

Introducción a OpenGL

7. Iluminación

- Es imprescindible la activación de la Eliminación de Partes Ocultas (EPO) para una correcta iluminación.
 - Pasos a seguir en OpenGL para su activación:

Inicializar OpenGL `glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH);`

Borrar z-Buffer `glClearDepth (1.0);`
`glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);`

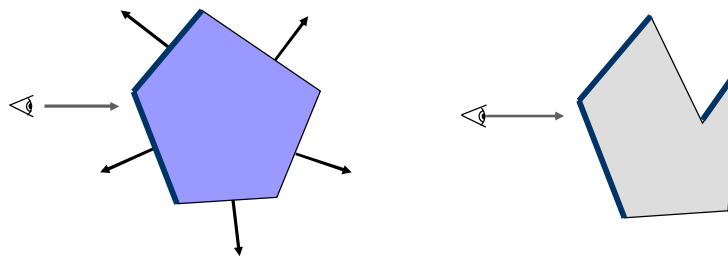
Activar EPO `glEnable(GL_DEPTH_TEST);`



Introducción a OpenGL

7. Iluminación

- Optimización mediante eliminación de la cara trasera



`void glCullFace (GLenum modo)` donde `modo` toma el valor `GL_BACK` o `GL_FRONT` para especificar las caras que no se consideran para el cálculo de la eliminación de partes ocultas

```
....  
glCullFace(GL_BACK);  
glEnable(GL_CULL_FACE);  
....
```



7. Iluminación

- Problema con el uso de CullFace



83

7. Iluminación

- Pasos para realizar la iluminación de una escena
 - Definir las luces (características). Una luz es un objeto más de la escena (ModelView)
 - Habilitar la iluminación (`glEnable (GL_LIGHTING)` obsoleto desde la versión 3.0)
 - Usar normales a las caras o a los vértices según el modo de suavizado
 - Definir los materiales de los objetos
 - Cambiar el modelo de iluminación

Introducción a OpenGL

7. Iluminación

- Creación de luces en una escena.

`void glLight {if}[v] (GLenum n_luz, GLenum p_luz, TYPE parametro)` crea una luz siendo `n_luz` el índice de la luz que puede expresarse con los nombres `GL_LIGHT0`, `GL_LIGHT1`, ... `GL_LIGHT7`. Las características de la luz se fijan con `p_luz` y `parametro` indica el valor de la característica, que puede ser un real o un vector.

Valores de `p_luz` (entre llaves se indica el valor por defecto):

<code>GL_AMBIENT</code> {(0.0,0.0,0.0,1.0)}	intensidad ambiente en RGBA .
<code>GL_DIFFUSE</code> {(1.0,1.0,1.0,1.0)}	intensidad difusa en RGBA .
<code>GL_SPECULAR</code> {(1.0,1.0,1.0,1.0)}	intensidad especular en RGBA .
<code>GL_POSITION</code> {(0.0,0.0,1.0,0.0)}	la posición de la luz se expresa con un vector (x,y,z,w), con <code>w=0.0</code> se indica que la luz es direccional y con <code>w=1.0</code> luz puntual.
<code>GL_SPOT_DIRECTION</code> {(0.0,0.0,-1.0)}	dirección de un cono de luz.
<code>GL_SPOT_EXPONENT</code> {0.0}	exponente de un cono de luz.
<code>GL_SPOT_CUTOFF</code> {180.0}	ángulo de corte de un cono de luz.
<code>GL_CONSTANT_ATTENUATION</code> {1.0}	constante de atenuación.
<code>GL_LINEAR_ATTENUATION</code> {0.0}	factor lineal de atenuación.
<code>GL_QUADRATIC_ATTENUATION</code> {0.0}	factor cuadrático.

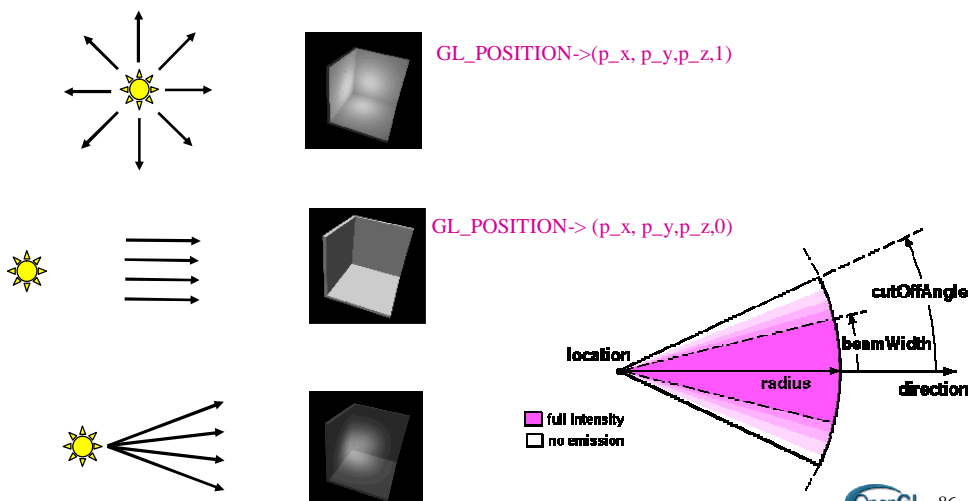


85

Introducción a OpenGL

7. Iluminación

- Creación de luces en una escena.



86

7. Iluminación

- Creación de luces en una escena.

```
void luces ( )
{
    GLfloat luz_ambiente [ ] = {0.3, 0.3, 0.3, 1.0},
           luz_difusa [ ] = {1.0, 1.0, 1.0, 1.0};

    glLightfv (GL_LIGHT0, GL_AMBIENT, luz_ambiente);
    glLightfv (GL_LIGHT1, GL_DIFFUSE, luz_difusa);
    glEnable (GL_LIGHTING); ←
    glEnable (GL_LIGHT0); ←
    glEnable (GL_LIGHT1); ←
}

void posicion_luz1 ( )
{
    GLfloat luz_posicion [ ] = {2.0, 4.0, 10.0 , ....};

    glLightfv (GL_LIGHT1, GL_POSITION, luz_posicion);
}
```

7. Iluminación

- Control de la posición y orientación de una luz.
 - OpenGL trata a una fuente de luz como si fuera otra primitiva gráfica.
 - Cuando hacemos uso de `glLightfv(...,GL_POSITION,...)` y/o `glLightfv(...,GL_SPOT_DIRECTION,...)` la posición y/o dirección es transformada por la matriz de modelado actual.
 - Para que una luz permanezca estacionaria o fija ha de ser creada su posición después de conmutar a la pila de matrices de modelado:

```
....
glViewport ( ... );
glMatrixMode (GL_PROJECTION);
glLoadIdentity ( );
glOrtho ( ... );
....
glMatrixMode (GL_ModelView);
glLoadIdentity ( );
posicion_luz1 ( ); ←
luces() ←
...
objetos ( );
....
```

7. Iluminación

- Control de la posición y orientación de una luz.
 - Rotación y/o traslación de una la luz sin que éstas transformaciones afecten a los objetos de la escena.

```
....
glMatrixMode (GL_ModelView);
glLoadIdentity ( );
posicion_luz1( );
...
objetos( );
...

void posicion_luz1 ( )
{
    GLfloat luz_posicion [ ] = {1.0, 1.0, 0.0 ,1.0};
    glPushMatrix(); ←
    glTranslatef ( ... );
    glRotatef ( angulo, ... );
    glLightfv (GL_LIGHT0, GL_POSITION, luz_posicion);
    glPopMatrix() ←
}
....
```

7. Iluminación

- Usando normales.

```
void objetos ( )
{
    ...
    glShadeModel(GL_FLAT); ←
    glEnable(GL_NORMALIZE);
    glEnable(GL_LIGHTING);
    ...
    glBegin(GL_QUADS);
        // Frente
        glNormal3f( 0.0f, 0.0f, 1.0f); ←
        glVertex3f(-1.0f, -1.0f, 1.0f);
        glVertex3f( 1.0f, -1.0f, 1.0f);
        glVertex3f( 1.0f, 1.0f, 1.0f);
        glVertex3f(-1.0f, 1.0f, 1.0f);
        // parte de Atras
        glNormal3f( 0.0f, 0.0f,-1.0f); ←
        glVertex3f(-1.0f, -1.0f, -1.0f);
        glVertex3f(-1.0f, 1.0f, -1.0f);
        glVertex3f( 1.0f, 1.0f, -1.0f);
        glVertex3f( 1.0f, -1.0f, -1.0f);
    ...
    glEnd();
    ...
}
```



7. Iluminación

- Usando normales.

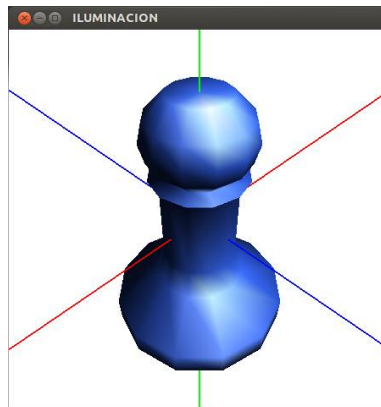
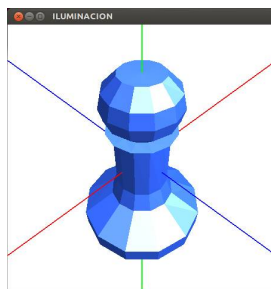
```
...
glShadeModel(GL_SMOOTH);
glEnable(GL_NORMALIZE);
glEnable(GL_LIGHTING);
...
glBegin(GL_QUADS);
    // Frente
    glNormal3f(-1.0f, -1.0f, 1.0f);
    glVertex3f(-1.0f, -1.0f, 1.0f);
    glNormal3f( 1.0f, -1.0f, 1.0f);
    glVertex3f( 1.0f, -1.0f, 1.0f);
    glNormal3f( 1.0f, 1.0f, 1.0f);
    glVertex3f( 1.0f, 1.0f, 1.0f);
    glNormal3f(-1.0f, 1.0f, 1.0f);
    glVertex3f(-1.0f, 1.0f, 1.0f);
    // parte de Atrás
    glNormal3f(-1.0f, -1.0f, -1.0f);
    glVertex3f(-1.0f, -1.0f, -1.0f);
    glNormal3f(-1.0f, 1.0f, -1.0f);
    glVertex3f(-1.0f, 1.0f, -1.0f);
    ...
glEnd();
...
```



7. Iluminación

- Usando normales.

```
...
glBegin(GL_TRIANGLES);
for (i=0;i<n_c;i++)
{
    for (j=0;j<3;j++)
    {
        n_v=malla->car[i].ind_ver[j];
        glNormal3f(malla->nor_v[n_v].coord[0], malla->nor_v[n_v].coord[1], malla->nor_v[n_v].coord[2]);
        glVertex3f(malla->ver[n_v].coord[0], malla->ver[n_v].coord[1], malla->ver[n_v].coord[2]);
    }
}
glEnd();
...
```



7. Iluminación

- Normalización.

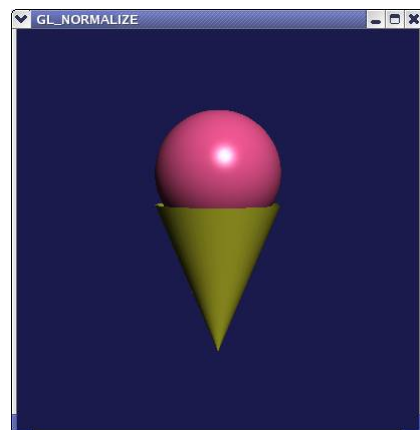
```
void objetos ( )  
{  
    ...  
    // Esfera  
    glPushMatrix();  
    glTranslatef( 0.0f, 1.9f, 0.0f);  
    glScalef( 0.10f, 0.10f, 0.10f); ←  
    gluSphere(qobj,10.0,72,72);  
    glPopMatrix();  
  
    ...  
    // Cone  
    glPushMatrix();  
    glTranslatef( 0.0f, -2.0f, -0.0f);  
    glRotatef( -90,1,0,0);  
    glScalef( 0.20f, 0.20f, 0.30f); ←  
    gluCylinder(qobj,0.0,5.0,12,36,2);  
    glPopMatrix();  
    ...  
}
```



7. Iluminación

- Normalización.

```
void objetos ( )  
{  
    ...  
    glEnable(GL_NORMALIZE); ←  
    glEnable(GL_LIGHTING);  
    ...  
    // Esfera  
    glPushMatrix();  
    glTranslatef( 0.0f, 1.9f, 0.0f);  
    glScalef( 0.10f, 0.10f, 0.10f); ←  
    gluSphere(qobj,10.0,72,72);  
    glPopMatrix();  
    ...  
    // Cone  
    glPushMatrix();  
    glTranslatef( 0.0f, -2.0f, -0.0f);  
    glRotatef( -90,1,0,0);  
    glScalef( 0.20f, 0.20f, 0.30f); ←  
    gluCylinder(qobj,0.0,5.0,12,36,2);  
    glPopMatrix();  
    ...  
}
```



7. Iluminación

- Definiendo el material de los objetos.

`void glMaterial{if}[v] (GLenum cara, GLenum p_cara, TYPE parametro)` especifica el material con el que está construido un objeto. La variable `cara` puede ser `GL_FRONT`, `GL_BACK`, o `GL_FRONT_AND_BACK` para indicar a que caras del objeto se aplica el material. Las características del material de se fijan con `p_cara` y `parametro` indica el valor de la característica, que puede ser un real o un vector.

Valores de `p_cara` (entre llaves se indica el valor por defecto):

`GL_AMBIENT` {(0.2,0.2,0.2,1.0)}
`GL_DIFFUSE` {(0.8,0.8,0.8,1.0)}
`GL_AMBIENT_AND_DIFFUSE`

color ambiental del material en **RGBA**.

color difuso del material en **RGBA**.

se fijan al mismo tiempo los colores ambiental y difuso.

`GL_SPECULAR`{(0.0,0.0,1.0,1.0)}
`GL_SHININESS` {0.0}
`GL_EMISSION`{0.0,0.0,0.0,1.0}

color especular del material en **RGBA**.

exponente de Phong. Está entre 0.0 y 128.0.

color emisor del material. Se utiliza para simular fuentes de luz.

`GL_COLOR_INDEXES` {0,1,1}

índices ambiente, difuso y especular.



95

7. Iluminación

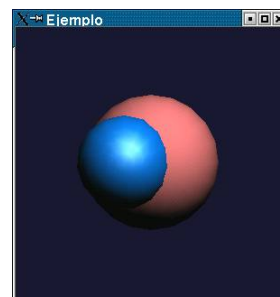
- Definiendo el material de los objetos.

```
void luces ( )
{ GLfloat   luz_difusa[]={1.0,1.0,1.0,1.0 },
  luz_especular[]={1.0,1.0,1.0,1.0 },
  luz_posicion[]={5.0,10.0,15.0,1.0};
```

```
glLightfv (GL_LIGHT1, GL_DIFFUSE, luz_difusa);
glLightfv (GL_LIGHT1, GL_SPECULAR, luz_especular);
glLightfv (GL_LIGHT1, GL_POSITION, luz_posicion);
glEnable (GL_LIGHTING);
glDisable (GL_LIGHT0);
glEnable (GL_LIGHT1); }
```

```
void objetos ( )
{ GLfloat   mat_difuso1[]={1.0,0.5,0.5,1.0 },
  mat_difuso2[]={0.0,0.5,1.0,1.0 },
  mat_especular2[]={0.6,0.6,0.6,1.0},
  mat_brillo2=20;
```

```
glMaterialfv(GL_FRONT, GL_DIFFUSE,mat_difuso1);
glutSolidSphere(1.0,20,20);
glMaterialfv(GL_FRONT, GL_DIFFUSE,mat_difuso2);
glMaterialfv(GL_FRONT, GL_SPECULAR,mat_especular2);
glMaterialf(GL_FRONT, GL_SHININESS,mat_brillo2);
glTranslatef(-0.4,0.0,0.7);
glutSolidSphere(0.7,15,15); }
```



96

Introducción a OpenGL

7. Iluminación

Material	GL_AMBIENT	GL_DIFFUSE	GL_SPECULAR	GL_SHININESS
Brass	0.329412 0.223529 0.027451 1.0	0.780392 0.568627 0.113725 1.0	0.992157 0.941176 0.807843 1.0	27.8974
Bronze	0.2125 0.1275 0.054 1.0	0.714 0.4284 0.18144 1.0	0.393548 0.271906 0.166721 1.0	25.6
Polished Bronze	0.25 0.148 0.06475 1.0	0.4 0.2368 0.1036 1.0	0.774597 0.458561 0.200621 1.0	76.8
Chrome	0.25 0.25 0.25 1.0	0.4 0.4 0.4 1.0	0.774597 0.774597 0.774597 1.0	76.8
Copper	0.19125 0.0735 0.0225 1.0	0.7038 0.27048 0.0828 1.0	0.256777 0.137622 0.086014 1.0	12.8
Polished Copper	0.2295 0.08825 0.0275 1.0	0.5508 0.2118 0.066 1.0	0.580594 0.223257 0.0695701 1.0	51.2
Gold	0.24725 0.1995 0.0745 1.0	0.75164 0.60648 0.22648 1.0	0.628281 0.555802 0.366065 1.0	51.2
Polished Gold	0.24725 0.2245 0.0645 1.0	0.34615 0.3143 0.0903 1.0	0.797357 0.723991 0.208006 1.0	83.2
Pewter	0.105882 0.058824 0.113725 1.0	0.427451 0.470588 0.541176 1.0	0.333333 0.333333 0.521569 1.0	9.84615

Material	GL_AMBIENT	GL_DIFFUSE	GL_SPECULAR	GL_SHININESS
Silver	0.19225 0.19225 0.19225 1.0	0.50754 0.50754 0.50754 1.0	0.508273 0.508273 0.508273 1.0	51.2
Polished Silver	0.23125 0.23125 0.23125 1.0	0.2775 0.2775 0.2775 1.0	0.773911 0.773911 0.773911 1.0	89.6
Emerald	0.0215 0.1745 0.0215 0.55	0.07568 0.61424 0.07568 0.55	0.633 0.727811 0.633 0.55	76.8
Jade	0.135 0.2225 0.1575 0.95	0.54 0.89 0.63 0.95	0.316228 0.316228 0.316228 0.95	12.8
Obsidian	0.05375 0.05 0.06625 0.82	0.18275 0.17 0.22525 0.82	0.332741 0.328634 0.346435 0.82	38.4
Pearl	0.25 0.20725 0.20725 0.922	1.0 0.829 0.829 0.922	0.296648 0.296648 0.296648 0.922	11.264
Ruby	0.1745 0.01175 0.01175 0.55	0.61424 0.04136 0.04136 0.55	0.727811 0.626959 0.626959 0.55	76.8
Turquoise	0.1 0.18725 0.1745 0.8	0.396 0.74151 0.69102 0.8	0.297254 0.30829 0.306678 0.8	12.8
Black Plastic	0.0 0.0 0.0 1.0	0.01 0.01 0.01 1.0	0.50 0.50 0.50 1.0	32
Black Rubber	0.02 0.02 0.02 1.0	0.01 0.01 0.01 1.0	0.4 0.4 0.4 1.0	10

97

Introducción a OpenGL

7. Iluminación

- Cambiando el modelo de iluminación.
 - Modelo de iluminación de OpenGL

Si la intensidad obtenida excede de 1 se trunca

$$I_{r,g,b} = E_{m(r,g,b)} + I_{a_g(r,g,b)} K_{a_m(r,g,b)} + \sum_i \left(\frac{1}{a_i + b_i d_{i,p} + c_i d_{i,p}^2} \right) (\text{efectos_spotlight})_i I_{i(r,g,b)}$$

$$I_{i(r,g,b)} = I_{a_i(r,g,b)} K_{a_m(r,g,b)} + I_{d_i(r,g,b)} K_{d_m(r,g,b)} \max(\vec{N}_p \cdot \vec{L}_i, 0) + I_{s_i(r,g,b)} K_{s_m(r,g,b)} \max(\vec{N}_p \cdot \vec{H}_i, 0)^n$$

7. Iluminación

- Cambiando el modelo de iluminación.
 - Modelo de iluminación de OpenGL

$I_{a_g(r,g,b)}$	intensidad ambiental de la luz
$E_{m(r,g,b)}$	emisividad del material
$K_{a_m(r,g,b)}, K_{d_m(r,g,b)}, K_{s_m(r,g,b)}$	constantes ambiental, difusa y especular del material
$I_{a_i(r,g,b)}, I_{d_i(r,g,b)}, I_{s_i(r,g,b)}$	colores ambiental, difuso y especular de la luz i
$\vec{N}_p \cdot \vec{L}_i$	producto escalar de la normal al punto p y el vector de la luz i
$\vec{N}_p \cdot \vec{H}_i$	producto escalar de la normal al punto p y el vector H de la luz i
a_i, b_i, c_i	factores de atenuación constante, lineal y cuadrático de la luz i
$d_{i,p}$	distancia entre la luz i con el punto p
$(\text{efectos_spotlight})_i$	1 si la luz i no es de tipo spotlight

7. Iluminación

- Cambiando el modelo de iluminación.

`void glLightModeli(int p_iluminacion, GLenum parametro)` permite controlar como las luces activas afectan a la iluminación de la escena.

La variable `p_iluminacion` puede ser `GL_LIGHT_MODEL_AMBIENT`, `GL_LIGHT_MODEL_LOCAL_VIEWER` o `GL_LIGHT_MODEL_TWO_SIDE` y `parametro` indica el valor de la característica, que puede ser un entero, un real o un vector.

`GL_LIGHT_MODEL_AMBIENT` define la intensidad una luz ambiental global y sus valores son del tipo `RGBA`

`GL_LIGHT_MODEL_LOCAL_VIEWER` modifica el cálculo de las reflexiones especulares. El observador está en Z infinito respecto del sistema de coordenadas del observador (`GL_FALSE`, valor por defecto) o a una distancia finita (`GL_TRUE`)

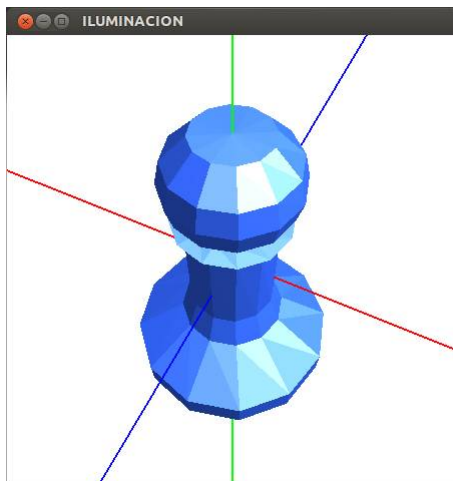
`GL_LIGHT_MODEL_TWO_SIDE` define si la luz debe iluminar también la parte trasera de las caras (`GL_TRUE`).

7. Iluminación



```
...  
glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER,  
              GL_TRUE);  
...
```

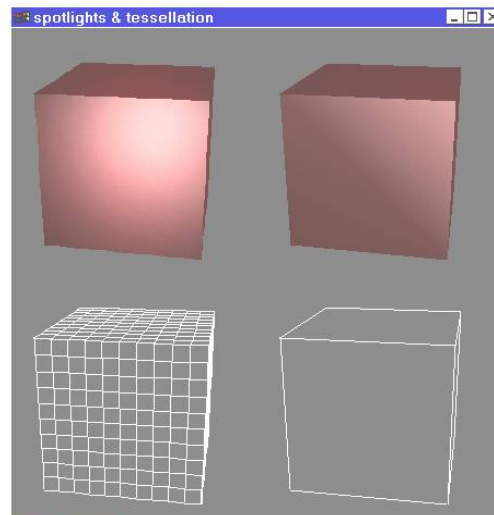
7. Iluminación



```
glShadeMode(GL_FLAT);  
...  
glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER,  
              GL_TRUE);  
...
```

7. Iluminación

- La importancia del número y tamaño de las caras.



8. Texturas

- Pasos para utilizar una imagen como textura
 - Cargar la imagen. OpenGL no dispone de funciones para ello.
 - Habilitar uso de texturas
 - Especificar la imagen usada como textura de OpenGL
 - Indicar como la textura se aplica (filtros, repetición)
 - Especificar las coordenadas de textura (texels) para los vértices de los objetos (explícita y procedural)

Introducción a OpenGL

8. Texturas

- Activar o desactivar la aplicación de texturas.

```
glEnable(GL_TEXTURE_2D);  
...  
glDisable(GL_TEXTURE_2D);
```

- Especificar la imagen usada como textura

`void glTexImage2D (GLenum objetivo, GLint nivel, GLint componentes, GLsizei ancho, GLsizei alto, GLint borde, GLenum formato, GLenum tipo, const GLvoid *texels)` donde:

`objetivo` normalmente vale `GL_TEXTURE_2D`,

`nivel` indica el nivel de detalle de la textura (habitualmente vale `0`),

`componentes` indica que componentes de color que se usan (`GL_LUMINANCE`, `GL_LUMINANCE_ALPHA`, `GL_DEPTH_COMPONENT`, `GL_RGB` o `GL_RGBA`),

`ancho` y `alto` especifican el tamaño de la imagen usada como textura y deben ser potencia de 2 (como mínimo una imagen ha de tener 64x64 píxeles),

`borde` indica si se utiliza un borde en la textura (`1`) o no (`0`) (normalmente es `0`),

`formato` describe los formatos de los píxeles de la imagen (los mismos que en `glReadPixels()`),

`tipo` es el tipo de datos usado para los píxeles (`GL_BYTE`, `GL_UNSIGNED_BYTE`, `GL_SHORT`, `GL_UNSIGNED_SHORT`, `GL_INT`, `GL_UNSIGNED_INT`, `GL_FLOAT`),

`*texels` un puntero a la imagen de la textura.



105

Introducción a OpenGL

8. Texturas

- Especificar como la textura se aplica

– Filtrado.

`void glTexParameterf (GLenum objetivo, GLenum nombre, GLfloat parametro)` donde:

`objetivo` normalmente vale `GL_TEXTURE_2D`,

`nombre` puede valer `GL_TEXTURE_MAG_FILTER` (si se especifica el filtro para cuando la textura está lejos) o `GL_TEXTURE_MIN_FILTER` (si se especifica el filtro para cuando la textura está cerca),

`parametro` indica el tipo de filtro a aplicar que puede ser `GL_NEAREST` (el texel con coordenadas más cerca del centro del píxel es el que se usa) o `GL_LINEAR` (se aplica un filtrado bilineal usando una matriz 2x2 de texels para cada píxel).

```
...  
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);  
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);  
...
```



106

Introducción a OpenGL

8. Texturas

- Especificar como la textura se aplica
 - Repetición.

`void glTexParameterf (GLenum objetivo, GLenum nombre, GLint parametro)` donde:

`objetivo` normalmente vale `GL_TEXTURE_2D`,
`nombre` puede valer `GL_TEXTURE_WRAP_S` (coordenadas X de la textura) o `GL_TEXTURE_WRAP_T` (coordenadas Y de la textura),
`parametro` indica la forma de repetición de la textura que puede ser `GL_CLAMP` (se repite el borde de la textura), `GL_CLAMP_TO_EDGE` (se ignora el borde), `GL_REPEAT` (se repite la textura completa) o `GL_MIRRORED_REPEAT` (se invierte la orientación de la textura en los bordes).

```
...  
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);  
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);  
...
```



107

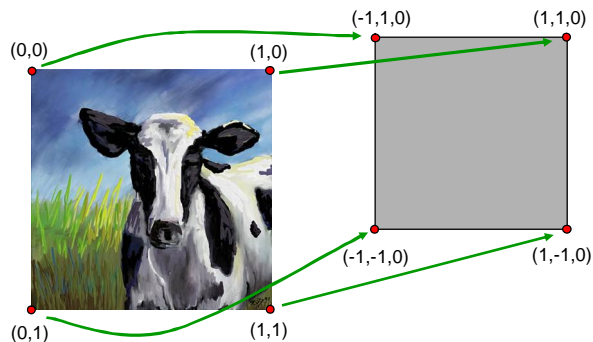
Introducción a OpenGL

8. Texturas

- Especificar la correspondencia entre las coordenadas de la imagen usada como textura y los vértices de los objetos.

`void glTexCoord2f({f,d}) ({GLfloat, GLdouble} s, {GLfloat, GLdouble} t)`

```
...  
glBegin(GL_QUADS);  
glNormal3f( 0.0, 0.0, 1.0);  
glTexCoord2f(0.0, 1.0);  
glVertex3f(-1.0, -1.0, 0.0);  
glTexCoord2f(1.0, 1.0);  
glVertex3f( 1.0, -1.0, 0.0);  
glTexCoord2f(1.0, 0.0);  
glVertex3f( 1.0,  1.0, 0.0);  
glTexCoord2f(0.0, 0.0);  
glVertex3f(-1.0,  1.0, 0.0);  
glEnd();  
...
```

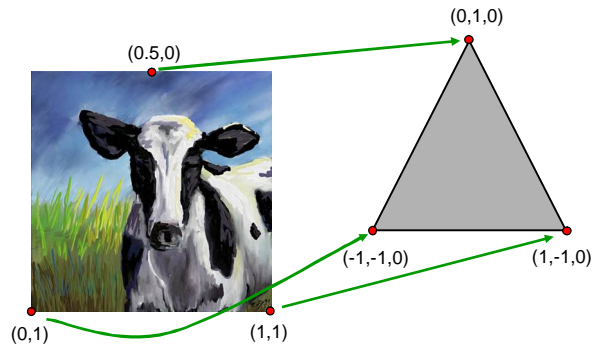


108

8. Texturas

- Especificar la correspondencia entre las coordenadas de la imagen usada como textura y los vértices de los objetos.

```
...  
glBegin(GL_POLYGON);  
glNormal3f( 0.0, 0.0, 1.0);  
glTexCoord2f(0.0, 1.0);  
glVertex3f(-1.0, -1.0, 0.0);  
glTexCoord2f(1.0, 1.0);  
glVertex3f( 1.0, -1.0, 0.0);  
glTexCoord2f(0.5, 0.0);  
glVertex3f( 0.0, 1.0, 0.0);  
glEnd();  
...
```



8. Texturas

- Especificar la correspondencia entre las coordenadas de la imagen usada como textura y los vértices de los objetos.



Introducción a OpenGL

8. Texturas

- Ejemplo.

```
...
#define Ancho_max 512
#define Alto_max 512
GLubyte imagen[Ancho_max][Alto_max][3];
...

void Cargaimagen (char *nombre, int N, int M,
                  GLubyte imagen[ ][ ][3])
{
    int i, j;
    FILE *file;

    file=fopen(nombre,"rb");
    if (file==NULL)
        {printf("Error fichero\n"); exit(1); }
        for (i = 0; i < N; i++) {
            for (j = 0; j < M; j++) {
                fread(&imagen[i][j], sizeof(unsigned char), 3, file);
            }
        }
    fclose(file);
}
...
```

```
...
void init_textura (void)
{
    Cargaimagen("imagen.rgb",imagen,256,256);
    glEnable(GL_DEPTH_TEST);
    glTexImage2D(GL_TEXTURE_2D, 0, 3,
                256, 256, 0,
                GL_RGB, GL_UNSIGNED_BYTE,
                &imagen[0][0][0]);
    glTexParameterf(GL_TEXTURE_2D,
                    GL_TEXTURE_MAG_FILTER,
                    GL_LINEAR);
    glTexParameterf(GL_TEXTURE_2D,
                    GL_TEXTURE_MIN_FILTER,
                    GL_LINEAR);
    glEnable(GL_TEXTURE_2D);
}
...
```



Introducción a OpenGL

8. Texturas

- Ejemplo (continuación).

```
...
void init_luces (void)
{
    GLfloat ambient[] = {0.5, 0.5, 0.5, 1.0};
    GLfloat position[] = {0.0, 0.0, 1.0, 0.0};

    GLfloat mat_diffuse[] = {0.9, 0.9, 0.9, 1.0};
    ...
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
    glLightfv(GL_LIGHT0, GL_POSITION, position);
    ...
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
}
...
void objeto (void)
{
    glBegin(...)
        glNormal3f(...);
        glTexCoord2f(...);
        glVertex3f(...);
    ...
    glEnd();
}

```

```
...
void display (void)
{
    glClear (GL_COLOR_BUFFER_BIT |
            GL_DEPTH_BUFFER_BIT);
    glMatrixMode( GL_MODELVIEW);
    ...
    objeto();
    ...
}
...
int main (int argc, char** argv)
{
    ...
    init_luces();
    init_textura();
    ...
}

```



Introducción a OpenGL

8. Texturas

- Aplicación procedural de texturas a objetos.

- Se utiliza la función de generación de coordenadas de textura

`void glTexGen(i){d}{v}(GLenum coordenadas, GLenum nombre, GLint parametro)` donde:

`coordenadas` puede valer `GL_S` o `GL_T`

`nombre` puede ser `GL_TEXTURE_G_MODE`, `GL_OBJECT_PLANE` o `GL_EYE_PLANE`.

`parametro` indica como se ha de calcular la transformación de la textura. Cuando fijamos la variable `nombre=GL_TEXTURE_G_MODE`, `parametro` puede valer `GL_OBJECT_LINEAR`, `GL_EYE_LINEAR`, `GL_SPHERE_MAP` o `GL_NORMAL_MAP`. Si fijamos `nombre` bien a `GL_OBJECT_PLANE` o bien a `GL_EYE_PLANE`, `parametro` ha de ser un vector que contenga los coeficientes de la ecuación de un plano $Ax+By+Cz+D=0$

- Se activa esta generación de coordenadas de textura

```
...
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
...
```



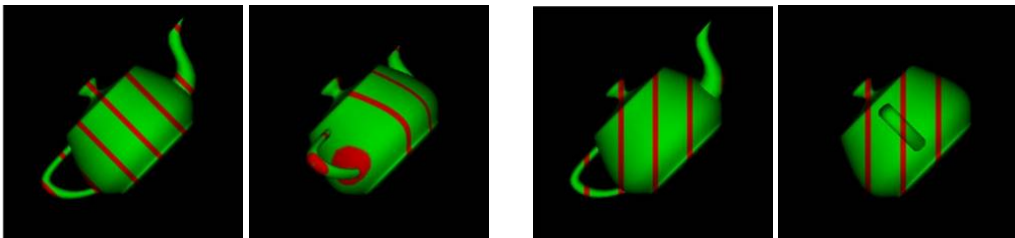
Introducción a OpenGL

8. Texturas

- Aplicación procedural de texturas a objetos.

- Con `GL_OBJECT_LINEAR` la textura se mueve con el objeto

- Con `GL_EYE_LINEAR` las coordenadas de textura se calculan con respecto al observador



8. Texturas

- Aplicación procedural de texturas a objetos.

– Ejemplo 1

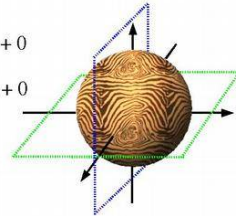
```
...
static GLfloat plano_s[4]={1.0, 0.0, 0.0, 0.0},
                plano_t[4]={0.0, 1.0, 0.0, 0.0};
...
Cargaimagen(...);
glTexImage2D(GL_TEXTURE_2D, 0, 3, ...,
            ..., 0, GL_RGB, GL_UNSIGNED_BYTE,
            &ColorImagen[0][0][0]);
```

```
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glEnable(GL_TEXTURE_2D);
```

```
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
glTexGenfv(GL_S, GL_OBJECT_PLANE, plano_s);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
glTexGenfv(GL_T, GL_OBJECT_PLANE, plano_t);
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
...
glutSolidSphere(2.8, 24, 24);
```

$$S = 1x + 0y + 0z + 0$$

$$T = 0x + 1y + 0z + 0$$



8. Texturas

- Aplicación procedural de texturas a objetos.

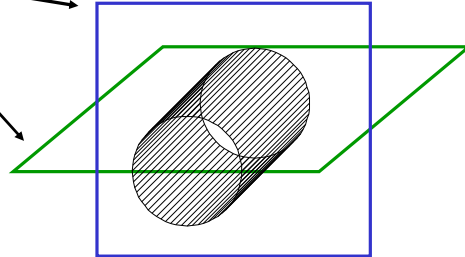
– Ejemplo 1 (continuación). Problema en la aplicación de la textura



8. Texturas

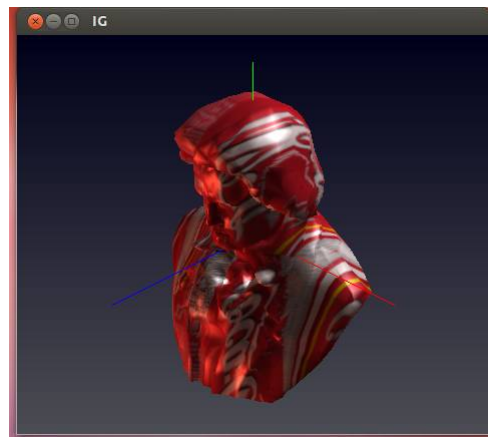
- Aplicación procedural de texturas a objetos.
 - Ejemplo 2

```
....  
static GLfloat plano_s[4]={0.0, 0.0, 1.0, 0.0},  
                plano_t[4]={0.0, 1.0, 0.0, 0.0};  
....  
qobj = gluNewQuadric();  
gluQuadricDrawStyle(qobj, GLU_FILL);  
gluCylinder(qobj, 2.5, 2.5, 3.5, 24, 2);  
....
```



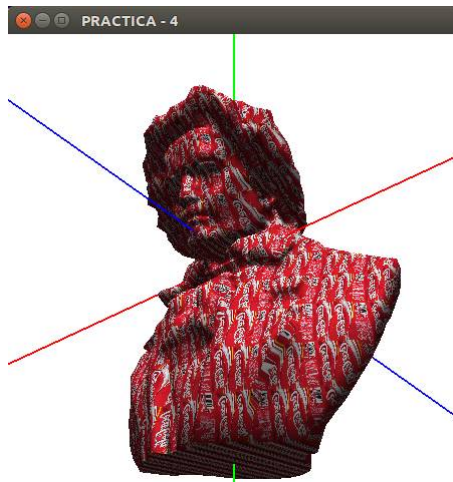
8. Texturas

- Aplicación procedural de texturas a objetos.
 - Objeto PLY



8. Texturas

- Aplicación procedural de texturas a objetos.
 - Objeto PLY



```
...
static GLfloat
plano_s[4]={1.0, 0.0, 0.0, 0.0},
plano_t[4]={0.0, 1.0, 0.0, 0.0};
...
```

8. Texturas

- Aplicación procedural de texturas a objetos.
 - Objeto PLY



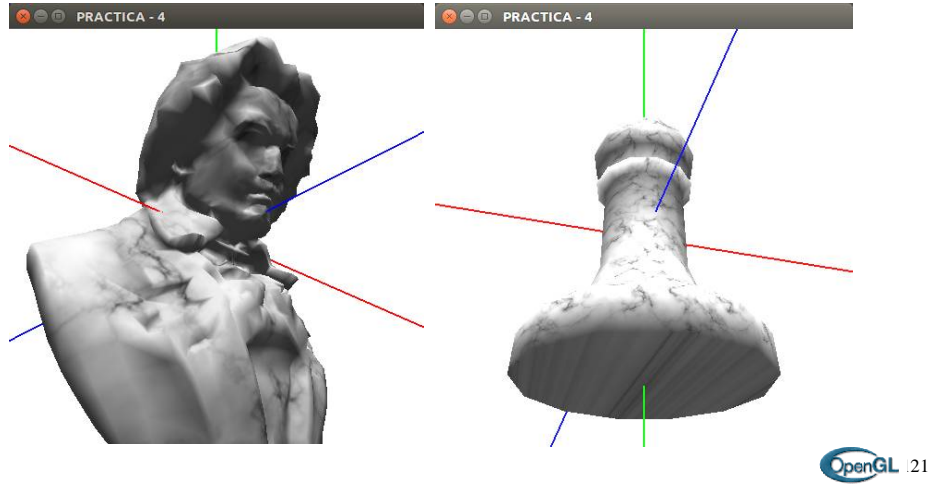
```
...
static GLfloat
plano_s[4]={1.0, 0.0, 0.0, 0.0},
plano_t[4]={0.0, 1.0, 0.0, 0.0};
...

// Se calculan los valores máximos y
// mínimos de los vértices de la figura
plano_s[0]/=(max_x-min_x);
plano_t[1]/=(max_y-min_y);
```

Introducción a OpenGL

8. Texturas

- Aplicación procedural de texturas a objetos.
 - Objeto PLY y objeto por revolución con textura natural



Introducción a OpenGL

8. Texturas

- Usando más de una textura.
 - OpenGL puede usar varias texturas pero solo una estará activa
 - Para diferenciar las texturas se les asigna un identificador

`void glGenTextures(GLsizei n, GLuint * ind_texturas)` donde:

`n` es un número entero

`ind_texturas` es un puntero a un array de enteros sin signo. Devuelve en `ind_texturas` los identificadores asignados a las `n` texturas

```
GLuint id_t;  
// Crea un solo identificador  
glGenTextures(1, &id_t);
```

```
GLuint id_t[N] ;  
// Crea varios identificadores  
glGenTextures(N, id_t);
```

Introducción a OpenGL

8. Texturas

- Usando más de una textura.
 - Cambiando la textura activa

`void glBindTexture(GLenum objetivo, GLuint ind_textura)` donde:

`objetivo` es normalmente `GL_TEXTURE_2D`

`ind_textura` es el identificador de textura a activar

```
// cubo con texturas distintas en sus caras
GLuint id_t[6];
glGenTextures(6, id_t);
...
glBindTexture(GL_TEXTURE_2D,id_t[0]);
CargarImagen(...);
glTexImage2D(...);
...
glBindTexture(GL_TEXTURE_2D,id_t[1]);
CargarImagen(...);
glTexImage2D(...);
...
...
glBindTexture(GL_TEXTURE_2D,id_t[0]);
// cara frontal del cubo
glBegin(GL_QUADS);
glTexCoord2f(0.0,0.0); glVertex3f(-1.0, 1.0, 1.0);
...
glEnd();
// seleccionar otra textura
glBindTexture(GL_TEXTURE_2D,id_t[1]);
glBegin(GL_QUADS);
glTexCoord2f(0.0,0.0f); glVertex3f(1.0,1.0,1.0);
....
glEnd();
...
```



Introducción a OpenGL

8. Texturas

- Usando más de una textura.

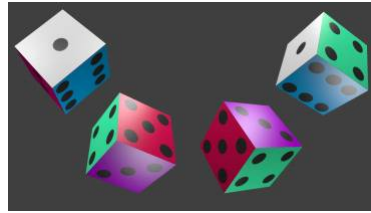
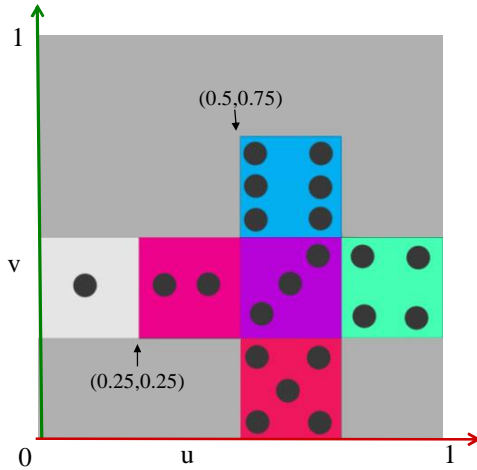


Andrés Campos Guijarro
FIG curso 2010



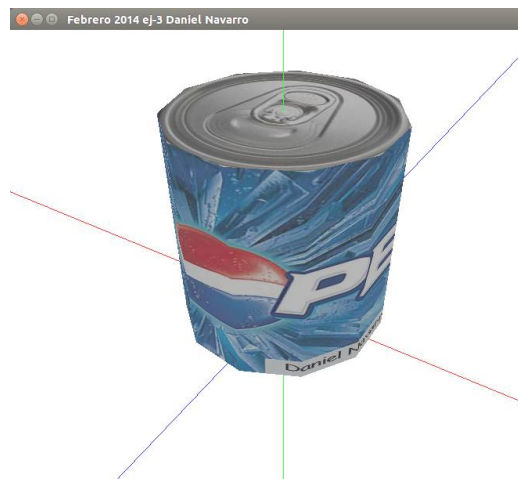
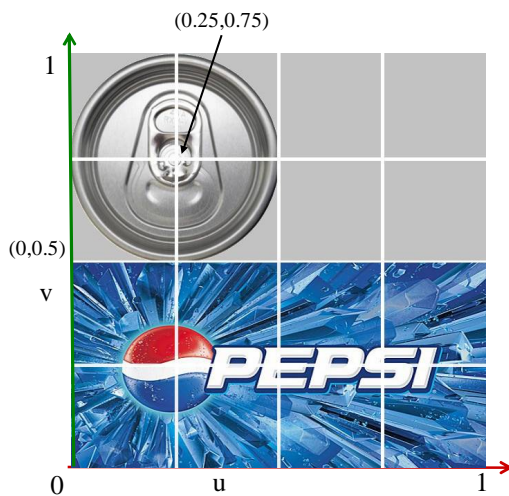
8. Texturas

- Trucos. Usando una única textura



8. Texturas

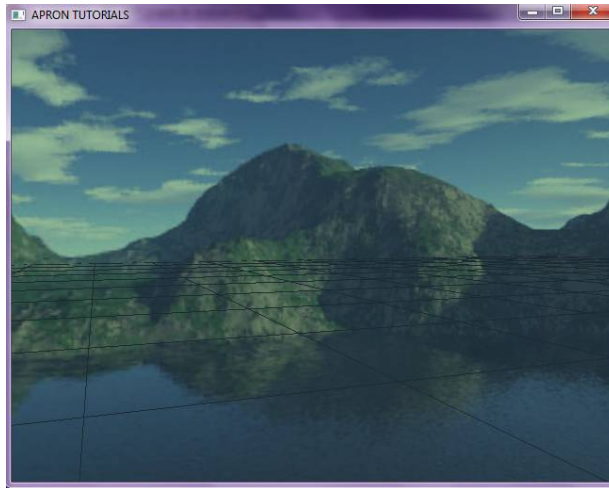
- Trucos. Usando una única textura



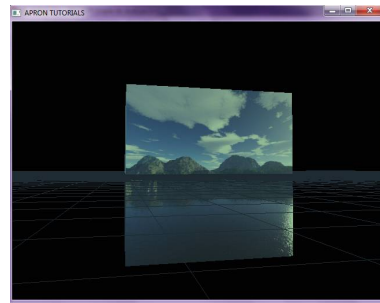
Introducción a OpenGL

8. Texturas

- Trucos. Skybox

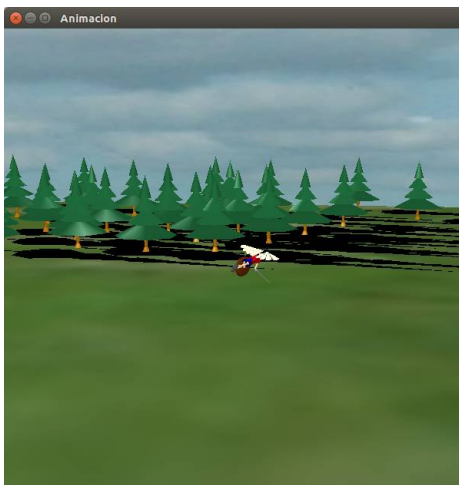


http://www.morrowland.com/apron/tut_gl.php



8. Texturas

- Trucos. Skybox



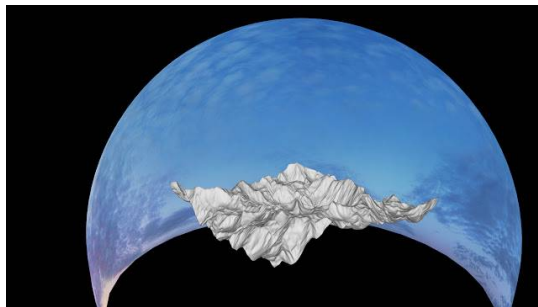
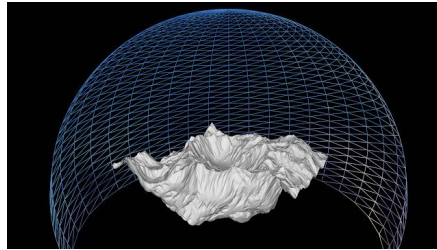
Juan Marfil Bustos
FIG curso 2006



Introducción a OpenGL

8. Texturas

- Trucos. Skysphere



OpenGL 129

Introducción a OpenGL

9. Selección

- Pasos para realizar una selección
 - Indicar un “buffer de selección” que será el que contenga el resultado de la selección. Se usa `glSelectBuffer()`.
 - Entrar en el modo selección con `glRenderMode(GL_SELECT)`.
 - Inicializar la pila de nombres que designaran los elementos a seleccionar. Se usa `glInitNames()` y `glPushName()`.
 - Definir el volumen de vista para la selección, de modo que haya una pequeña ventana de selección (mínimo un píxel).
 - Dibujar la escena asignando nombres a los elementos que se quieran seleccionar,
 - Salir del modo de selección y procesar el “buffer de selección”.

OpenGL 130

Introducción a OpenGL

9. Selección

- El “buffer de selección” es un vector de números naturales y se crea antes de entrar al modo de selección
- Puede ser local a un procedimiento o global

```
#define tam_buffer 100
...
void pick(...)
{
    GLuint selectBuf[tam_buffer]={0};
    ...
    // declarar buffer de selección
    glSelectBuffer(tam_buffer, selectBuf);
    ...
}
```



Introducción a OpenGL

9. Selección

- OpenGL tiene tres modos de funcionamiento:
 - **GL_RENDER** es el modo por defecto
 - **GL_SELECT** devuelve los nombres o identificadores de los elementos seleccionados
 - **GL_FEEDBACK** devuelve información geométrica de los elementos dibujados

`void glRenderMode(GLenum modo)` donde `modo` es **GL_RENDER**, **GL_SELECT**, **GL_FEEDBACK**

```
void pick(...)
{
    GLint hits;
    ...
    // activar selección
    glRenderMode(GL_SELECT);
    // hacer selección
    ...
    // desactivar selección
    hits=glRenderMode(GL_RENDER);
    ...
}
```



9. Selección

- Inicializar la pila de nombres

`void glInitNames(void)` borra la pila de nombres o identificadores para la selección, dejándola vacía

`void glPushName(GLint id)` donde `id` es el nombre asignado a un elemento de la escena. El valor inicial debe ser cero

```
void pick (...)  
...  
glRenderMode (GL_SELECT);  
// inicializar pila de nombres  
glInitNames();  
glPushName(0);  
...  
}
```

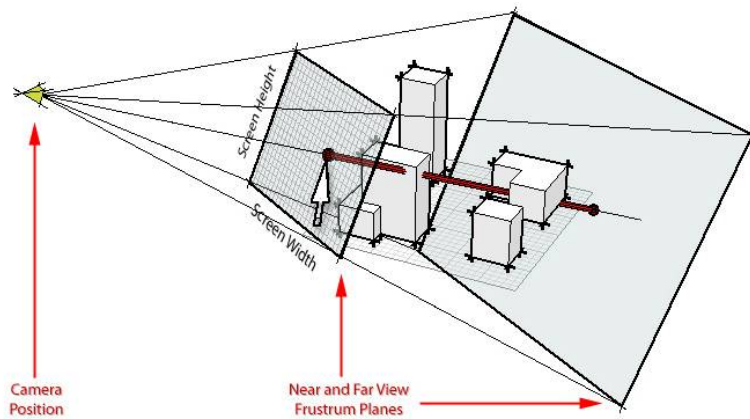
9. Selección

- Definir el volumen de vista para la selección
 - Se tiene que calcular una matriz de proyección restringida a una pequeña región del viewport alrededor del cursor usado en la selección
 - En el modo de selección solo se hace el dibujado de la escena para esta pequeña región del viewport
 - Esta matriz de proyección se llama matriz de selección (picking)
 - En la selección se determina que elementos gráficos están más cerca del cursor
 - Se realiza la transformación de coordenadas de dispositivo (región del viewport) a coordenadas de vista (región de selección)
 - Se usa la función `gluPickMatrix()` para calcular la matriz de selección

Introducción a OpenGL

9. Selección

- Definir el volumen de vista para la selección



<http://andrewmarsh.com/blog/2011/12/04/gluunproject-p3d-and-opengl-sketches>



Introducción a OpenGL

9. Selección

- Definir el volumen de vista para la selección

`void gluPickMatrix(GLdouble X, GLdouble Y, GLdouble ancho, GLdouble alto, GLint viewport[4])`
esta función calcula la matriz de selección que se multiplica por la matriz almacenada en la pila `GL_PROJECTION`, donde:

`X, Y` es el centro de la región de selección, expresado en coordenadas de dispositivo,

`ancho, alto` tamaño en píxeles de la región de selección

`viewport[4]` datos del viewport

```
void pick(...)
{
    GLint Viewport[4];
    ...
    // obtener datos viewport
    glGetIntegerv (GL_VIEWPORT, Viewport);
    ...
    // crear matriz de selección
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    gluPickMatrix ( x, Viewport[3] - y, ancho, alto, Viewport);
    glFrustum(Min_x,Max_y,Min_y,Max_y,Front_plane,Back_plane);
    ...
}
```



9. Selección

- Dibujar la escena asignando nombres
 - OpenGL usa una pila de números naturales como nombres para los elementos de una escena
 - Dos elementos gráficos tienen nombre distintos si el contenido de la pila es distinto

`void glLoadName(GLuint iden)` coloca en el tope de la pila el valor indicado por `iden`

`void glPushName(GLuint iden)` apila el nombre `iden`

`void glPopName(void)` desapila un nombre

`void glInitNames(void)` borra la pila de nombres



9. Selección

- Dibujar la escena asignando nombres
 - Identificando objetos

```
void dibujar_escena(int modo)
{
    ...
    If (modo==GL_SELECT) glLoadName(1);
    dibujar_objeto1();
    If (modo==GL_SELECT) glLoadName(2);
    dibujar_objeto2();
    ...
}
```

- Identificando elementos

```
...
for (i=0;i<n_caras;i++)
{
    glLoadName(i);
    glBegin(GL_TRIANGLES);
    glVertex3f();
    ...
    glEnd();
}
...
```



Introducción a OpenGL

9. Selección

- Dibujar la escena asignando nombres
 - Estableciendo una jerarquía con `glPushName` y `glLoadName`

```
// Dibujo de un tablero de ajedrez
...
for ( i=0;i<7;i++ )
{glPushName(i);
 glTranslatef(1.0,0.0,0.0);
 glPushMatrix();
  for (j=0;j<7;j++)
  {glLoadName(j);
   glTranslatef(0.0,1.0,0.0);
   pintar_casilla();
  }
 glPopMatrix();
 glPopMatrix();
}
```



Introducción a OpenGL

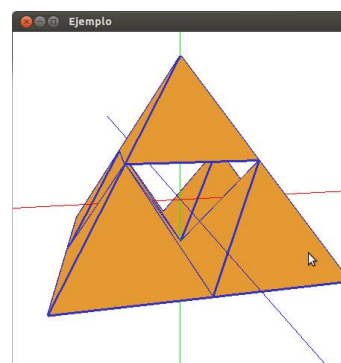
9. Selección

- Ejemplo

```
void draw_objects(int modo)
{
 glTranslatef(0.0,-0.4,0.0);
 glPushMatrix();
 glTranslatef(0.0,0.8,0.0);
 if (modo==GL_SELECT) glLoadName(1);
 draw_solido(piramide1,0.2,0.2,0.8,1);
 // se cambia el color cuando se selecciona
 draw_solido(piramide1,piramide1->r,piramide1->g,piramide1->b,2);
 glPopMatrix();

 ...

 glPushMatrix();
 glTranslatef(0.5,0.0,-0.5);
 if (modo==GL_SELECT) glLoadName(5);
 draw_solido(piramide5,0.2,0.2,0.8,1);
 draw_solido(piramide5,piramide5->r,piramide5->g,piramide5->b,2);
 glPopMatrix();
}
```



9. Selección

```
void pick (int x, int y)
{ GLuint selectBuf[100]={0};
  GLint viewport[4], hits=0;
  // Declarar buffer de selección
  glSelectBuffer(100, selectBuf);
  // Obtener los parámetros del viewport
  glGetIntegerv (GL_VIEWPORT, viewport);
  // Pasar OpenGL a modo selección
  glRenderMode (GL_SELECT);
  glInitNames(); glPushName(0);
  // Fijar la transformación de proyección para la selección
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  gluPickMatrix (x,(viewport[3] - y),5.0, 5.0, viewport);
  glFrustum(-Window_width,Window_width,-Window_height,Window_height,Front_plane,Back_plane);
  // Dibujar la escena
  draw_scene();
  // Pasar OpenGL a modo render
  hits = glRenderMode (GL_RENDER);
  // Restablecer la transformación de proyección
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity();
  glFrustum(-Window_width,Window_width,-Window_height,Window_height,Front_plane,Back_plane);
  // Procesar el contenido del buffer de selección
  procesar_hits(hits, selectBuf);
  // Dibujar la escena para actualizar cambios
  draw_scene(); }
```

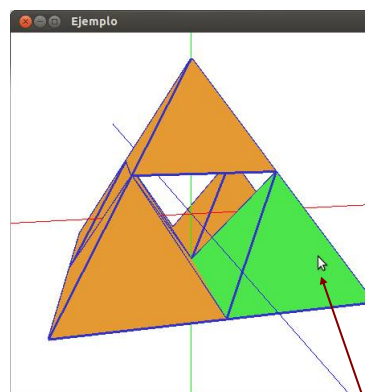


141

9. Selección

• Ejemplo

```
void procesar_hits(GLint hits, GLuint *names)
{
  ...
  // mostrar contenido de la pila
  printf("%d hits:\n", hits);
  for (i = 0; i < hits; i++)
    printf("Número: %d\n"
           "Min Z: %d\n"
           "Max Z: %d\n"
           "Nombre en la pila: %d\n",
           (GLubyte)names[i * 4],
           (GLubyte)names[i * 4 + 1],
           (GLubyte)names[i * 4 + 2],
           (GLubyte)names[i * 4 + 3]);
  printf("\n");
  switch (names[0+3])
  {case 1:
    ...
    // procesar el cambio de color
  }
  ...
}
```



```
1 hits:
Número: 1
Min Z: 0
Max Z: 0
Nombre en la pila: 3
```

Contenido de la pila de nombres
para la pirámide seleccionada

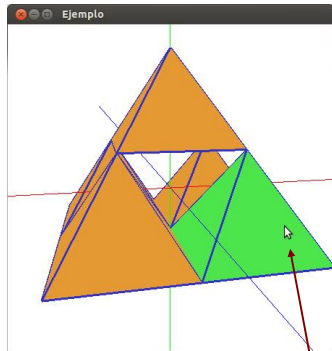
Posición cursor



142

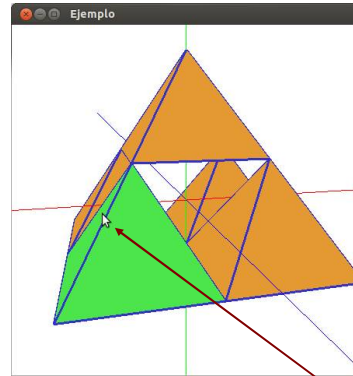
9. Selección

• Ejemplo



```
1 hits:
Número: 1
Min Z: 0
Max Z: 0
Nombre en la pila: 3
```

Posición cursor



```
2 hits:
Número: 1
Min Z: 0
Max Z: 0
Nombre en la pila: 2
Número: 1
Min Z: 0
Max Z: 0
Nombre en la pila: 4
```

Posición cursor

9. Selección

- Interpretar datos de la pila de nombres
 - Se almacena la información en bytes
 - Para cada elemento gráfico seleccionado tenemos:

Número de nombres en la pila	Z mínimo	Z máximo	nombre	nombre	...	nombre
------------------------------	----------	----------	--------	--------	-----	--------

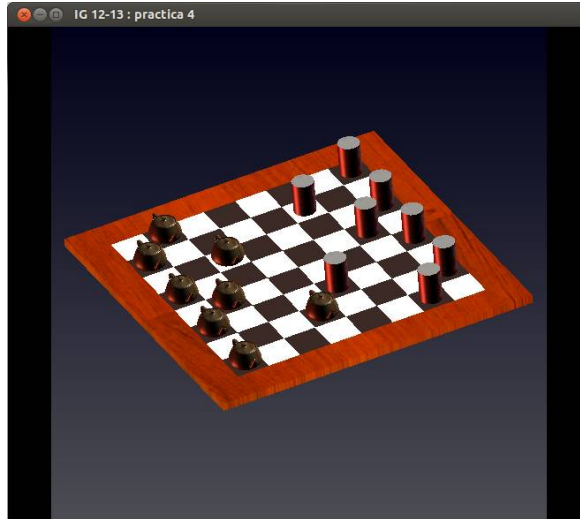
Ejemplo: dos elementos seleccionados, uno con una pila de nombres de tamaño 1 y el otro con pila de tamaño 4

1	Z mínimo	Z máximo	nombre1	4	Z mínimo	Z máximo	nombre1	nombre2	nombre3	nombre4
---	----------	----------	---------	---	----------	----------	---------	---------	---------	---------

Introducción a OpenGL

9. Selección

- Ejemplo

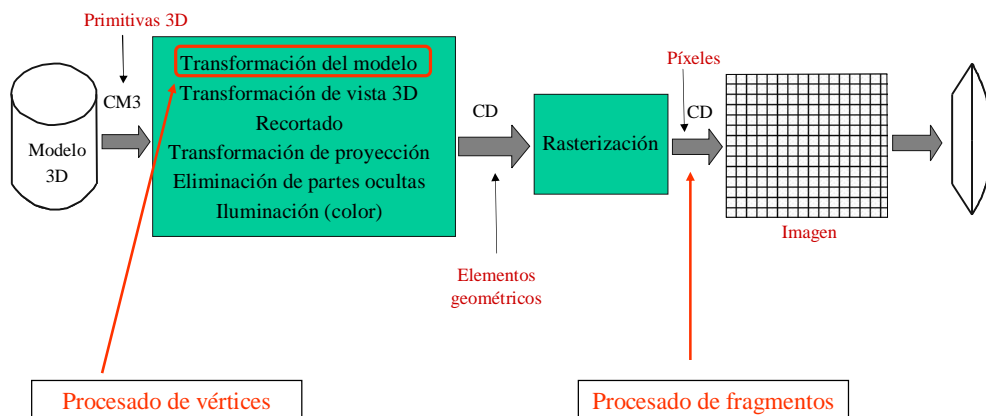


145

Introducción a OpenGL

10. Lenguaje de shading

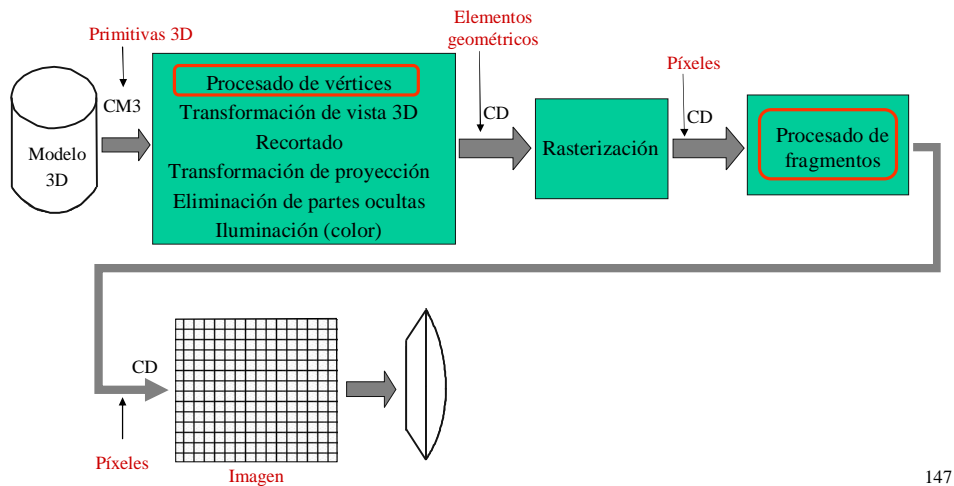
- Definición. Procesado gráfico fijo



146

10. Lenguaje de shading

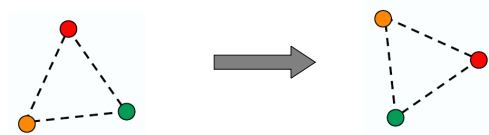
- Definición. Procesado gráfico programable



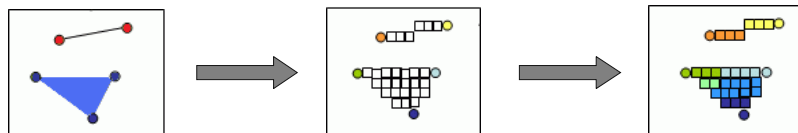
147

10. Lenguaje de shading

- Procesado de vértices o "Vertex Shader"



- Procesado de fragmentos o "Fragment Shader"

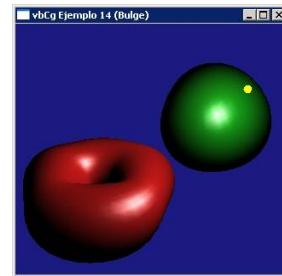


148

Introducción a OpenGL

10. Lenguaje de shading

- Vertex Shader

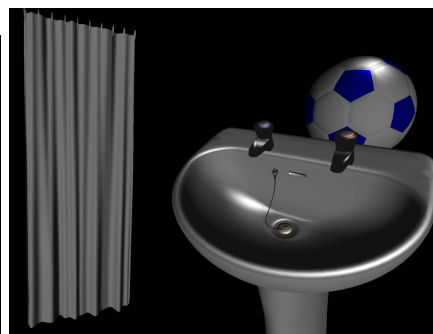


149

Introducción a OpenGL

10. Lenguaje de shading

- Fragment Shader



150

10. Lenguaje de shading

- Cg de Nvidia (histórico)
- GLSL OpenGL Shading Language
- HLSL High-Level Shader Language (usado con Direct3D)
- RSL Renderman Shading Language
- Houdini VEX (vector expression)

151

10. Lenguaje de shading

- GLSL “OpenGL Shading Language”. Aparece a partir de OpenGL 2.0
- Su especificación procede de OpenGL ARB “Architecture Review Board” (consorcio de empresas del que depende la evolución de OpenGL)
 - <http://www.kronos.org>
- Características de GLSL similares a un lenguaje de programación
 - Lenguaje basado en C++
 - Tipos de datos básicos y definidos por el usuario
 - Operadores, expresiones, estructuras, funciones, sentencias
 - Directivas al compilador

152

Introducción a OpenGL

10. Lenguaje de shading

- Tipos de datos

void, bool, int, uint, float

vec2 vector de dos números tipo **float**
vec3 vector de tres números tipo **float**
vec4 vector de cuatro números tipo **float**
bvec2 vector de dos números tipo **bool**
bvec3 vector de tres números tipo **bool**
bvec4 vector de cuatro números tipo **bool**
ivec2 vector de dos números tipo **int**
ivec3 vector de tres números tipo **int**
ivec4 vector de cuatro números tipo **int**
mat2 matriz 2x2 de números tipo **float**
mat3 matriz 3x3 de números tipo **float**
mat4 matriz 4x4 de números tipo **float**
mat2x2 como **mat2**
mat2x3 matriz 2x3 (columnasxfilas) tipo **float**
mat2x4 matriz 2x4 (columnasxfilas) tipo **float**

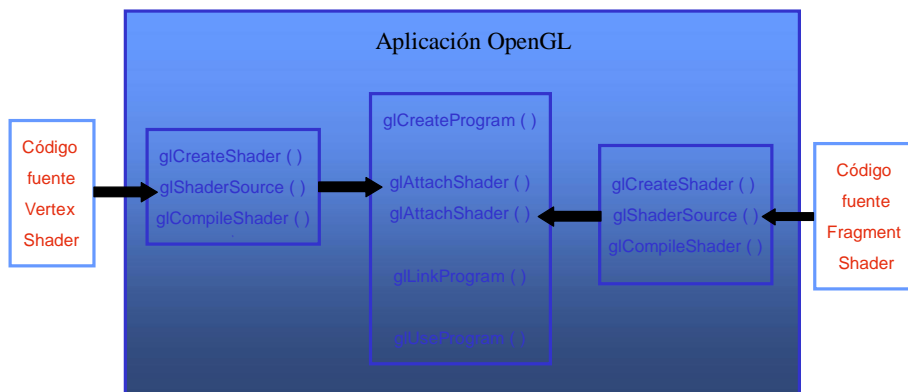
mat3x2 matriz 3x2 (columnasxfilas) tipo **float**
mat3x3 como **mat3**
mat3x4 matriz 3x4 (columnasxfilas) tipo **float**
mat4x2 matriz 4x2 (columnasxfilas) tipo **float**
mat4x3 matriz 4x3 (columnasxfilas) tipo **float**
mat4x4 como **mat4**
sampler1D enlace a una textura 1D
sampler2D enlace a una textura 2D
sampler3D enlace a una textura 3D
samplerCube enlace a una textura a aplicar a las caras de un cubo
sampler1DShadow
sampler2DShadow

153

Introducción a OpenGL

10. Lenguaje de shading

- Creación de shaders
- Se han de crear dos programas (**vertex** y **fragment shaders**) que se enlazan con una aplicación OpenGL



154

10. Lenguaje de shading

- Ejemplo (**Vertex Shader**) (ejemplo.vert) (versión GLSL hasta 1.3)

```
uniform float temperatura_menor;
uniform float temperatura_rango;

/* el calificativo attribute indica una variable que cambia para cada vértice */
attribute float vertex_temp;

/* el calificativo varying indica una variable accesible tanto para el vertex como el fragment
shader */
varying float temperatura;

void main ( )
{
    temperatura=(vertex_temp-temperatura_menor)/temperatura_rango;

    /* instrucción principal de un vertex shader: multiplica una matriz por los vértices */
    ➔ gl_Position=gl_ModelViewProjectionMatrix*gl_Vertex;
}
```

155

10. Lenguaje de shading

- Ejemplo (**Fragment Shader**)(ejemplo.frag)(versión GLSL hasta 1.3)

```
precision highp float;
uniform vec3 color_mas_frio;
uniform vec3 color_mas_calido;

/* el calificativo varying indica una variable accesible tanto para el vertex
como el fragment shader */
varying float temperatura;

void main ( ) {
    /* interpolación del color según el valor de la temperatura */
    vec3 color=mix(color_mas_frio, color_mas_calido, temperatura);

    /* función principal de un fragment shader: color aplicado a los
fragmentos */
    ➔ gl_FragColor=vec4(color, 1.0);
}
```

156

Introducción a OpenGL

10. Lenguaje de shading

- Ejemplo (aplicación OpenGL)

```
void Shaders( )
{
    const GLchar *vertexFile="ejemplo.vert",
    const GLchar *fragmentFile="ejemplo.frag";

    /* variables globales */
    GLuint prog=0;
    GLuint vertexShader;
    GLuint fragmentShader;

    vertexShader=glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(vertexShader, 1, &vertexFile, 0);
    glCompileShader(vertexShader);

    fragmentShader=glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(fragmentShader, 1, &fragmentFile, 0);
    glCompileShader(fragmentShader);

    prog=glCreateProgram( );
    glAttachShader(prog, vertexShader);
    glAttachShader(prog, fragmentShader);
    glLinkProgram(prog);
    glUseProgram(prog);
}
```

157

Introducción a OpenGL

10. Lenguaje de shading

- Ejemplo (aplicación OpenGL)

```
main(int argc, char **argv)
{
    char file_vert[]="ejemplo.vert",
    file_frag[]="ejemplo.frag";

    ...
    glutInit(&argc, argv);
    glutCreateWindow(" Shader ");

    readShaderSource(file_vert, &vertexFile); /* función para leer el shader */
    readShaderSource(file_frag, &fragmentFile);
    Shaders();

    ...
    glutReshapeFunc(reshape);
    glutDisplayFunc(display);
    glutMainLoop();

    return 0;
}
```

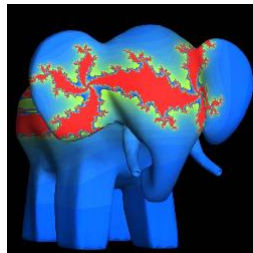
158

10. Lenguaje de shading

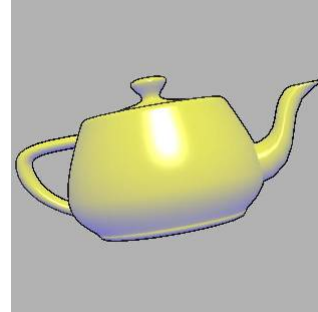
- Ejemplos de shaders



Difracción



Conjunto Julia



Shader de Amy y Bruce Gooch

159

10. Lenguaje de shading

- Entornos de desarrollo que Facilitan la creación de Shader
- AMD GPU PerfStudio
<http://developer.amd.com/tools-and-sdks/graphics-development/gpu-perfstudio/>
- Kick.js Shader Editor (editor online)
http://www.kickjs.org/example/shader_editor/shader_editor.html
- GeexLab y GLSL Hacker
<http://www.geeks3d.com/geexlab/>
<http://www.geeks3d.com/glslhacker/>

160

Introducción a OpenGL

10. Lenguaje de shading

- Ejemplo de diferencias entre GLSL 1.3 y GLSL 1.4

```
varying vec3 nor;
void main()
{
    vec4 v = vec4(gl_Vertex);
    nor = gl_NormalMatrix * gl_Normal;
    gl_Position = gl_ModelViewProjectionMatrix * v;
}
```

```
attribute vec3 vertex;
attribute vec3 normal;

uniform mat4 _mvProj; // matriz model-view-projectionx
uniform mat3 _norm; // matriz de normales

varying vec3 nor;
void main()
{
    nor = _norm * normal;
    gl_Position = _mvProj * vec4(vertex, 1.0);
}
```

161

Introducción a OpenGL

10. Lenguaje de shading

- Ejemplo de diferencias entre GLSL 1.3 y GLSL 1.4

```
varying vec3 nor;
void main()
{
    float intensity;
    vec4 color;
    vec3 n = normalize(nor);
    intensity = dot(vec3(gl_LightSource[0].position),n);
    if (intensity > 0.95) color = vec4(1.0,0.5,0.5,1.0);
    else if (intensity > 0.5) color = vec4(0.6,0.3,0.3,1.0);
    else if (intensity > 0.25) color = vec4(0.4,0.2,0.2,1.0);
    else color = vec4(0.2,0.1,0.1,1.0);
    gl_FragColor = color;
}
```

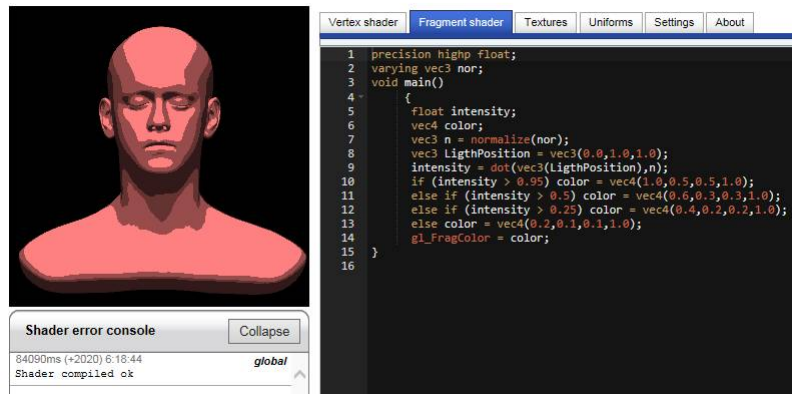
```
precision highp float;
varying vec3 nor;
void main()
{
    float intensity;
    vec4 color;
    vec3 n = normalize(nor);
    vec3 LigthPosition = vec3(0.0,1.0,1.0);
    intensity = dot(vec3(LigthPosition),n);
    if (intensity > 0.95) color = vec4(1.0,0.5,0.5,1.0);
    else if (intensity > 0.5) color = vec4(0.6,0.3,0.3,1.0);
    else if (intensity > 0.25) color = vec4(0.4,0.2,0.2,1.0);
    else color = vec4(0.2,0.1,0.1,1.0);
    gl_FragColor = color;
}
```

162

Introducción a OpenGL

10. Lenguaje de shading

- Ejemplo de shader con Kick.js Editor

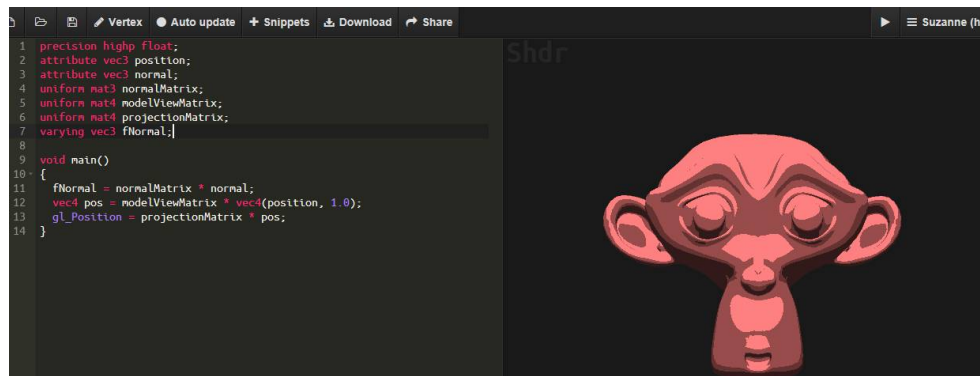


163

Introducción a OpenGL

10. Lenguaje de shading

- Ejemplo de shader con Shdr Editor



164