters which are values for how many units to move in the x-direction (left or right), in the y-direction (up or down) and the z-direction (in or out). For the top left cross, we translate 0.6 units left (note that it is minus 0.6: -0.6, if it was just 0.6 we would be moving towards the right.) and 0.5 units up. For this tutorial, we will never translate along the z-axis (third parameter of glTranslatef is always 0). Likewise, the other crosses also use the glTranslatef function to move the pen (for example, the bottom right cross translate 0.6 in the x-dir, and -0.5 in the y-dir). Only when we come to draw the centre cross we do not need to translate as the pen is already in the middle. We then set the pen colour to Red, and draw a rectangle. On line 22 the matrix is "popped" off the stack. This is effectively as if we returned to our "checkpoint" on line 18. (Before we translated the pen, or changed its colour). Also, notice that when we set the colour, we set different alpha values (Fourth parameter of glColorf), whereas before we always set this to 1.0f.

So now that we are back in the centre, we have to translate again back to the same position to draw the horizontal strip over it. We do exactly the same as we did in lines 18 – 22, but this time we also rotate 90 degrees before we draw the rectangle. This places this second rectangle perpendicular to the first.

We repeat this procedure of making a cross (lines 19 – 28) another four times, to produce a cross in all corners and the centre.

Render

```
1:      void Render()
2:      {
3:         glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
4:         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
5:
6:         DrawStrips();          //Draw Strips onto screen
7:
8:         glFinish();
9:         eglSwapBuffers(FrameworkGL_GetDisplay(), FrameworkGL_GetSurface());
10:     }
```

Analysis: Our render function remains unchanged. We have only updated it so that it now calls function DrawStrips instead of function DrawCube as it did previously.

Update

```
1:  void Update()
2:  {
3:  GLfixed mat[4][4];
4:
5:     memset(mat, 0, sizeof(mat));
6:     mat[0][0] = (int) (65536.0f *  cos(rot[0]));
7:     mat[0][2] = (int) (65536.0f *  sin(rot[0]));
8:     mat[1][1] = (int) (65536.0f *  cos(rot[1]));
9:     mat[1][2] = (int) (65536.0f *  sin(rot[1]));
10:    mat[2][0] = (int) (65536.0f * -sin(rot[0]) * cos(rot[1]));
11:    mat[2][2] = (int) (65536.0f *  cos(rot[0]) * cos(rot[1]));
12:    mat[3][2] = (int) (65536.0f * loc[2]);
13:    mat[3][3] = 65536;
14:
15:    glMatrixMode(GL_MODELVIEW);
16:    glLoadMatrixx(&mat[0][0]);
17:
18:    if (Framework_IsButtonPressed(FRAMEWORK_BUTTON_LEFT_SHOULDER ) &&
       !LButton)
19:    {
20:          Framework2D_Begin();           //Begin the 2D Framework
21:          BlendSrcNumber++;              //Increment the current Blending
             Source Factor
22:
23:          if (BlendSrcNumber > 1)        //If Blending Source Factor is above
             1, reset it
24:             BlendSrcNumber = 0;
25:
26:          switch (BlendSrcNumber)        //Write Blend Source Factor to Screen
```

```
27:         {
28:            case 0:
29:            Framework2D_DrawText(L"Source: GL_ONE", 110, 115, (RGB(255, 255,
               255)));
30:            break;
31:            case 1:
32:            Framework2D_DrawText(L"Source: GL_SRC_ALPHA", 100, 115,
               (RGB(255, 255, 255)));
33:            break;
34:         }
35:         Framework2D_End();              //End the 2D Framework
36:         LButton = true;                 //LButton being held down
37:         TextDisplayed = true;           //There is text displayed on screen
38:    }
39:    // If the Left Shoulder Button has been pressed: Toggle the Blending
       Source Factor
40:    // And write it onto the screen
41:
42:    if (Framework_IsButtonPressed(FRAMEWORK_BUTTON_RIGHT_SHOULDER ) &&
       !RButton)
43:    {
44:         Framework2D_Begin();            //Begin the 2D Framework
45:         BlendDstNumber++;               //Increment the current Blending
            Destination Factor
46:
47:         if (BlendDstNumber > 3)         //If Blending Destination Factor is
            above 3, reset it
48:            BlendDstNumber = 0;
49:
50:         switch (BlendDstNumber)         //Write Blend Destination Factor to
            Screen
51:         {
52:            case 0:
53:            Framework2D_DrawText(L"Destination: GL_ZERO", 108, 115,
               (RGB(255, 255, 255)));
54:            break;
55:            case 1:
56:            Framework2D_DrawText(L"Destination: GL_ONE", 110, 115, (RGB(255,
               255, 255)));
57:            break;
58:            case 2:
59:            Framework2D_DrawText(L"Destination: GL_DST_ALPHA", 50, 115,
               (RGB(255, 255, 255)));
60:            break;
61:            case 3:
62:            Framework2D_DrawText(L"Destination: GL_ONE_MINUS_SRC_ALPHA", 50,
               115, (RGB(255, 255, 255)));
63:            break;
64:         }
65:         Framework2D_End();              //End the 2D Framework
66:         RButton = true;                 //LButton being held down
67:         TextDisplayed = true;           //There is text displayed on screen
68:    }
69:    // If the Right Shoulder Button has been pressed: Toggle the Blending
       Destination Factor
70:    // And write it onto the screen
71:
72:
73:    if (Framework_IsButtonPressed(FRAMEWORK_BUTTON_FORWARD))
74:         TextDisplayed = false;
75:    // If the Forward Button has been pressed: Clear the text off the screen
76:    // And return to strips
77:
78:    if (!Framework_IsButtonPressed(FRAMEWORK_BUTTON_RIGHT_SHOULDER  ))
79:         RButton = false;
```

```
80:    // If the Right Shoulder Button is NOT being pressed, reset RButton
81:
82:    if (!Framework_IsButtonPressed(FRAMEWORK_BUTTON_LEFT_SHOULDER    ))
83:          LButton = false;
84:    // If the Left Shoulder Button is NOT being pressed, reset LButton
85: }
```

Analysis: Our Key monitoring functions have changed significantly. We now only keep an eye on the FORWARD, REWIND, LEFT DHOULDER and RIGHT SHOULDER keys. The rest have been omitted.

First up: the Left shoulder Button on lines 18 -38. Note that we now longer simply just check if the Left shoulder button is being pressed. We also check to see if it is being held down. Which is why we have added the "&& !LButton" part onto line 18. Line 20 "Begins" the 2D Framework. You must remember to start and stop the 2D Framework every time you wish to use it. Line 21 increments the BlendSrcNumber. This is main function of the Left Shoulder Button: to increment the blending source factor. Lines 23 and 24 make sure that if the BlendSrcNumber variable is incremented above two, it is reset as we only have two values in our blendSrc enumeration. Lines 26 to 34 check which blending source factor is being used, then writes it to the screen. A switch clause is used to determine which blending source factor is being used, and the Framework2D_DrawText function is used to display the text. As mentioned earlier in this tutorial the Framework2D_DrawText function has four parameters. The first is the Text your writing in wide characters, also known as Unicode (Hence the "L" prefix before the text, which converts the text to a wide-string literal format, for us). The next two variables are the x and y position of the text respectively. The final parameter is the colour of the text as an integer. Win32 has a special API which converts colours from a RGB format to an integer format for us. We simply type in RGB (red value, green value, blue value), and the RGB API will convert it into an integer. For example red which has the following integer value: 16711680, which can also be written as RGB(255, 0, 0). Finally, when the switch clause is finished we end the 2D Framework on line 35. Line 36 states that the Left shoulder button is being held down (LButton is set to rue). Finally we set TextDisplayed to true as well, since there is now text on the screen.

This entire is procedure is repeated again, but for the Right shoulder button on lines 42 - 68. And instead of working with the blending source factor, it performs operations concerning the blending destination factor.

On line 73 and 74, we check to see if the forward button has been pressed. If so, the text is cleared off the screen (variable TextDisplayed is set to false).

The last two checks we do are to see if the Left shoulder Button and Right shoulder button are _not_ being pressed. (Note the ! – exclamation mark – at the start of the if statements). If so, we set LButton and

RButton to false, respectively. This is part of the code to make sure if the buttons are being held down, they are not counted.

Main

```
1:      int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR
lpCmdLine, int nCmdShow )
2:      {
3:          if (!Framework_Init(hInstance, hPrevInstance))
4:          {
5:              return 0;
6:          }
7:
8:          if (!FrameworkGL_Init())
9:          {
10:             Framework_Close();
11:             return 0;
12:         }
13:
14:         //Initialise the 2D Framework.
15:         if (!Framework2D_Init())
16:         {
17:             Framework_Close();
18:             return 0;
19:         }
20:
21:         Init();
22:
23:         do
24:         {
25:             //If text is displayed on the screen, dont draw the strips
26:             if (!TextDisplayed)
27:             {
28:             Render();
29:             }
30:
31:             Update();
32:
33:             Framework_ProcessMessages();
34:
35:             if (!Framework_IsAppActive())
36:             {
37:                   while (!Framework_IsAppActive())
38:                   {
39:                         Framework_ProcessMessages();
40:                         Sleep(50);
41:                   }
42:             }
43:
44:         } while (!Framework_IsButtonPressed(FRAMEWORK_BUTTON_HOME));
45:
46:         //Clean Up
47:         FrameworkGL_Close();
48:         Framework2D_Close();
49:         Framework_Close();
50:
51:         return 0;
52:     }
```

Analysis: For the first time, we are changing our main function. We have made several changes. Firstly, the 2D Framework is initialised on line 15 -19, just as the GL Framework is initialised on lines 8 – 12. The

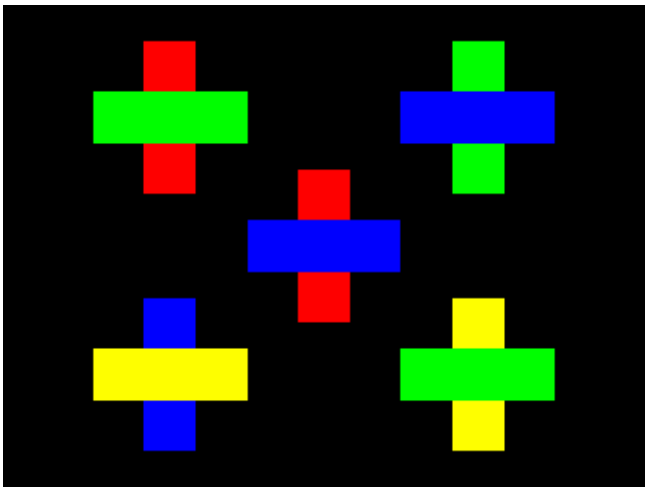
next change is that we no longer simply call our render function on every frame. First we check to see if text is displayed onto the screen. If so, we don't call the render function (i.e. the strips our not drawn), until the text has disappeared. Referring back to the update function, the only way to clear the text (set TextDisplayed to false) is to press the Forward Button. The rest of this function has remained unchanged except for the addition of line 28, which closes the 2D Framework.

Output

Although there are a vast combination of blending you can create using this tutorial, this Output analysis will only discuss a few. From this, you should be able to work out what the combinations will do.

Blending Source Factor: **GL_ONE**; Blending Destination Factor: **GL_ZERO**

This blending function only uses the source colour, and ignores the destination colour. You can reverse this and so that destination colour is used, and the source colour is ignored.

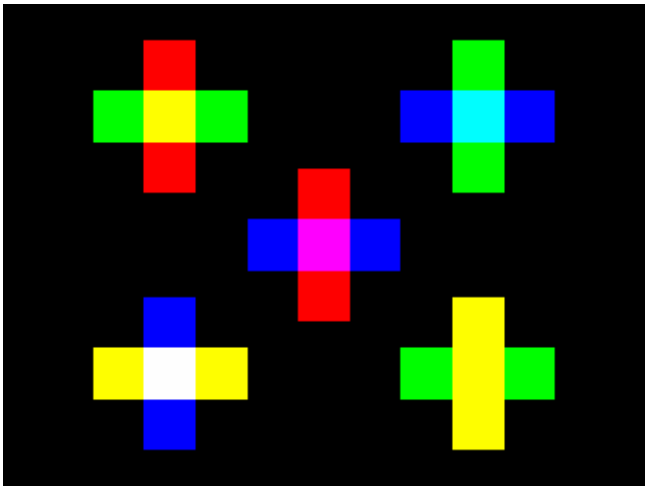


Blending Source Factor: **GL_ONE**; Blending Destination Factor: **GL_ONE**

This takes the two colours and adds them together. It takes each R, G, B value from each colour separately and adds it together. The maximum each value can have is 1, thus if the sum of the two values is greater than one, it is simply reduced back to one. For example:

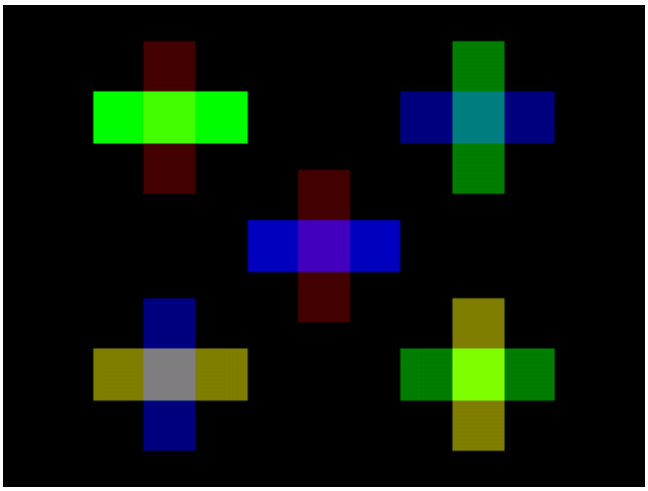
Red: 1.0, 0.0, 0.0 +	Yellow: 1.0, 1.0, 0.0 +	Blue: 0.0, 0.0, 1.0 +
Green: 0.0, 1.0, 0.0	Green: 0.0, 1.0, 0.0	Yellow: 1.0, 1.0, 0.0
---	---	---
Yellow: 1.0, 1.0, 0.0	Yellow: 1.0, 1.0, 0.0	White: 1.0, 1.0, 1.0

This is evident in our output from the code where the red mixing with the green produces yellow, the yellow mixing with the green produces yellow, and the blue mixing with the yellow produces white.



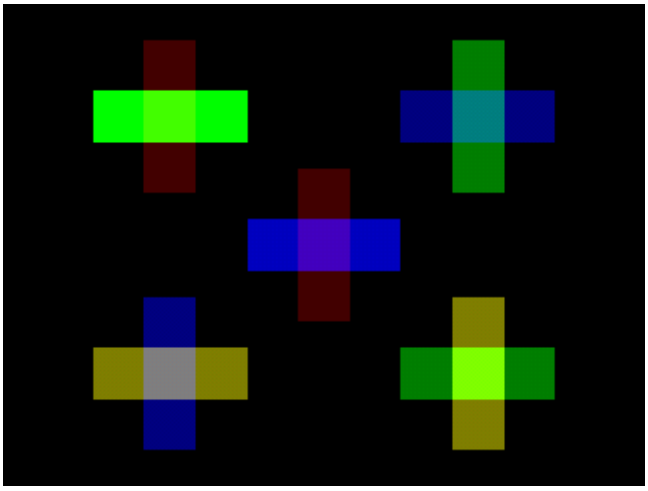
Blending Source Factor: **GL_SRC_ALPHA**; Blending Destination Factor: **GL_DST_ALPHA**

This blending function here has been a little improved. By using the GL_SRC_ALPHA and GL_DST_ALPHA flags, the alpha values of all the primitives have been taken into consideration. For example, look at the top left Green Vertical strip. It is more noticeable than the Red strip underneath it as it has an alpha values of 0.75f, whereas the red strip has an alpha value of 0.25f.



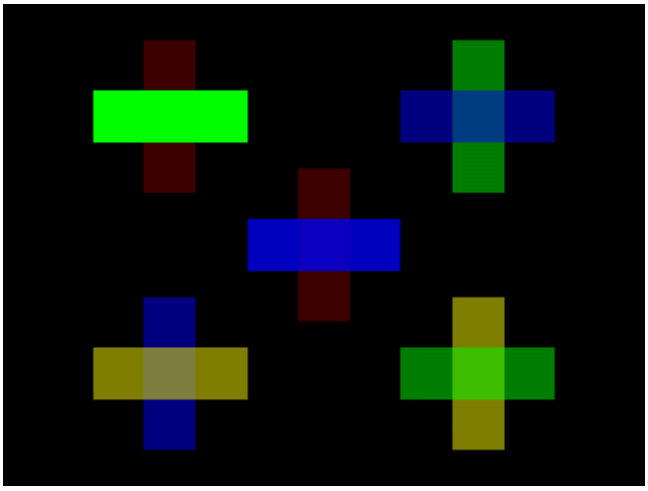
Blending Source Factor: **GL_SRC_ALPHA**; Blending Destination Factor: **GL_ONE**

This is similar to the previous blending function. It takes into consideration the Alpha value of the source, combined with only the destination values.



Blending Source Factor: **GL_SRC_ALPHA**; Blending Destination Factor: **GL_ONE_MINUS_SRC_ALPHA**

According to Khronos, transparency is best implemented using this function. Each strip, has the alpha value taken into consideration. Note that you can also use this blending function to render antialiased points and lines.



- End of OpenGL|ES Tutorials: Blending & Text -