INTRODUCCIÓN A OPENGL

Computación Gráfica

Introducción a OpenGL

- OpenGL
- GLUT - registrar funciones de callback (freeGLUT o glut)
  - código asociado a “eventos” en las ventanas.
  - sirven para administrar lo que se dibuja.
- GLEW

S.J. Gortler. Foundations of 3D Computer Graphics. MIT Press. 2012.

Librerías a incluir

```
#include <GL/glew.h>
#ifdef __APPLE__
#include <GLUT/freeglut.h>
#else
#include <GL/freeglut.h> // or glut.h
#endif
```

S.J. Gortler. Foundations of 3D Computer Graphics. MIT Press. 2012.

Programa principal

```
int main()
{
    initGlutState();
    glewInit();
    initGLState();
    initShaders();
    initVBOs();
    glutMainLoop();
    return 0;
}
```

S.J. Gortler. Foundations of 3D Computer Graphics. MIT Press. 2012.

Inicializando el estado de GLUT y OpenGL

```
/* ------------------------------------------------------------
 * initGlutState()
 * request a window of size g_with, g_height
 * register callback functions display and reshape
 * ------------------------------------------------------------ */
void initGlutState()
{
    glutInit();
    glutInitDisplayMode( GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH );
    glutInitWindowsSize( g_width, g_height );
    glutCreateWindow("Hello World");
    glutDisplayFunc( display );
    glutReshapeFunc( reshape );
}
```

S.J. Gortler. Foundations of 3D Computer Graphics. MIT Press. 2012.

Reshape

```
void reshape( int w, int h )
{
    g_width = w;
    g_height = h;
    glViewport( 0, 0, w, h );
    glutPostRedisplay();
}
```

Parameters

x, y	Especifica la esquina izquierda inferior del viewport, en pixels. El valor inicial es (0,0).
width, height	Especifica el ancho y alto del viewport.

Las coordenadas de la ventana (x_w, y_w) se calculan:

$$x_w = (x_{nd} + 1)\left(\frac{width}{2}\right) + x$$

$$y_w = (y_{nd} + 1)\left(\frac{height}{2}\right) + y$$

(x_{nd}, y_{nd})	Coordenadas normalizadas del device.

https://www.khronos.org/opengles/sdk/docs/man/xhtml/glViewport.xml

glewInit e initGL

`glewInit();` Inicialización de la librería GLEW. Nos da acceso a la interfaz del API más reciente en OpenGL.

```
void initGLState()
{
    glClearColor( 128./255, 200./255, 1, 0 );
    glEnable( GL_FRAMEBUFFER_SRGB );
}
```

S.J. Gortler. Foundations of 3D Computer Graphics. MIT Press. 2012.

initShaders

```
// global handles
static GLuint h_program;
static GLuint h_aTexCoord;
static GLuint h_aColor;


/* ------------------------------------------------------------------
 * initShaders()
 * upload the text of the shaders from a file to OpenGL
 * compile them
 * ask OpenGL for handles for the input variables for these shaders
 * ------------------------------------------------------------------ */
initShaders()
{
    readAndCompileShader("./shaders/simple.vshader", "./shaders/simple.fshader", &h_program );

    h_aVertex = safe_glGetAttribLocation( h_program, "aVertex" );
    h_aColor = safe_glGetAttribLocation( h_program, "aColor" );
    glBindFragDataLocation( h_program, 0, "fragColor" );
}
```

S.J. Gortler. Foundations of 3D Computer Graphics. MIT Press. 2012.

GLSL

- Un programa de GLSL debe contener un **vertex shader** y un **fragment shader**.
- La mayoría de veces se compilan diferentes programas de GLSL para dibujar diferentes objetos.
- **readAndCompileShader** - lee los dos archivos, da el código a OpenGL y crea un handle para ese programa de GLSL.
- obtenemos handles para las variables de atributos: **aVertex** y **aColor**.
- decimos a OpenGL que la variable **fragColor** es la salida de nuestro fragment shader (el color que se dibujará).

http://www.3dgraphicsfoundations.com/code.html

S.J. Gortler. Foundations of 3D Computer Graphics. MIT Press. 2012.

initVBOs

- Cargar la información de la geometría en los buffers de objetos vértice.
- Estos serán utilizados más tarde cuando dibujemos.
- La escena consiste en un cuadrado con su esquina inferior izquierda en (-0.5, -0.5) y la esquina superior derecha en (0.5, 0.5).
- Formado por dos triángulos.
- Se asocia a cada vértice un color en RGB (almacenado en sqCol).

```
//geometry
GLfloat sqVerts[6*2] =
{
     -.5, -.5,
      .5,  .5,
      .5, -.5,
     -.5, -.5,
     -.5,  .5,
      .5,  .5
};

GLfloat sqCol[6*3] =
{
     1, 0, 0,
     0, 1, 1,
     0, 0, 1,
     1, 0, 0,
     0, 1, 0,
     0, 1, 1
};
```

S.J. Gortler. Foundations of 3D Computer Graphics. MIT Press. 2012.

initVBOs

```
static GLuint sqVertBO, sqColBO


/* ------------------------------------------------------------------
 * initVBOs()
 * ------------------------------------------------------------------ */
static void initVBOs(void)
{
    glGenBuffers( 1, &sqVertBO);
    glBindBuffer( GL_ARRAY_BUFFER, sqVertBO );
    glBufferData( GL_ARRAY_BUFFER, 12*sizeof(GLfloat), sqVerts, GL_STATIC_DRAW );
    glGenBuffers( 1, &sqColBO );
    glBindBuffer( GL_ARRAY_BUFFER, sqColBO );
    glBufferData( GL_ARRAY_BUFFER, 18*sizeof(GLfloat), sqCol, GL_STATIC_DRAW );
}
```

S.J. Gortler. Foundations of 3D Computer Graphics. MIT Press. 2012.

display

```
void display(void)
{
    glUseProgram( h_program );
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
    drawObj( sqVertBO, sqColBO, 6 );
    glutSwapBuffers();

    // check for errors
    if( glGetError() != GL_NO_ERROR){
        const GLubyte *errString;
        errString = gluErrorString( errCode );
        printf("error: %s\n" errString );
```

S.J. Gortler. Foundations of 3D Computer Graphics. MIT Press. 2012.

drawObj

```
void drawObj( GLuint vertbo, GLuint colbo, int numverts )
{
    glBindBuffer( GL_ARRAY_BUFFER, vertbo );
    safe_glVertexAttribPointer2( h_aVertex );
    safe_glEnableVertexAttribArray( h_aVertex );

    glBindBuffer( GL_ARRAY_BUFFER, colbo );
    safe_glVertexAttribPointer3( h_aColor );
    safe_glEnableVertexAttribArray( h_aColor);

    glDrawArrays( GL_TRIANGLES, 0, numverts );

    safe_glDisableVertexAttribArray( h_aVertex );
    safe_glDisableVertexAttribArray( h_aColor );
}
```

S.J. Gortler. Foundations of 3D Computer Graphics. MIT Press. 2012.

Vertex Shader

```
#version 330
in vec2 aVertex;
in vec3 aColor;

out vec3 vColor;

void main()
{
    gl_Position = vec4( aVertex.x, aVertex.y, 0, 1 );
    vColor = aColor;
}
```

S.J. Gortler. Foundations of 3D Computer Graphics. MIT Press. 2012.

Fragment Shader

```
#version 330

in vec3 vColor;
out vec4 fragColor;

void main(void)
{
    fragColor = vec4( vColor.x, vColor.y, vColor.z, 1 );
}
```

S.J. Gortler. Foundations of 3D Computer Graphics. MIT Press. 2012.

Resultado

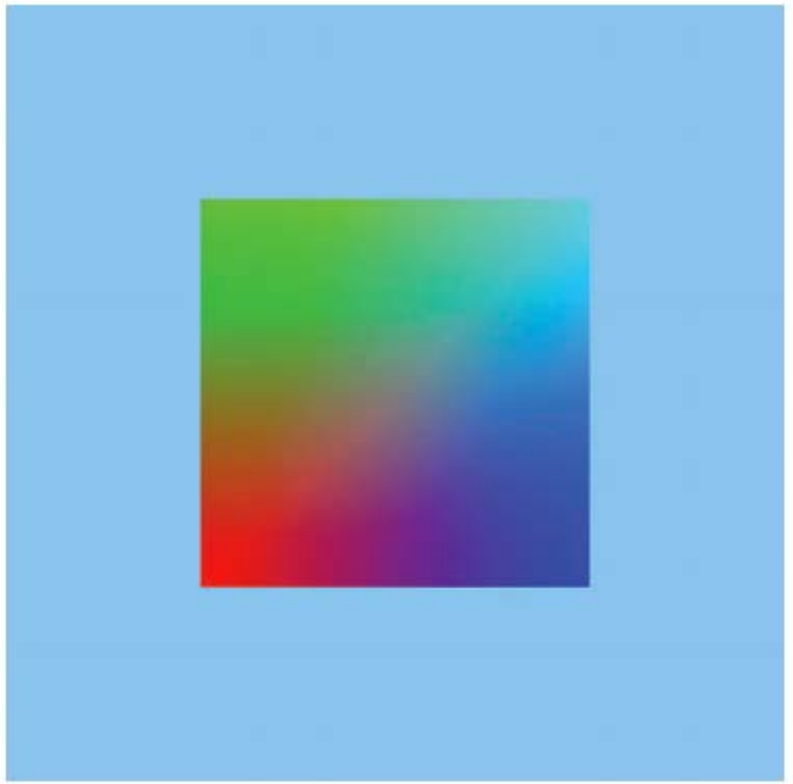

Figure A.1
A colored square is drawn in our first OpenGL program.

S.J. Gortler. Foundations of 3D Computer Graphics. MIT Press. 2012.

Vectores coordenados

- Conveniente tener tipos de datos para representar vectores coordenados (Cvec2, Cvec3, Cvec4).
- Implementar operaciones con vectores (o tomar una librería que lo haga)
  - Suma de dos Cvecs del mismo tamaño.
  - Multiplicación con un real escalar (r*v).
  - Para Cvec4 : (x,y,z,w) donde w=1 para un punto y w=0 para un vector.

S.J. Gortler. Foundations of 3D Computer Graphics. MIT Press. 2012.

Matrices

- Tipo de dato Matriz (Matrix4) para transformaciones afines.
- Debe soportar multiplicación por la derecha con un Cvec4, (M*v)
- Multiplicación de dos Matrix4 (M*N).
- Operación inversa inv(M)
- Transpuesta transpose(M).

S.J. Gortler. Foundations of 3D Computer Graphics. MIT Press. 2012.

Matrices

- Matrices útiles creadas con las siguientes operaciones:

```
Matrix4 identity();
Matrix4 makeXRotation( double ang );
Matrix4 makeYRotation( double ang );
Matrix4 makeZRotation( double ang );
Matrix4 makeScale( double sx, double sy, double sz );
Matrix4 makeTranslation( double tx, double ty, double tz );
```

- En C++ es útil que el constructor mismo regrese la matriz identidad.
- makeMixedFrame(O, E) : $(O)_T(E)_R$.
- doMtoOwrtA(M, O, A) : do M to O with respect to A : $AMA^{-1}O$

S.J. Gortler. Foundations of 3D Computer Graphics. MIT Press. 2012.

Dibujando una forma

```
static void InitGLState()
{
    glClearColor( 128./255., 200./255., 255./255., 0. );
    glClearDepth( 0.0 );
    glEnable( GL_DEPTH_TEST );
    glDepthFunc( GL_GREATER );
    glEnable( GL_CULL_FACE );
    glCullFace( GL_BACK );
}
```

- Variable global Matrix4 objRbt representa la matriz O que relaciona el marco del objeto con el del mundo.
- Variable global Matrix4 eyeRbt representa la matriz E que relaciona el marco de la cámara con el del mundo.

S.J. Gortler. Foundations of 3D Computer Graphics. MIT Press. 2012.

Dibujando una forma

```
Glfloat floorVerts[18] =
{
    -floor_size, floor_y, -floor_size,
     floor_size, floor_y,  floor_size,
     floor_size, floor_y, -floor_size,
    -floor_size, floor_y, -floor_size,
    -floor_size, floor_y,  floor_size,
     floor_size, floor_y,  floor_size
};
```

```
GLfloat floorNorms[18] =
{
    0, 1, 0,
    0, 1, 0,
    0, 1, 0,
    0, 1, 0,
    0, 1, 0,
    0, 1, 0
}
```

S.J. Gortler. Foundations of 3D Computer Graphics. MIT Press. 2012.

Dibujando una forma

```
GLfloat cubeVerts[36*3] =
{
    -0.5, -0.5, -0.5,
    -0.5, -0.5, +0.5,
    +0.5, -0.5, +0.5,
    // 33 vertices mas
};
```

```
GLfloat cubeNorms[36*3] =
{
    +0.0, -1.0, +0.0,
    +0.0, -1.0, +0.0,
    +0.0, -1.0, +0.0,
    // 33 normales mas
}
```

S.J. Gortler. Foundations of 3D Computer Graphics. MIT Press. 2012.

Inicializar los vertex buffer objects (VBOs)

```
static GLuint floorVertBO, floorNormBO, cubeNormBO, cubeNormBO;

static void initVBOs(void)
{
    glGenBuffers( 1, &floorVertBO );
    glBindBuffer( GL_ARRAY_BUFFER, floorVertBO );
    glBufferData( GL_ARRAY_BUFFER, 18*sizeof(GLfloat), floorVerts, GL_STATIC_DRAW );

    glGenBuffers( 1, &floorNormBO );
    glBindBuffer( GL_ARRAY_BUFFER, floorNormBO );
    glBufferData( GL_ARRAY_BUFFER, 18*sizeof(GLfloat), floorNorms, GL_STATIC_DRAW );

    glGenBuffers( 1, &cubeVertBO );
    glBindBuffer( GL_ARRAY_BUFFER, cubeVertBO );
    glBufferData( GL_ARRAY_BUFFER, 36*3*sizeof(GLfloat), cubeVerts, GL_STATIC_DRAW );

    glGenBuffers( 1, &cubeNormBO );
    glBindBuffer( GL_ARRAY_BUFFER, cubeNormBO );
    glBufferData( GL_ARRAY_BUFFER, 36*3*sizeof(GLfloat), cubeNorms, GL_STATIC_DRAW );
}
```

S.J. Gortler. Foundations of 3D Computer Graphics. MIT Press. 2012.

Dibujar los objetos

```
void drawObj( GLuint vertbo, GLuint normbo, int numverts )
{
    glBindBuffer( GL_ARRAY_BUFFER, vertbo );
    safe_glVertexAttribPointer( h_aVertex );
    safe_glEnableVertexAttribArray( h_aVertex );

    glBindBuffer( GL_ARRAY_BUFFER, normbo );
    safe_glVertexAttribPointer( h_aNormal );
    safe_glEnableVertexAttribArray( h_aNormal );

    glDrawArrays( GL_TRIANGLES, 0, numverts );

    safe_glDisableVertexAttribArray( h_aVertex );
    safe_glDisableVertexAttribArray( h_aNormal );
}
```

S.J. Gortler. Foundations of 3D Computer Graphics. MIT Press. 2012.

Display function

```
static void display()
{
    safe_glUseProgram( h_program_ );
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

    Matrix4 projmat = makeProjection( frust_fovy, frust_ar, frust_near, frust_far );
    sendProjectionMatrix( projmat );

    Matrix4 MVM = inv(eyeRbt);
    Matrix4 NMVM = normalMatrix( MVM );
    sendModelViewNormalMatrix( MVM, NMVM );

    safe_glVertexAttrib3f( h_aColor, 0.6, 0.8, 0.6 );
    drawObj( floorVertBO, floorNormBO, 6 );

    MVM = inv(eyeRbt)*objRbt;
    NMVM = normalMatrix( MVM );
    sendModelViewNormalMatrix( MVM, NMVM );

    safe_glVertexAttrib3f( h_aColor, 0.0, 0.0, 1.0 );
    drawObj( cubeVertBO, cubeNormBO, 36 );

    glutSwapBuffers();

    if( glGetError() != GL_NO_ERROR){
        const GLuByte *errString;
        errString = gluErrorString( errCode);
        printf("error: %s\n", errString );
    }
}
```

S.J. Gortler. Foundations of 3D Computer Graphics. MIT Press. 2012.