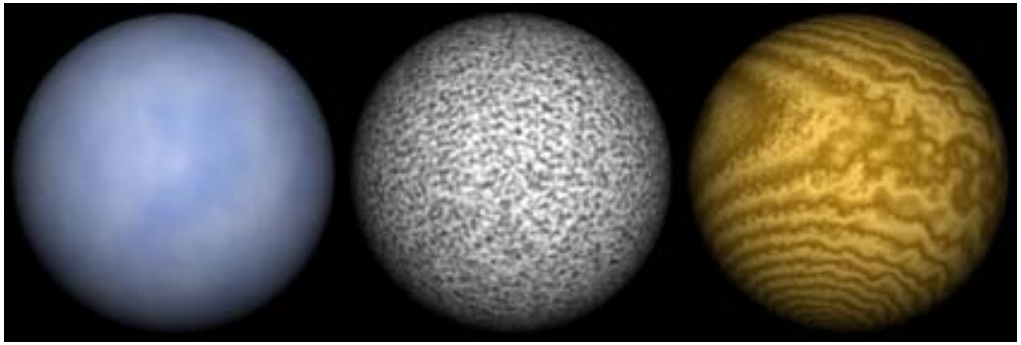


Tema 9: Texturas Avanzadas

J. Ribelles

SIE020: Síntesis de Imagen y Animación
Institute of New Imaging Technologies, Universitat Jaume I

Contenido

1. Introducción
2. Environment Mapping
3. Enrejado
4. Texturas 3D
5. Bump Mapping
6. Desplazamiento

Introducción

Introducción

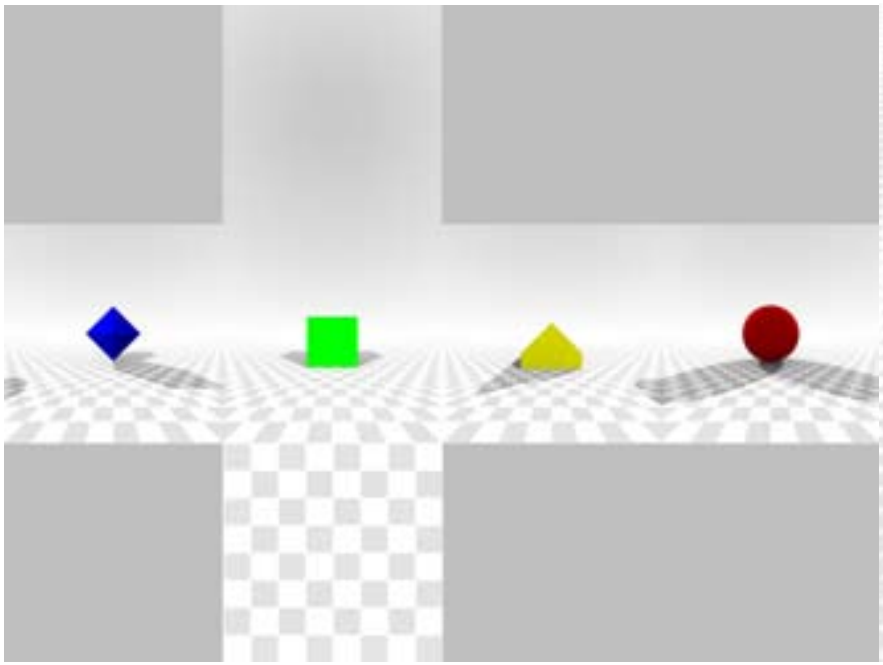
- La principal diferencia con lo visto hasta el momento en la asignatura reside en un nuevo concepto de textura más general.
- Ahora una textura se va a utilizar para modificar no sólo el color sino también el aspecto general de un objeto, incluyendo por ejemplo las normales y su geometría.

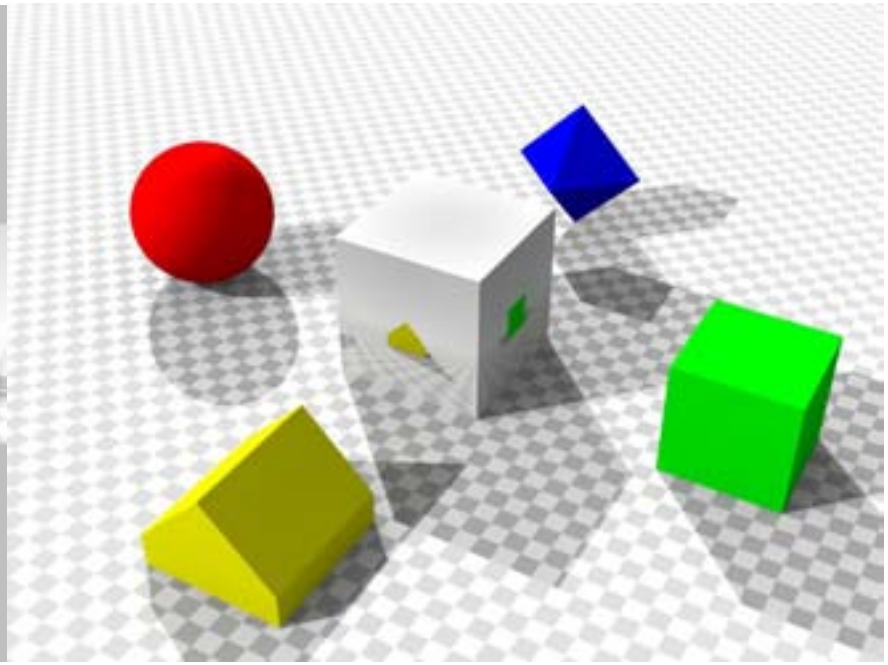
Environment Mapping

Descripción

- El objetivo es simular objetos que reflejan su entorno.
- Se utiliza una textura que contenga la escena que rodea al objeto.
- En tiempo de ejecución, se determinan las coordenadas de textura que dependen del vector dirección del observador o, mejor dicho, de su reflexión en cada punto de la superficie.
- Las coordenadas de textura cambian al moverse el observador.
- A la textura o conjunto de texturas que se utilizan para almacenar el entorno de un objeto se le denomina mapa de entorno.
- Este mapa puede estar formado por sólo una textura, al cual acceder utilizando coordenadas esféricas, o por un conjunto de seis texturas cuadradas formando un cubo.

Ejemplo





En OpenGL

- Para cada punto de la superficie del objeto reflejante se obtiene el vector de reflexión respecto a la normal en ese punto de la superficie.
- ¿Cuál de las seis texturas se ha de utilizar? Se elige la coordenada de mayor magnitud. Si es la coordenada x, se utilizan la cara derecha o izquierda del cubo, dependiendo del signo.
- Después, hay que obtener las coordenadas u y v para acceder a la textura seleccionada:

$$u = \frac{y + x}{2x} \qquad v = \frac{z + x}{2x} \tag{1}$$

- Este cálculo lo realiza en OpenGL la función *texture* cuando accede a una textura de tipo *samplerCube*.

El Shader

```
// Vertex Shader ----------------------------------------------------
uniform mat4  projection, camera, transf; // matrices
uniform mat3  normal;

in  vec3 posicion, vNormal;      // recibe vertice y normal
out vec3 reflexion;              // vector de reflexion

void main()
{
  vec4 ecPosition = camera * transf * vec4(posicion,1.0);
  vec3 N          = normalize(normal * vNormal);
  vec3 reflexion  = reflect(ecPosition.xyz, N);
  gl_Position     = projection * ecPosition;
}

// Fragment Shader --------------------------------------------------
uniform samplerCube mapa;

in  vec3 reflexion;
out vec4 colorFragmento;

void main()
{
  vec3 color = vec3(texture(mapa, reflexion));
  colorFragmento = vec4 (color, 1.0);
}
```

Creación del mapa de entorno

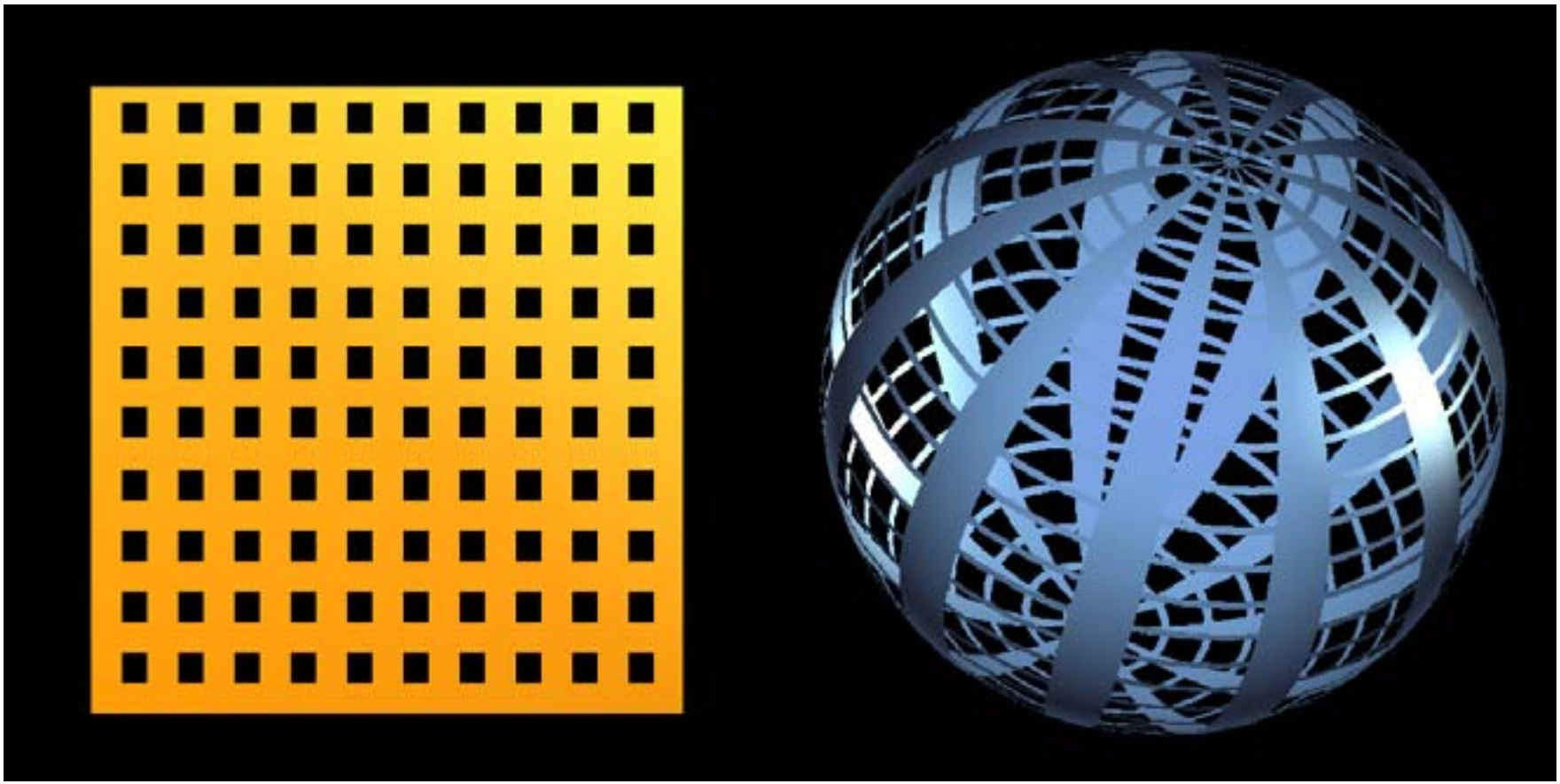
```
GLubyte *textura1= leeTextura("iquierda.rgb", 512, 512);
GLubyte *textura2= leeTextura("derecha.rgb",  512, 512);
GLubyte *textura3= leeTextura("arriba.rgb",   512, 512);
GLubyte *textura4= leeTextura("abajo.rgb",    512, 512);
GLubyte *textura5= leeTextura("delante.rgb",  512, 512);
GLubyte *textura6= leeTextura("detras.rgb",   512, 512);

GLuint nombre;
glGenTextures (1, &nombre);
glBindTexture (GL_TEXTURE_CUBE_MAP, nombre);

glTexImage2D (GL_TEXTURE_CUBE_MAP_POSITIVE_X, ..., textura1);
glTexImage2D (GL_TEXTURE_CUBE_MAP_NEGATIVE_X, ..., textura2);
glTexImage2D (GL_TEXTURE_CUBE_MAP_POSITIVE_Y, ..., textura3);
glTexImage2D (GL_TEXTURE_CUBE_MAP_NEGATIVE_Y, ..., textura4);
glTexImage2D (GL_TEXTURE_CUBE_MAP_POSITIVE_Z, ..., textura5);
glTexImage2D (GL_TEXTURE_CUBE_MAP_NEGATIVE_Z, ..., textura6);

glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_WRAP_R, GL_REPEAT);
glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

glActiveTexture (GL_TEXTURE0);
glBindTexture   (GL_TEXTURE_CUBE_MAP, nombre);
```

Enrejado

Descripción

- Es un tipo de textura que se califica como procedural.
- Una textura procedural es aquella que se obtiene mediante la ejecución de un procedimiento.
- En lugar de acceder a una imagen para obtener el color de un fragmento, ahora habrá un algoritmo que al ejecutarse nos porporcionará dicho valor.
- El enrejado consiste en utilizar la orden *discard* para eliminar fragmentos, produciendo en consecuencia agujeros en la superficie del objeto.
- Las coordenadas de textura se utilizan para controlar el tamaño del agujero y la repetición. En concreto, es la parte fraccional de las coordenadas de textura las que se utilizan.

Ejemplos



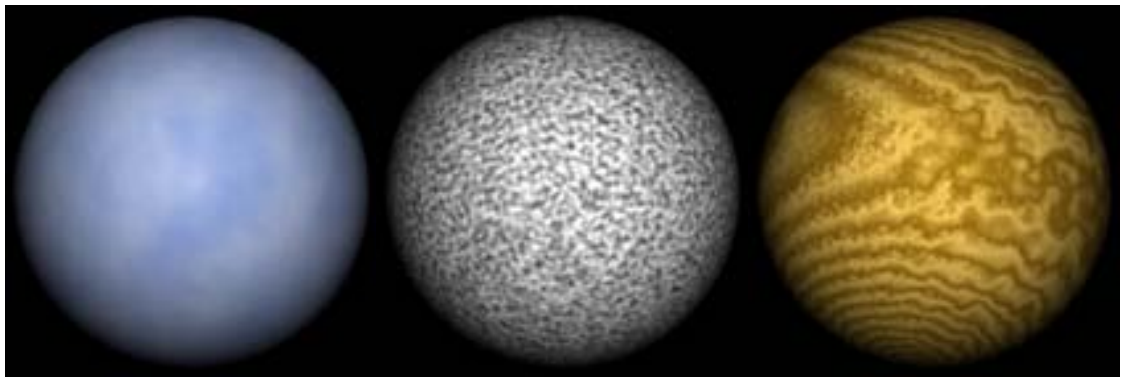
El Fragment Shader

```
in  vec3 color;
in  vec2 coordTextura;
out vec4 colorFragmento;

uniform vec2 nVeces;
uniform vec2 talla;

void main()
{
  float s = fract (coordTextura.s * nVeces.s);
  float t = fract (coordTextura.t * nVeces.t);

  if ((s > talla.s) && (t > talla.t)) discard;

  colorFragmento = vec4(color, 1.0);
}
```

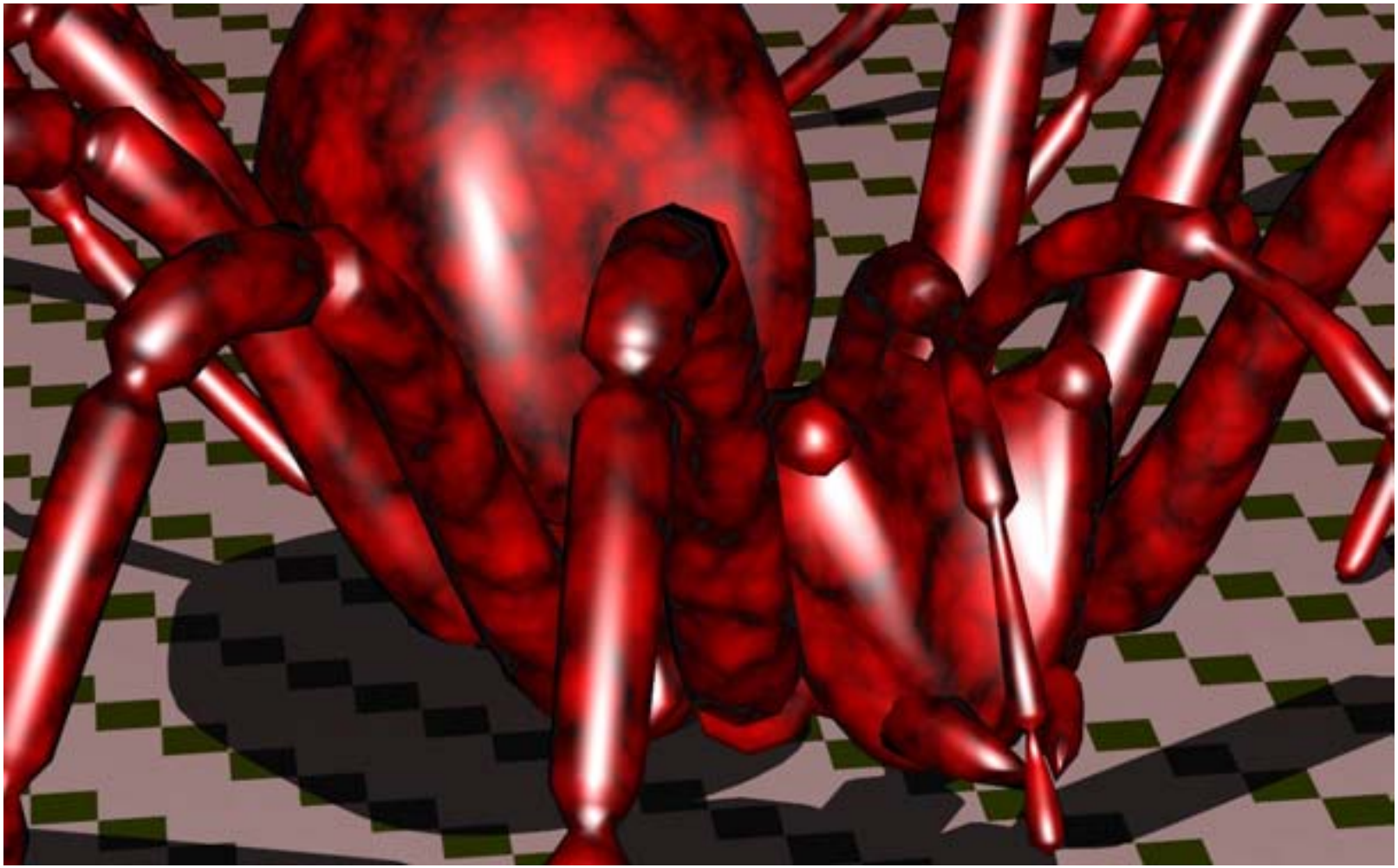
Texturas 3D

Descripción

- Una textura 3D define un valor para cada punto del espacio.
- Ideales para objetos que son creados a partir de un medio sólido como una talla de madera o un bloque de piedra.
- Vamos a utilizar una función de ruido para crear dicha textura.
- La función de ruido que nos interesa ha de producir siempre la misma salida para el mismo valor de entrada al mismo tiempo que aparentar aleatoriedad.



Ejemplo



Función de ruido

Opciones

- Una es utilizar la familia de funciones *noise* de GLSL.
- Otra es implementar una función de ruido propia en el *Shader*.
- Y la última es almacenar el resultado de la función de ruido y proporcionarlo en forma de textura al procesador gráfico.
- En el *shader* se accede a la textura través de un *sampler3D* utilizando tres coordenadas de textura. Estas coordenadas pueden ser las propias coordenadas geométricas del vértice.

```
vec4 valor = texture(madera, coordVertice * S1) * S2;
```

Creación de la Textura 3D

En OpenGL

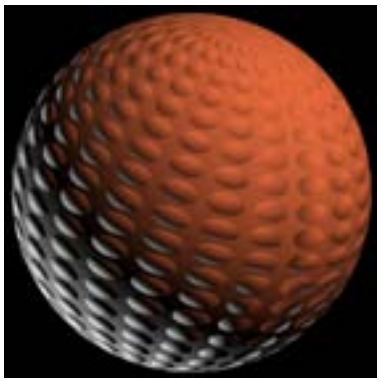
La textura 3D se especifica utilizando la función *glTexImage3D*. Igual que para una textura 2D, hay que crear un objeto textura, asignarlo a una unidad y activarla.

```
GLubyte *textura= leeTextura3D("datos.rgb", 512, 512, 512);

GLuint nombre;
glGenTextures (1, &nombre);

glBindTexture (GL_TEXTURE_3D, nombre);

glTexImage3D (GL_TEXTURE_3D, 0, GL_RGB, 512, 512, 512,0, GL_RGB, GL_UNSIGNED_BYTE, textura);
glTexParameteri (GL_TEXTURE_3D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri (GL_TEXTURE_3D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri (GL_TEXTURE_3D, GL_TEXTURE_WRAP_R, GL_REPEAT);
glTexParameteri (GL_TEXTURE_3D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri (GL_TEXTURE_3D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

glActiveTexture (GL_TEXTURE0);
glBindTexture   (GL_TEXTURE_3D, nombre);
```

Bump Mapping

Descripción

- Consiste en modificar la normal de la superficie para dar la ilusión de rugosidad.
- El cálculo de la variación de la normal se puede realizar en el propio *shader* utilizando un algoritmo.
- O también precalcularlo y proporcionárselo a la GPU como textura, conocida con el nombre de mapa de normales o *bump map*.



Desplazamiento

Descripción

- Consiste en aplicar un desplazamiento en cada vértice.
- El caso más sencillo es aplicarlo en la dirección de la normal.
- El desplazamiento se puede almacenar en una textura a la que se le conoce como mapa de desplazamiento.
- Desde el *vertex shader* se accede a la textura y se modifica la posición del vértice.