OpenGL|ES Tutorials (8): Applying Mathematics (Sin/Cos/Pi)

Introduction

In this tutorial you will create a mesh, so that you can deform a Primitive. You will also apply a the Sine wave to flag, so that it creates a "waving" effect. Similarly, you can also use the Cosine Wave, to produce a waving effect.

Unlike the other tutorial, in this tutorial we will separate our drawing-related code from the rest. The non-relevant GL code will reside in a header file called "Functions.h"

File: Functions.h

Here we have added a few declarations, and one extra function to initialise our Flag. Part of the code that used to be in main.cpp has now also moved to here.

```
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:
7:  #pragma comment(lib, "Framework.lib")
8:  #pragma comment(lib, "libGLES_CM.lib")
9:
10: static float rot[3];
11: static float loc[3];
12:
13: float xrot = 0.0f;
14: float yrot = 0.0f;
15:
16: float lightAmbient[] = { 0.7f, 0.8f, 0.9f, 1.0f };
17: float lightDiffuse[] = { 0.7f, 0.8f, 0.9f, 1.0f };
18:
19: float matAmbient[] = { 0.6f, 0.6f, 0.6f, 1.0f };
20: float matDiffuse[] = { 0.6f, 0.6f, 0.6f, 1.0f };
21:
22: #define AMPLITUDE 2.0f                          //Amplitude of Wave
23: #define PIOVER180 0.01745329252                 //Simply PI divided by 180
24: #define PERIOD 20.0f                            //How many waves

25: GLuint texture[1];                              //Storage for one texture
26: float flagPoints[10][36][3];                    //Storage for Flag Colours
27: float flagCoords[10][36][2];                    //Storage for Flag Vertexes
28: float wrapValue;                                //Temporary storage for taking
the last value, and placing it at the beginning
29:
30: bool wireFrame = true;                          //Indicates whether wireframe
mode is selected
31:
```

```
32: //-----------------------------------------------------------------------
33: unsigned char *LoadBitmap(char *filename, BITMAPINFOHEADER *bmpInfo)
34: {
35:                 FILE *file;                     //Actual File
36:                 BITMAPFILEHEADER bmpFileHeader;             //Bitmap File
Header
37:                 unsigned char *bmpImage = NULL;             //Actual Bitmap
Image
38:                 unsigned char tmpRGB;           //Temporary storage for when
converting from BGR to RGB
39:
40:                 TCHAR path[256];                //Path
41:                 char fullPath[256];             //Path
42:
43:                 GetModuleFileName(NULL, path, 256);         //Get Current
Working Directory
44:                 TCHAR *pos = wcsrchr(path, '\\');           //Find last \\
so that we can remove the last part
45:                 *(pos + 1) = '\0';              //Add NULL Character to
position found, effectively removing last part
46:                 wcstombs(fullPath, path, 256); //Convert Wide characters to
multi-byte
47:                 strcat(fullPath, filename);     //Combine path with filename
48:
49:                 file = fopen(fullPath,"rb");    //Open File
50:                 if (!file)                      //Check validity of file
51:                 {
52:                     MessageBox(NULL, L"Can't Find Bitmap", L"Error", MB_OK);
53:                     return NULL;
54:                 }
55:
56:                 fread(&bmpFileHeader,sizeof(BITMAPFILEHEADER),1,file); //Read
header file into structure
57:
58:                 if (bmpFileHeader.bfType != 0x4D42)         //Check to see
if the file is a bitmap
59:                 {
60:                     MessageBox(NULL, L"Incorrect texture type", L"Error",
MB_OK);
61:                     fclose(file);
62:                     return NULL;
63:                 }
64:
65:                 fread(bmpInfo,sizeof(BITMAPINFOHEADER),1,file); //Read header
info into structure
66:
67:                 fseek(file,bmpFileHeader.bfOffBits,SEEK_SET); //Move to start
of image data
68:
69:                 bmpImage = new unsigned char[bmpInfo->biSizeImage];
70:
71:                 if (!bmpImage)                  //Memory allocated for image
data
72:                 {
73:                     MessageBox(NULL, L"Out of Memory", L"Error", MB_OK);
74:                     delete[] bmpImage;
75:                     fclose(file);
76:                     return NULL;
77:                 }
78:
79:                 fread(bmpImage,1,bmpInfo->biSizeImage,file);//Load image
data
80:
81:                 if (!bmpImage)
82:                 {
83:                     MessageBox(NULL, L"Error reading bitmap", L"Error",
MB_OK);
```

```
84:                     fclose(file);
85:                     return NULL;
86:                 }
87:
88:                 for (unsigned int i = 0; i < bmpInfo->biSizeImage; i+=3)
    //Convert BGR -> RGB
89:                 {
90:                     tmpRGB = bmpImage[i];
91:                     bmpImage[i] = bmpImage[i+2];
92:                     bmpImage[i+2] = tmpRGB;
93:                 }
94:
95:                 fclose(file);                   //Close File
96:
97:                 return bmpImage;                //Return Bitmap Image
98: }
99:
100://--------------------------------------------------------------------
101:                bool LoadTextures()
102:                {
103:                BITMAPINFOHEADER info;          //Structure to hold bitmap
data
104:                unsigned char *bitmap = NULL;  //Pointer to point to image
data
105:
106:                bitmap = LoadBitmap("GFlag.bmp", &info);     //Load Bitmap
107:
108:                if (!bitmap)
109:                    return false;
110:
111:                glGenTextures(1, texture);     //Generate Texture Identifiers
112:
113:                glBindTexture(GL_TEXTURE_2D, texture[0]);    //Select
texture[0]
114:
115:                glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, info.biWidth,    //Set
Texture Properties
116:                    info.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE,
117:                    bitmap);
118:

119:                glTexParameterf(GL_TEXTURE_2D, //Set further texture
properties
120:                    GL_TEXTURE_MIN_FILTER, GL_LINEAR);
121:                glTexParameterf(GL_TEXTURE_2D,
122:                    GL_TEXTURE_MAG_FILTER, GL_LINEAR);
123:
124:                delete[] bitmap;                                    //Clean Up
125:

126:                return true;
127:                }
128://--------------------------------------------------------------------
```

Analysis: This is all the content that is in the file: Functions.h. Line 22 is the first addition; which is a pre-processor directive to replace AMPLTUDE with the value 2.0f. Line 23 is another pre-processor directive. It simply holds the value of Pi divided by 180. (You can even type this into a calculator). The last #define statement, on line 24, replaces the word PERIOD with 20.0f. A multi-dimensional array, of type float, is defined on line 26; this will hold the flag colour co-ordinates. Similarly, line 27 creates an array to hold the flag vertices. The first value of the arrays is the number of rows (10), next it is the number of columns (18)

– this value needs to be doubled because we will be drawing the strips in anti-clockwise order later on (duplicate vertices). Finally the last value is the either X, Y or Z value of that particular point. Don't worry if you don't understand this yet, as it will become more clear later on. In previous tutorials we simply created a rectangle, consisting of four vertices. In this tutorial, we will deform the flag by creating a mesh. Line 28 is another float variable, which we will use later on to take the end value of the flag, and place it at the beginning. The rest of this file contains the functions LoadTextures and LoadBitmap, which has already been discussed in a previous tutorial.

File: main.cpp

This file contains the rest of the code, which is only relevant to drawing etc.

Declarations

```
1:      #include "Functions.h"    // Non-relevant drawing code in this header
```

Analysis: All the declarations have been moved to the Functions.h Header File, thus all we need to do now is just link to that.

Function: Flag_Init()

```
1:      void Flag_Init()
2:      {
3:         int x;     //Current X Cell Index
4:         int y;     //Current Y Cell Index
5:         int apos;  //Current X Position
6:
7:         for (y = 0; y < 10; y++) //Loop through each Row
8:         {
9:              for (x = 0, apos = 0; x < 18; x++, apos += 2)  //Loop through
                each Column
10:             {
11:                   flagPoints[y][apos][0] = float(x);      //X value
12:                   flagPoints[y][apos][1] = float(y + 1); //1 above the Y
                      value
13:                   flagPoints[y][apos][2] = float(sin(x * PERIOD *
                      PIOVER180)) * AMPLITUDE;
14:                   // Z value generated by sin wave.
15:
16:                   flagCoords[y][apos][0] = x * 1.0f / 18.0f;  //X Texcoord
17:                   flagCoords[y][apos][1] = (y + 1) * 1.0f / 10.0f;  //Y
                      TexCoord
18:                   //Specify Texture co-ordinates for current cell
19:
20:                   //Next section of code is a repeat of prevoius block,
                      except
21:                   //that it operates on the next apos value. Y value is
22:                   //also different in the section below
23:                   flagPoints[y][apos + 1][0] = float(x);
24:                   flagPoints[y][apos + 1][1] = float(y);
25:                   flagPoints[y][apos + 1][2] = float(sin(x * PERIOD *
                      PIOVER180)) * AMPLITUDE;
26:
27:
28:                   flagCoords[y][apos + 1][0] = x * 1.0f / 18.0f;
29:                   flagCoords[y][apos + 1][1] = y * 1.0f / 10.0f;
30:             }
31:        }
```

```
32:     }
```

Analysis: An extra function is defined here, to initialise the Flag. The first three variables defined are the current X cell index, current Y cell index, and the current X Position, respectively. These variables are merely just to help us loop through each cell. To loop through each cell, we need to create a double for loop. Line 7 loops through each row, and line 9 loops through each column.

Line 11 sets the X Value, to the current value of the variable x. Line 12 does the same for the Y value, but it places it one value higher (i.e. y + 1). The Z value is a little more complex. To generate the Z value, we use the sine curve. Line 23 – 29 does exactly the same as lines 11 – 17, but for the next apos value.

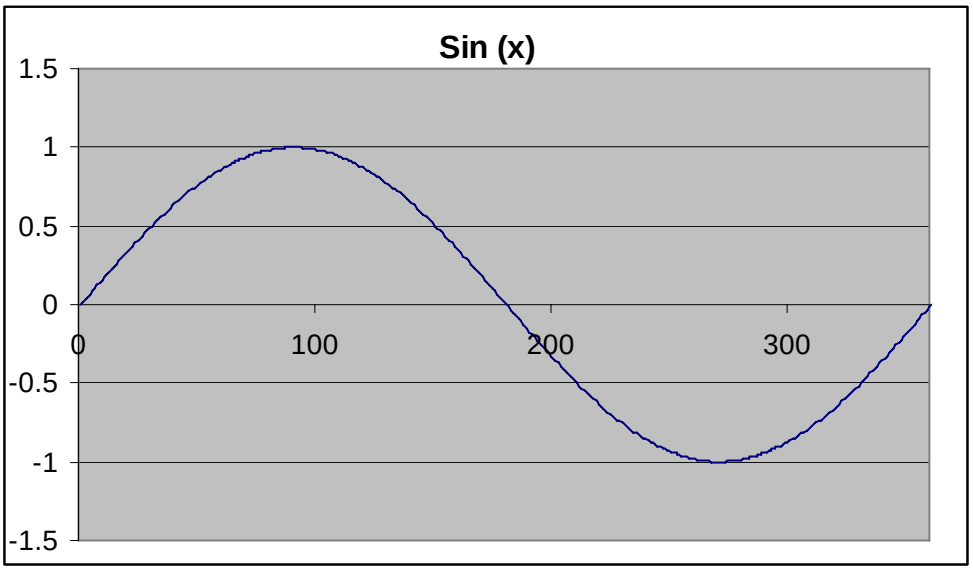

Figure 8.1 Sine Curve

Figure 8.1 is the graph of y = sin(x). This is what the flag will look like from above. For the Z value, we first multiply the x value, by 20, since there are 10 columns (PERIOD). This is because the period of one wave is 360 units (refer to figure 8.1). If we double this, we would have four waves in our flag. We then multiply by the value PIOVER180, simply to convert Radians to Degrees.

Line 16 and 17 relate to the texture. Line 16 sets the X texture co-ordinate. Since there are 18 columns, we need to divide the image into 18 columns, hence 1.0 / 18.0. This is then multiplied by the x value so we get the right segment. Line 17 does the same, but for the Y texture co-ordinate instead.

Figure 8.2 is how our mesh will look.

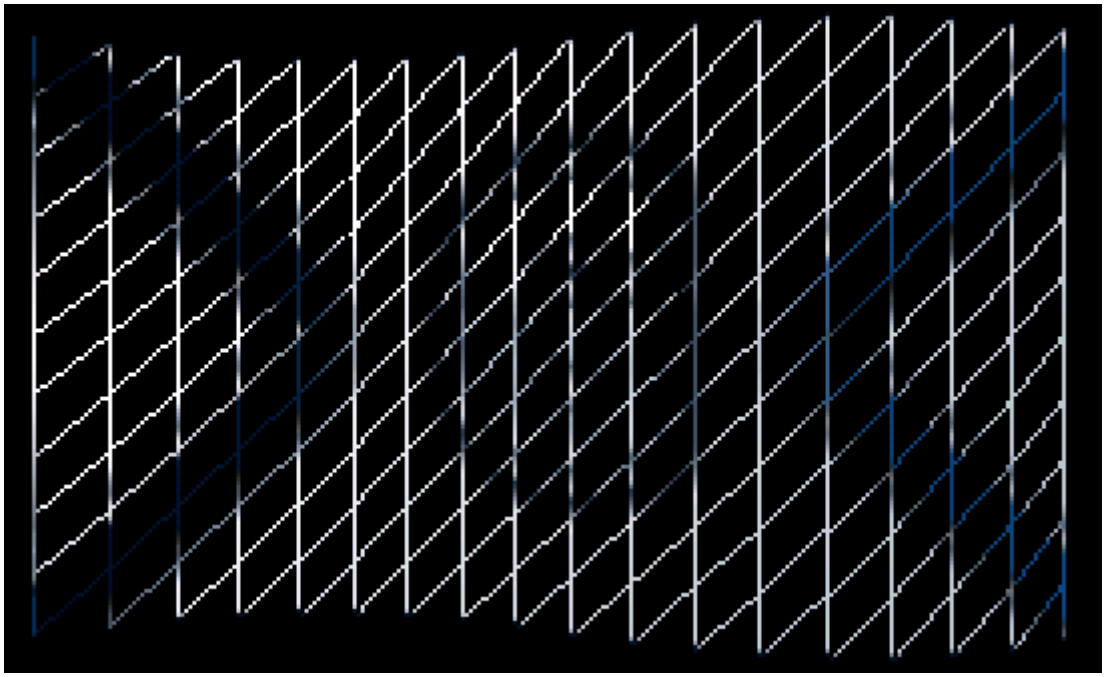

Figure 8.2 Flag Mesh

Init

```
1:    void Init()
2:    {
3:       glEnable(GL_LIGHTING);           //Enable Lighting
4:       glEnable(GL_LIGHT0);             //Enable GL_LIGHT0
5:       glEnable(GL_COLOR_MATERIAL);     //Enable GL_LIGHT0
6:       glDisable(GL_CULL_FACE);         //Disable Back face culling
7:
8:       glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, matAmbient);
9:       glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, matDiffuse);
10:
11:      glLightfv(GL_LIGHT0, GL_AMBIENT, lightAmbient);
12:      glLightfv(GL_LIGHT0, GL_DIFFUSE, lightDiffuse);
13:
14:      if (!LoadTextures())
15:      {
16:           MessageBox(NULL, L"Error loading textures", L"Error", MB_OK);
17:      }
18:      else
19:      {
20:           glEnable(GL_TEXTURE_2D);
21:      }
22:
23:      Flag_Init();                     //Intialise Flag
24:
25:      glLoadIdentity();
26:
27:      GLfixed mat[4][4];
28:
29:      float nearz, farz;
30:
31:      farz  = 300.0f;
32:      nearz = 0.01f;
33:
34:      memset(mat, 0, sizeof(mat));
35:      mat[0][0] = (int) (65536.0f * 2.4f);
36:      mat[1][1] = (int) (65536.0f * 3.2f);
37:      mat[2][2] = (int) (65536.0f * (farz / (farz - nearz)));
38:      mat[2][3] = (int) (65536.0f * 1.0f);
39:      mat[3][2] = (int) (65536.0f * ((-farz * nearz) / (farz - nearz)));
40:
41:      glMatrixMode(GL_PROJECTION);
42:      glLoadMatrixx(&mat[0][0]);
43:
44:      rot[0] = 0;
45:      rot[1] = 0;
46:      rot[2] = 0;
47:      loc[0] = 0;
48:      loc[1] = 0;
49:      loc[2] = 3;
50:
51:      glVertexPointer(3, GL_FLOAT, 0, flagPoints);
52:      glTexCoordPointer(2, GL_FLOAT, 0, flagCoords);
53:      glEnableClientState(GL_VERTEX_ARRAY);
54:      glEnableClientState(GL_TEXTURE_COORD_ARRAY);
55:   }
```

Analysis: The only new addition to this function is a call to the Flag_Init, which is where all our Flag Initialisation is done. We have also disabled Back Face Culling on line 6, as if it was enabled, we would only see one of the faces of the flag. I have also moved lines 51 – 54 from the Render function, to this function.

This is because it is not necessary to Enable Vertex/Texture Arrays, or point to the arrays in every frame. Thus by doing this once, in just the init function, the program will run faster.

Function: DrawFlag();

```
1:   void DrawFlag()
2:   {
3:      glRotatef(xrot,1.0f,0.0f,0.0f);    //Rotate axis based on value of xrot
4:      glRotatef(yrot,0.0f,1.0f,0.0f);    //Rotate axis based on value of yrot
5:
6:
7:      glPushMatrix();
8:
9:          glScalef(0.07f, 0.07f, 0.07f);      //Scale Down the Flag
10:         glTranslatef(-10.0f, -5.0f, 0.0f);
            //Translate the flag to the center of the screen
11:
12:         for (int i = 0; i < 10; i++)
13:             glDrawArrays(wireFrame ? GL_LINE_STRIP : GL_TRIANGLE_STRIP, i *
                36, 36);
14:
15:         //Draw the flag onto the screen in regard to wireframe status
16:     glPopMatrix();
17:  }
```

Analysis: Lines 3 and 4 rotate the scene, as we have done in previous tutorials. Next we push the matrix on to the stack. The first thing we do is scale everything down. To do this, the glScalef function is used. This function takes three parameters, which are values for how much to scale the scene on the X, Y, and Z planes respectively. In this tutorial we scale everything down by 0.07%, in all directions (Hence 0.07f for all three parameters). Next we translate (move) the flag. This is simply to the centre the flag in the middle of the screen. On line 12 we begin to draw the flag; we do this by creating a for loop, and then looping through each row. For every row we draw the strip (Each strip is defined every 36 elements). We also draw the flag depending on the wireframe variable. If the wireframe variable is true, we draw the flag using lines, and if it is false, we render it using Triangle Strips.

Output

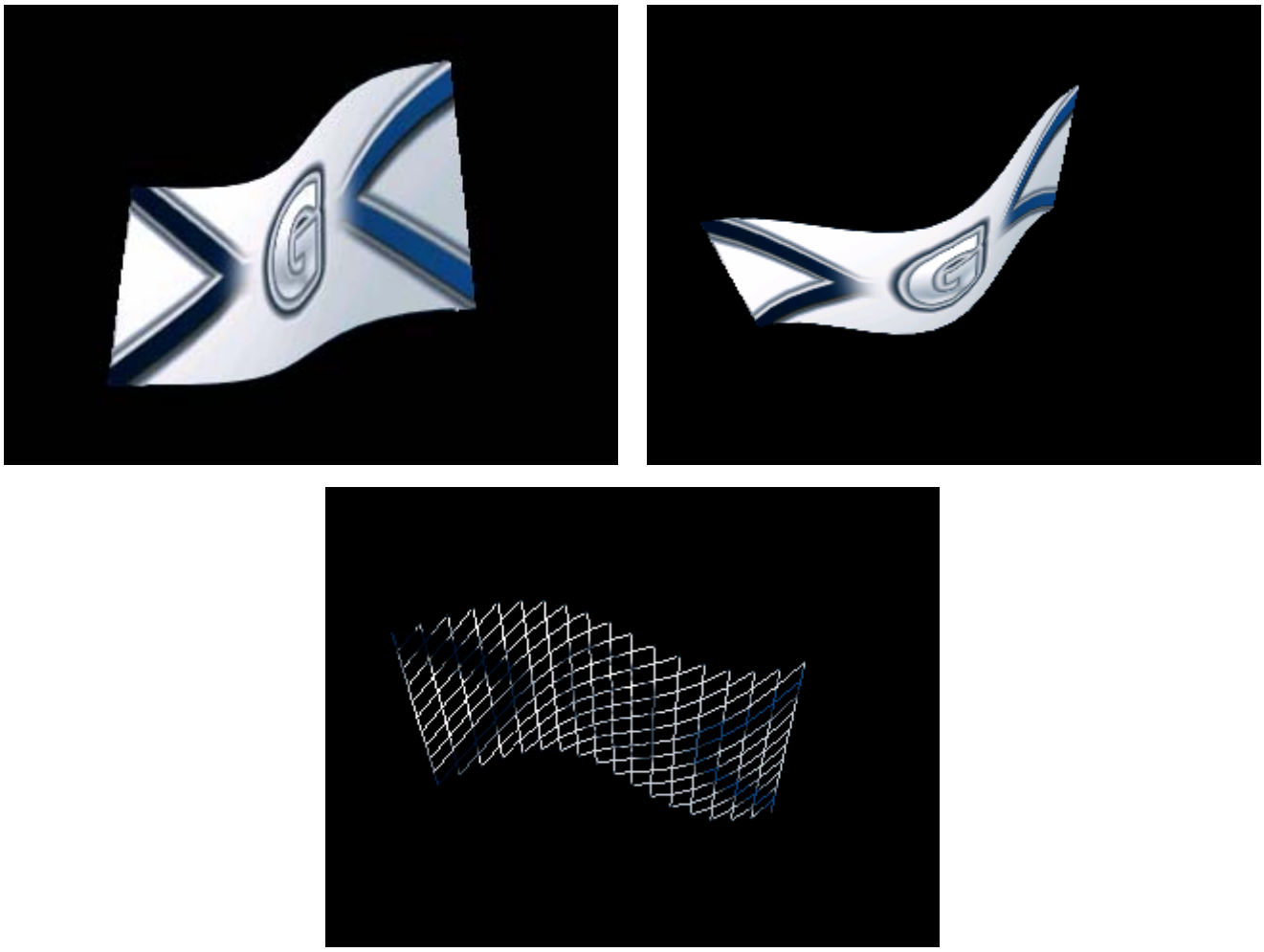

Figure 8.3 Tutorial 8 Output

As you can see, Figure 8.3 shows the output for this tutorial. The first two images show the flag rendered using GL_TRIANGLE_STRIPS, and the bottom image shows the flag rendered using GL_LINES. The wire frame status can be toggled using the REWIND button. If you execute this code on a Gizmondo Device, you can see for yourself how the FORWARD button has been implemented to move the sine curve through the Flag.

- End of OpenGL|ES Tutorials: Applying Mathematics (Sin/Cos/Pi) -