OpenGL

Texturas

Bibliografía:

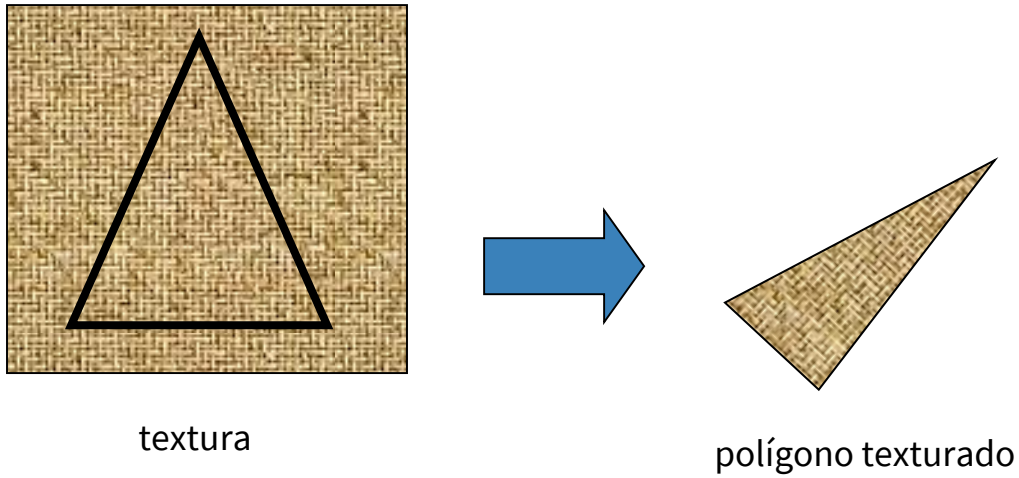
• Superbiblia 8ª ed., pp. 137-165

Introducción

- Las texturas permiten construir escenas más detalladas, sin tener que aumentar el número de polígonos
- La textura es un conjunto de *texels*, donde cada uno define un color, o color más alfa, o profundidad…
- Las texturas se pueden aplicar a cualquier tipo de primitiva OpenGL

Introducción

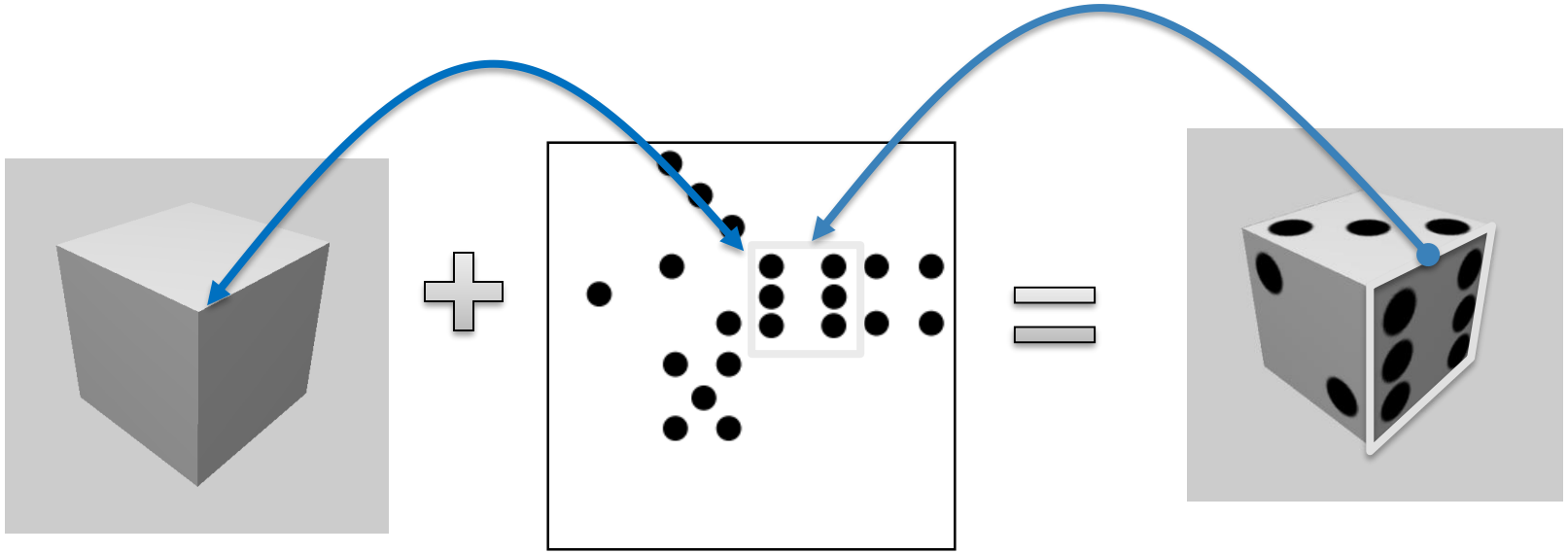
- Las texturas son normalmente imágenes bitmap que se “pegan” sobre los polígonos de un modelo
- OpenGL debe calcular cómo asignar la textura al polígono texturado, es decir, cómo hacer corresponder uno o más texels a cada píxel



textura

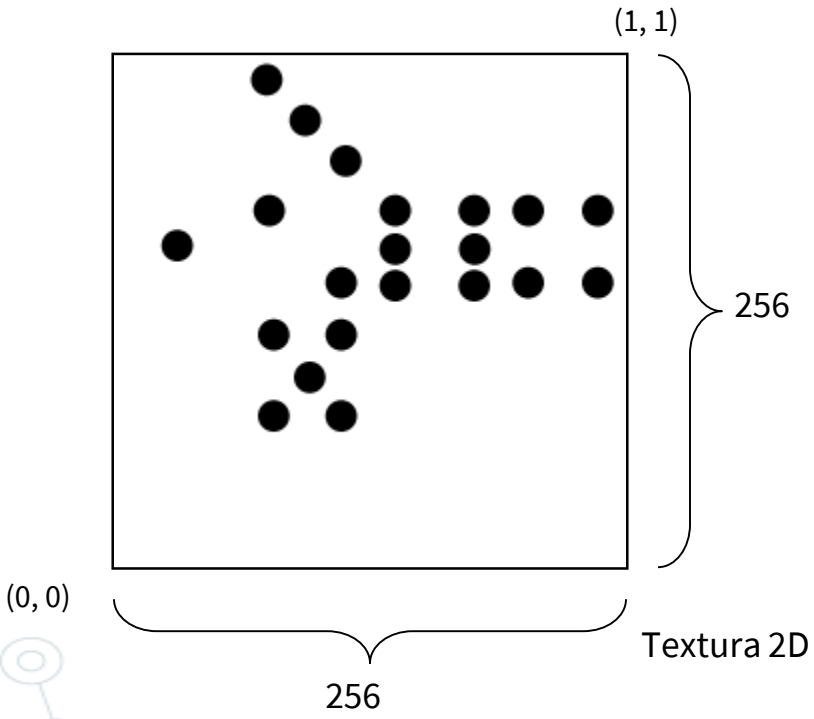
polígono texturado

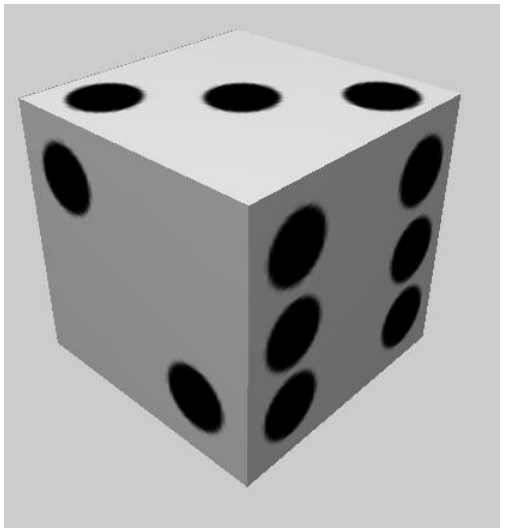
Introducción



Introducción

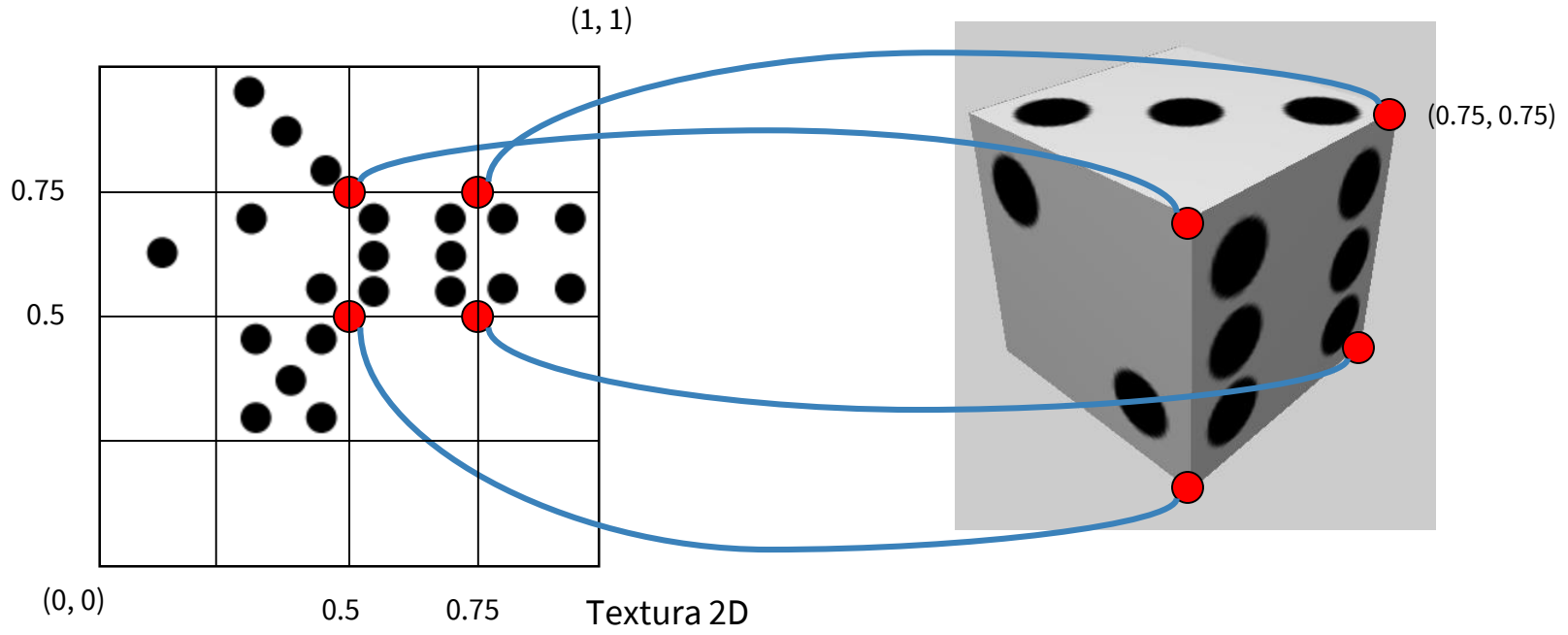
Asignando las coordenadas de textura a los vértices



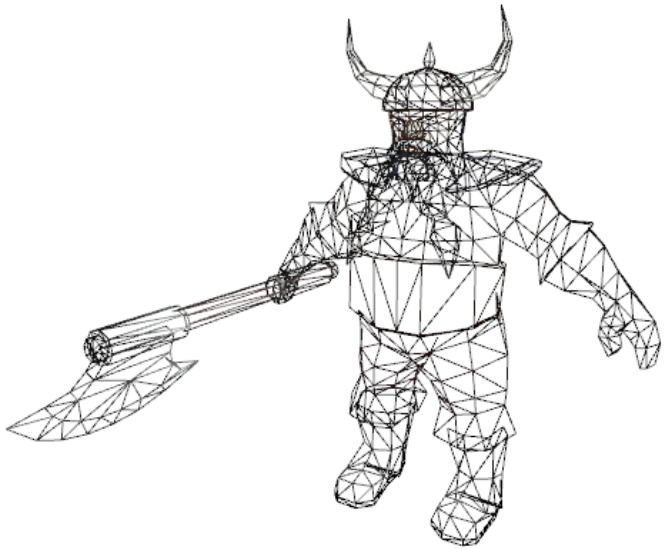


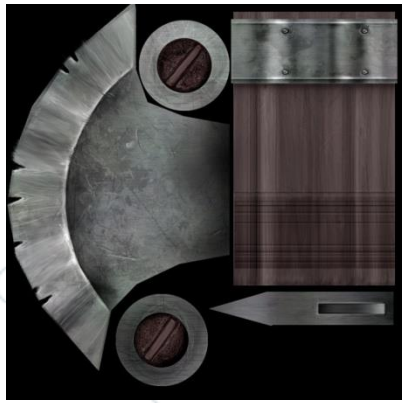
Introducción

Asignando las coordenadas de textura a los vértices



Introducción





Texturado básico

- Pasos (texturado básico):

// Inicialización

1. Crear un objeto textura y cargar sus texels
2. Configurar el acceso a la textura
3. Cargar los modelos indicando, además de la posición, normal y/o color, las coordenadas de textura de cada vértice

// Dibujado de un objeto

4. Vincular el objeto textura
5. Dibujar la escena,

Texturado básico

1. Crear un objeto textura y cargar su textura
   - Una textura puede ser:
     - 1D, 2D, 3D…
     - De 1 a 4 componentes por texel
     - Cada componente puede ser entera o real

(tamx-1, tamy-1)

0, 0

Textura 2D

Texturado básico

1. Crear un objeto textura y especificar su textura
   - Generar *n* identificadores de textura
     - **`void glGenTextures( GLsizei n, GLuint *textures )`**
   - Vincular el objeto textura
     - **`void glBindTexture(target, GLuint texture);`**
     - donde:
       - `target`: **`GL_TEXTURE_{1,2,3}D…`**
       - `texture`: identificador

Novedad en GL 4.5:

```
void glCreateTextures(GLenum target, GLsizei n, GLuint *textures);
```

Tipos de textura

- GL_TEXTURE_1D
- GL_TEXTURE_1D_ARRAY
- GL_TEXTURE_2D
- GL_TEXTURE_2D_ARRAY
- GL_TEXTURE_3D
- GL_TEXTURE_CUBE_MAP
- GL_TEXTURE_CUBE_MAP_ARRAY
- GL_TEXTURE_BUFFER
- GL_TEXTURE_2D_MULTISAMPLE
- GL_TEXTURE_2D_MULTISAMPLE_ARRAY
- GL_TEXTURE_RECTANGLE

Texturado básico

- Reserva de espacio y carga de una textura
- Antes:
  - Texturas mutables
    - `glTexImage*D`
- Ahora (a partir de OpenGL 4.2)
  - Texturas inmutables
    - `glTexStorage*D`
    - `glTexSubImage*D`

Texturado básico

- **`void glTexImage2D(GLenum target, GLint level, GLint internalformat, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid *texels );`**
  - Sustituye el nivel de la textura vinculada en “target” por el indicado
  - donde:
    - **`target`**: **`GL_TEXTURE_2D…`**
    - **`level`**: para pasar a OpenGL la textura a diferentes resoluciones. 0: textura a máxima resolución
    - **`internalformat`**: formato que debe usar OpenGL: **`GL_RED, GL_RGB`**, **`GL_RGBA, GL_DEPTH_COMPONENT, GL_DEPTH_STENCIL…`** (tablas 8.11-8.14 de la especificación OpenGL 4.6)
    - **`width`**, **`height`**: tamaño de la imagen
    - **`border`**: 0 siempre a partir de OpenGL 3
    - **`format`**: formato de la imagen: **`GL_RGB`**, **`GL_RGBA`**, **`GL_BGR`**, etc. (tabla 8.3 de la especificación OpenGL 4.6)
    - **`type`**: **`GL_BYTE`**, **`GL_UNSIGNED_BYTE`**, **`GL_SHORT, GL_FLOAT`**…
    - **`texels`**: puntero a los datos
  - ¡Ojo! Antes de OpenGL 2.0, *width* y *height* debían ser potencias de *2*.

Texturado básico

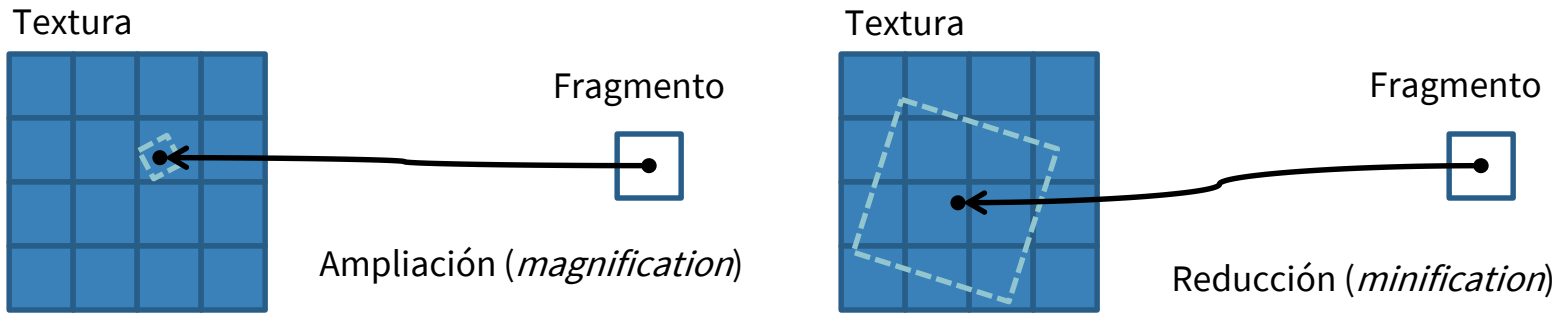
- **`void glTexStorage2D(GLenum target, GLsizei levels, GLint internalformat, GLsizei width, GLsizei height);`**
  - Crea una textura inmutable, con el espacio necesario para almacenar *levels* niveles de *mipmap*.
  - donde:
    - **`target`**: **`GL_TEXTURE_2D`…**
    - **`levels`**: reservar espacio para este número de niveles de mipmap
    - **`internalformat`**: formato que debe usar OpenGL: **`GL_RED`**, **`GL_RGB`**, **`GL_RGBA`**, **`GL_DEPTH_COMPONENT`**, **`GL_DEPTH_STENCIL`**…
    - **`width`**, **`height`**: tamaño de la imagen
    - Desde OpenGL 4.2

Texturado básico

- **`void glTexSubImage2D(GLenum target, GLint level, GLint xoffset, GLint yoffset, GLsizei width, GLsizei height, GLenum format, GLenum type, const GLvoid *texels );`**
  - Reemplaza los texels de una parte de (o de toda) la textura
  - donde:
    - **`target`**, **`level, width`**, **`height, format, type, texels`**: igual que para glTexImage2D
    - **`xoffset`**, **`yoffset`**: coordenadas del primer texel a reemplazar

Texturado básico

2. Configurar el acceso a la textura
   - Normalmente no habrá una relación uno a uno entre texels y píxels, por lo que hay que definir cómo asignar los unos a los otros



   - En ambos casos se pueden producir efectos de *aliasing*, por lo que se debe aplicar un filtrado

Filtrado de la textura

- Filtro de ampliación
  - `GL_NEAREST`: selecciona el color del texel más cercano a la posición del centro del píxel
  - `GL_LINEAR`: se calcula una media ponderada de los 2x2 texels más cercanos al centro del píxel

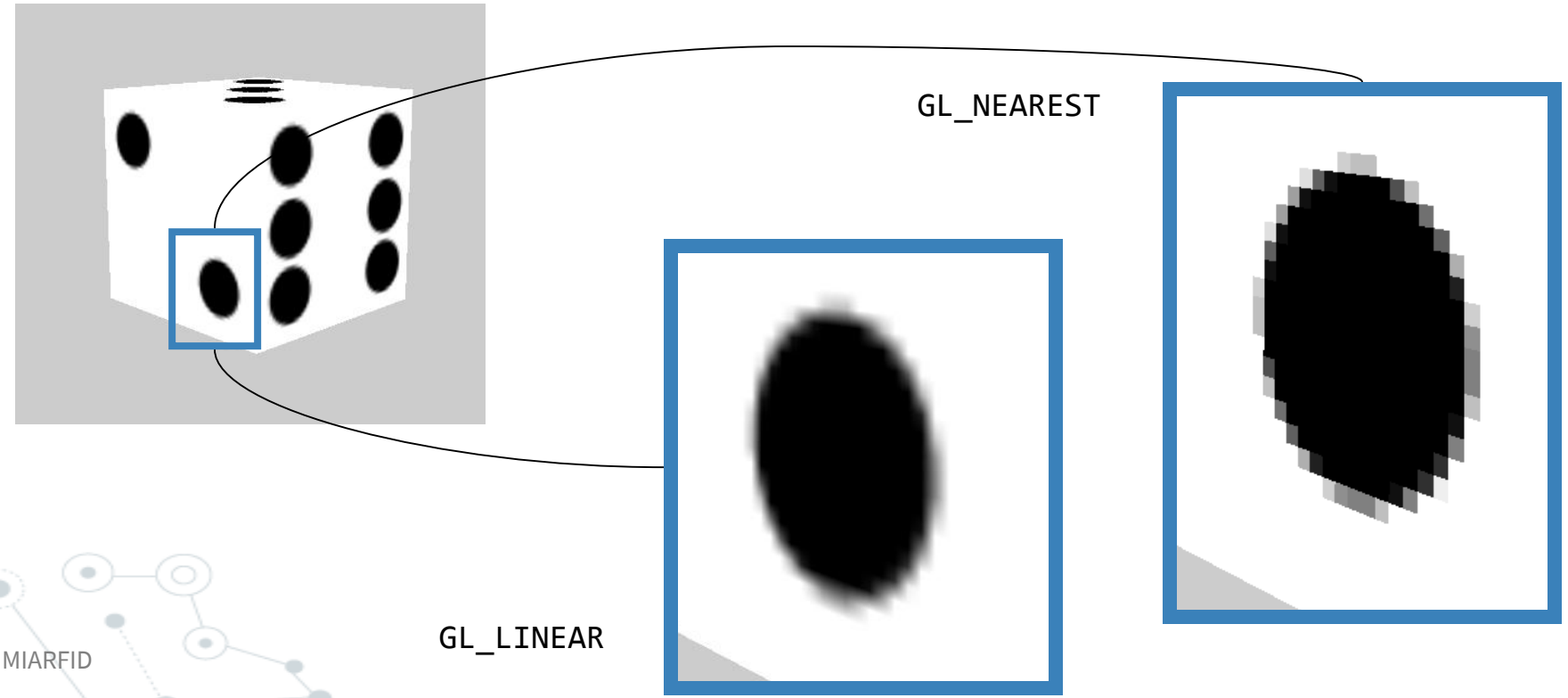
  - **`glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, <filtro>);`**

- Filtro de reducción
  - Se aplica cuando la textura es mayor que su proyección en pantalla
  - Las dos opciones anteriores y, además otros modos para controlar el *mipmapping*

  - **`glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, <filtro>);`**

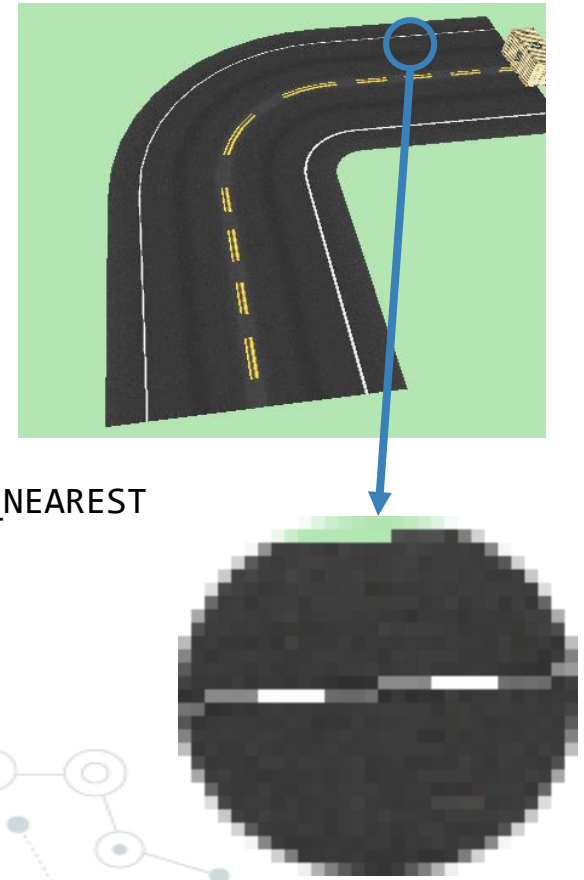
Filtrado de la textura

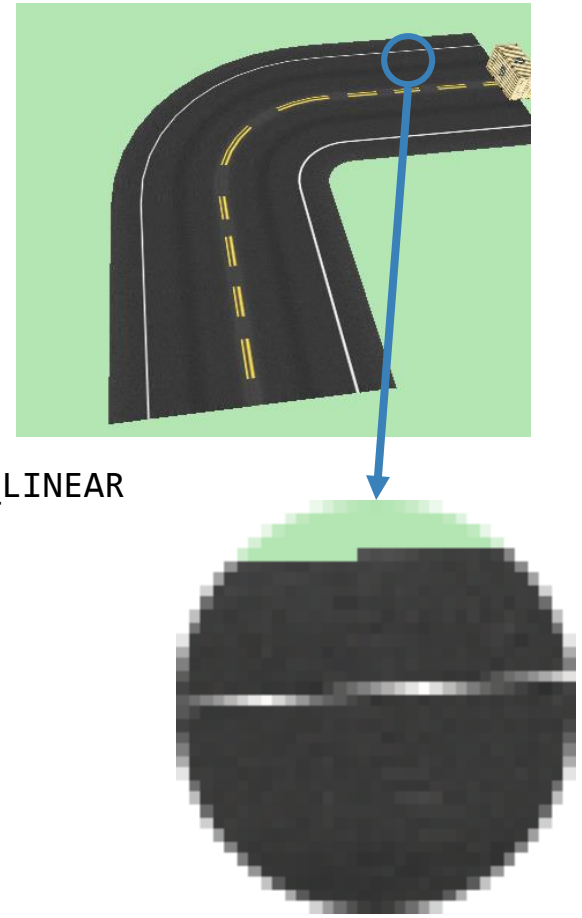
- Filtro de ampliación
  - Se aplica cuando la textura en pantalla es mayor que la textura original
  - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, <filtro>);`



Filtrado de la textura

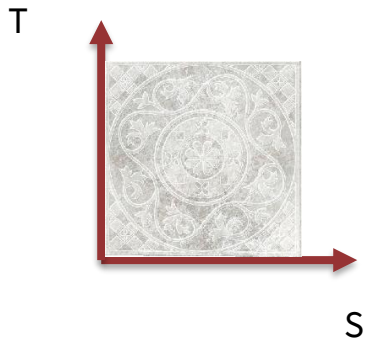
- Filtro de reducción

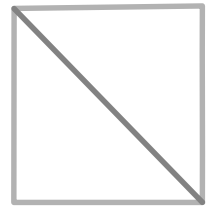




Repitiendo la textura

- Es común querer repetir la misma textura varias veces sobre un objeto
- Para ello se usan coordenadas de textura fuera del rango [0, 1]
  - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);`
  - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);`
- Se puede usar un comportamiento distinto en cada dirección:





La geometría se define como dos triángulos

Repitiendo la textura

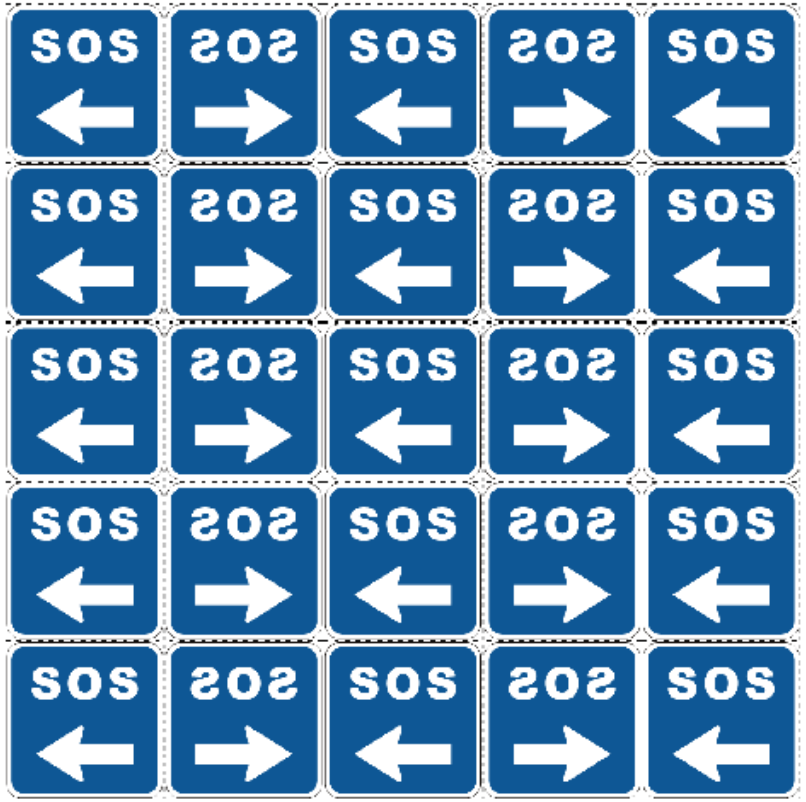
- Modos de repetición disponibles:
  - `GL_CLAMP_TO_EDGE`
  - `GL_CLAMP_TO_BORDER`
  - `GL_REPEAT`
  - `GL_MIRRORED_REPEAT`
  - `GL_MIRROR_CLAMP_TO_EDGE (GL 4.4)`

Repitiendo la textura

Proyecto ej3-2

Textura original

GL_TEXTURE_WRAP_T:
GL_REPEAT

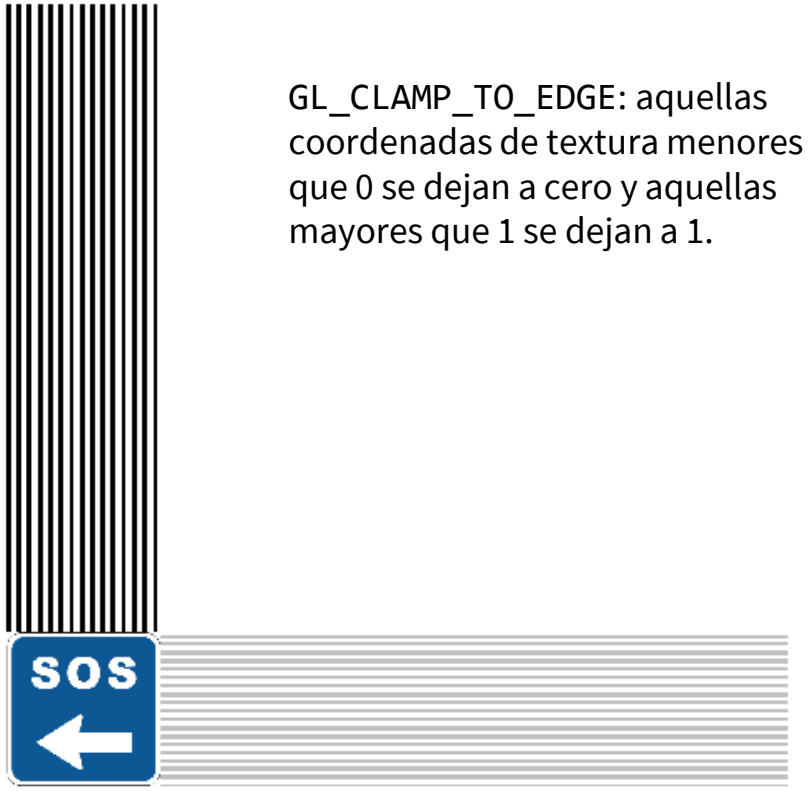


GL_TEXTURE_WRAP_S: GL_MIRRORED_REPEAT

Repitiendo la textura

Textura original

GL_TEXTURE_WRAP_T,
GL_CLAMP_TO_EDGE



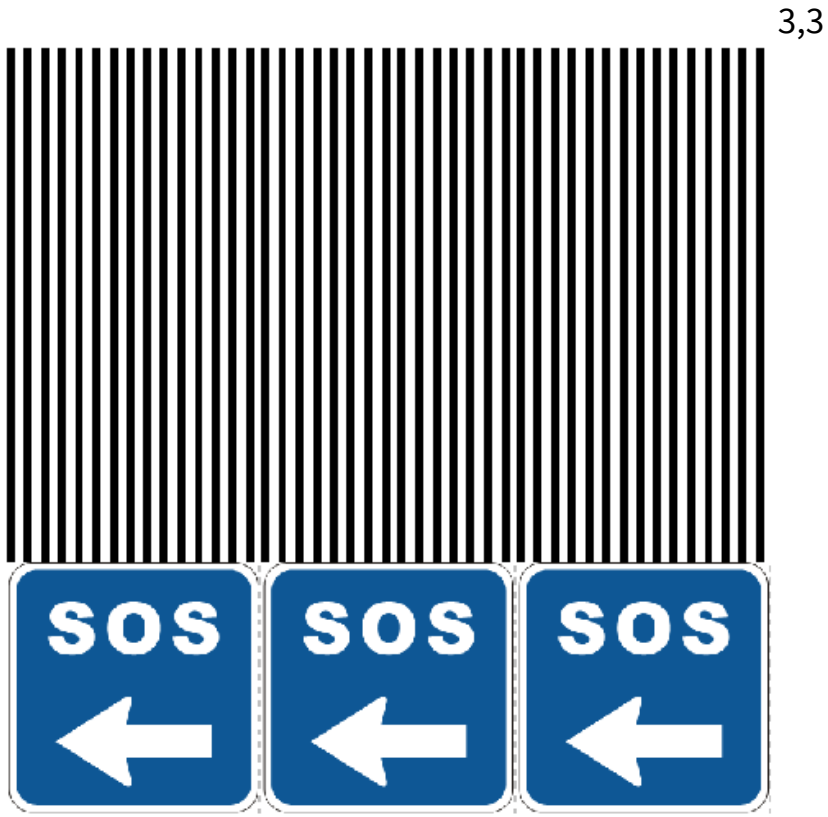
GL_CLAMP_TO_EDGE: aquellas coordenadas de textura menores que 0 se dejan a cero y aquellas mayores que 1 se dejan a 1.

GL_TEXTURE_WRAP_S: GL_CLAMP_TO_EDGE

Repitiendo la textura

Textura original

GL_TEXTURE_WRAP_T:
GL_CLAMP_TO_EDGE



GL_TEXTURE_WRAP_S: GL_REPEAT

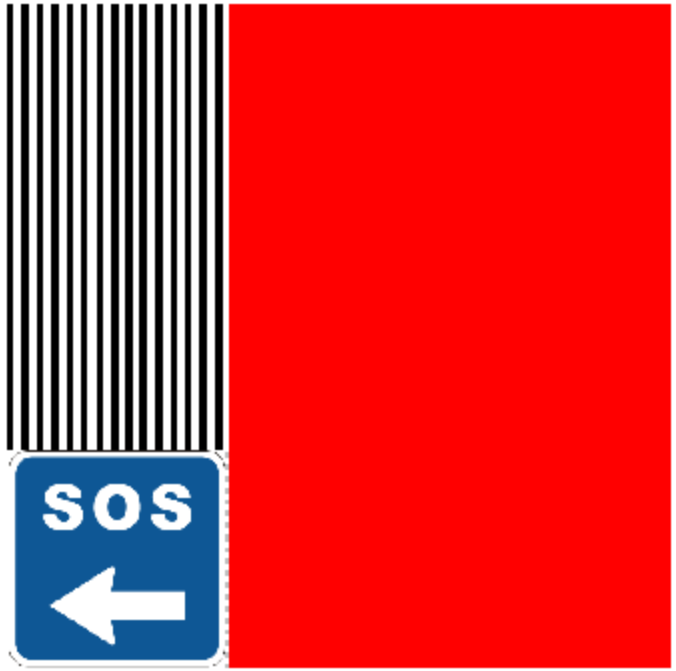
Repitiendo la textura

GL_TEXTURE_WRAP_T,
GL_CLAMP_TO_EDGE

GL_CLAMP_TO_BORDER: usa el color del borde de la textura para extenderla. Para establecer el color de borde:

```
static float bc[4]={1, 0, 0, 1};

glTexParameterfv(GL_TEXTURE_2D,
GL_TEXTURE_BORDER_COLOR, bc);
```



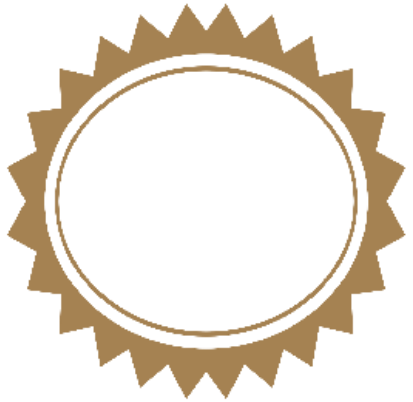
GL_TEXTURE_WRAP_S: GL_CLAMP_TO_BORDER

Repitiendo la textura

(2, 2)

```
GL_TEXTURE_WRAP_T,
GL_MIRROR_CLAMP_TO_EDGE
```

Textura original



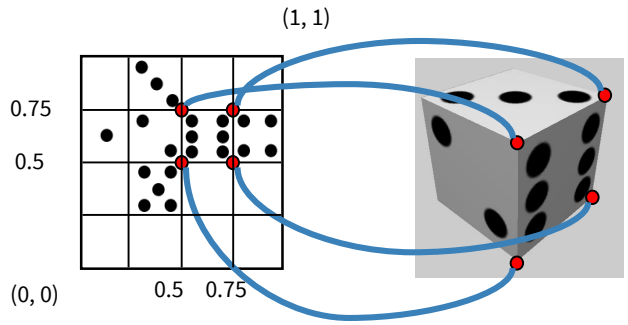
(-2, -2)

A partir de OpenGL 4.4

```
GL_TEXTURE_WRAP_S: GL_MIRROR_CLAMP_TO_EDGE
```

Texturado básico

3. Cargar los modelos indicando, además de la posición, normal y/o color, las coordenadas de textura de cada vértice



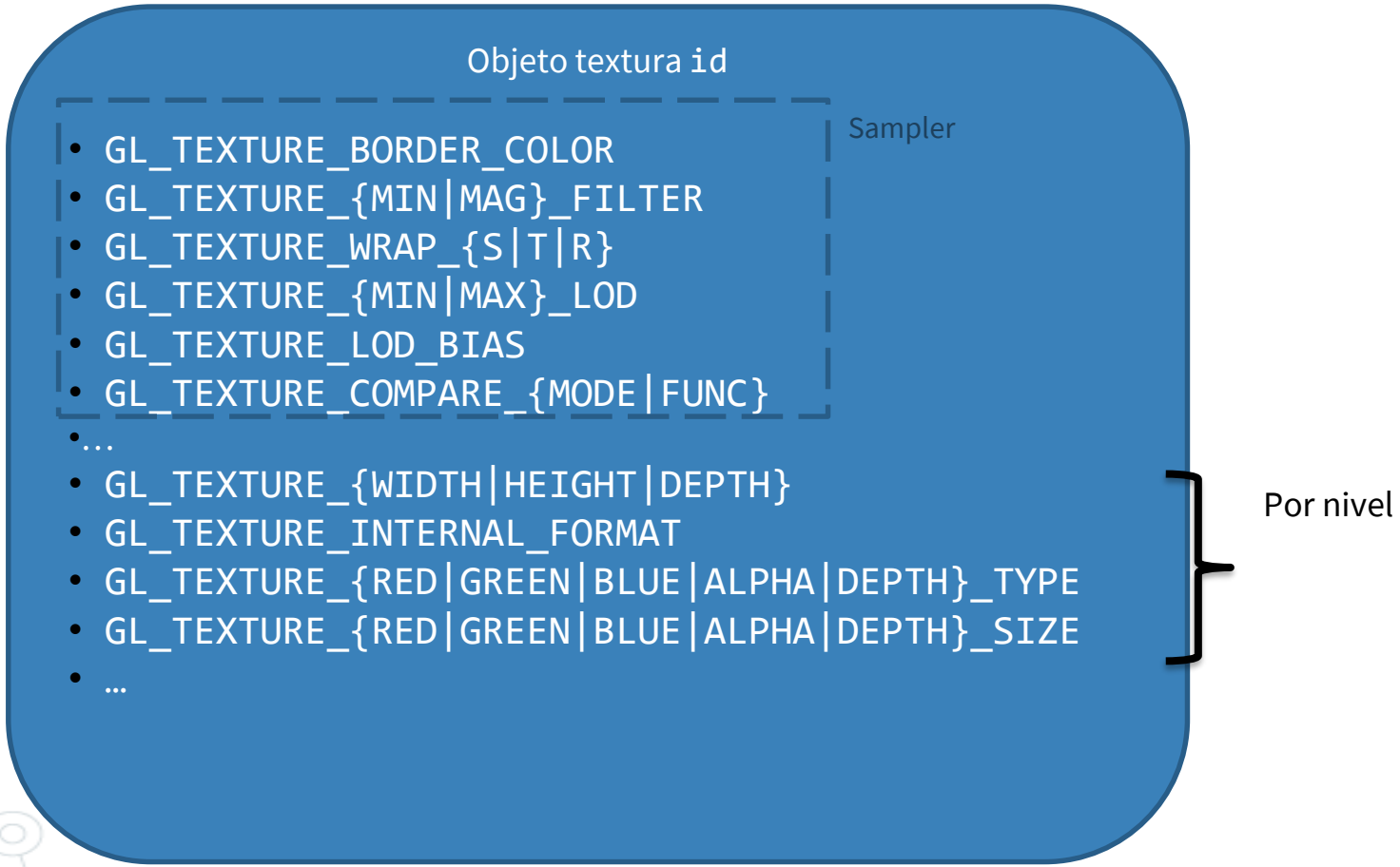
Las coordenadas de textura también se pueden generar en el *shader*

```
glm::vec2 tc[] = {
  glm::vec2(0.5f, 0.5f),
  glm::vec2(0.75f, 0.5f),
  glm::vec2(0.75f, 0.75f),
  glm::vec2(0.5f, 0.75f)
};
GLuint vbot;
glGenBuffers(1, &vbot);
glBindBuffer(GL_ARRAY_BUFFER, vbot);
glBufferData(GL_ARRAY_BUFFER, sizeof(tc), tc, GL_STATIC_DRAW);
glVertexAttribPointer(3, 2, GL_FLOAT, GL_FALSE, 0, (const void *)0);
glEnableVertexAttribArray(3);
// Establecer el resto de atributos de vértice
glDrawElements(…)
```

Objeto textura

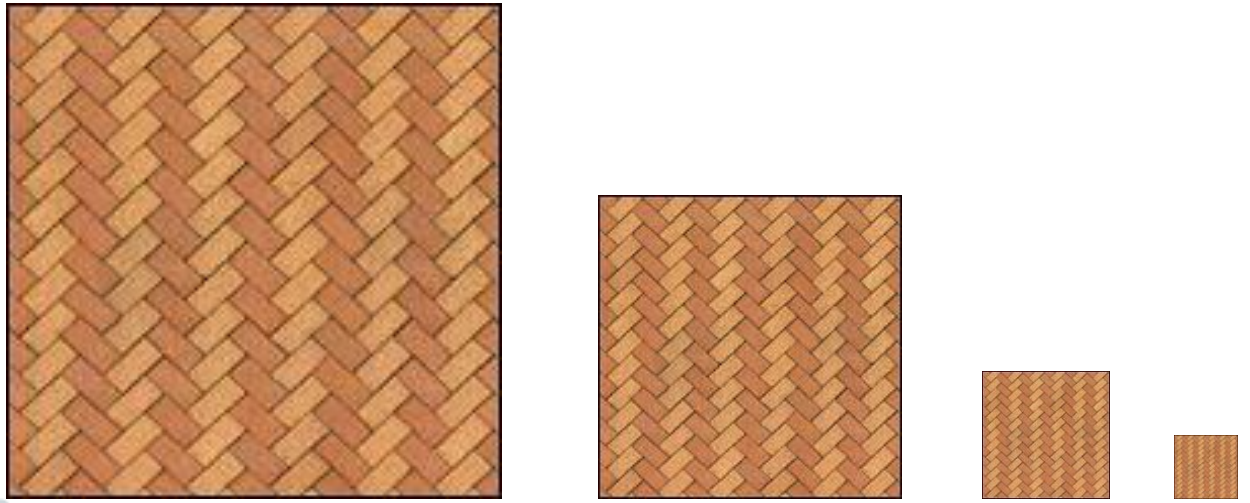
- Un objeto textura almacena la siguiente información:
  - La imagen (o imágenes) de la textura
  - Ancho, alto, formato interno, resolución de los componentes
  - Parámetros de aplicación de la textura:
    - Filtros de reducción/aumento
    - Modos de repetición de textura
    - Color del borde
    - Nivel de anisotropía
- Siempre que se active un objeto textura, se “recordarán” estos parámetros.
- Esto permite tener varias texturas preparadas y cambiar de una a otra rápidamente

Objeto textura



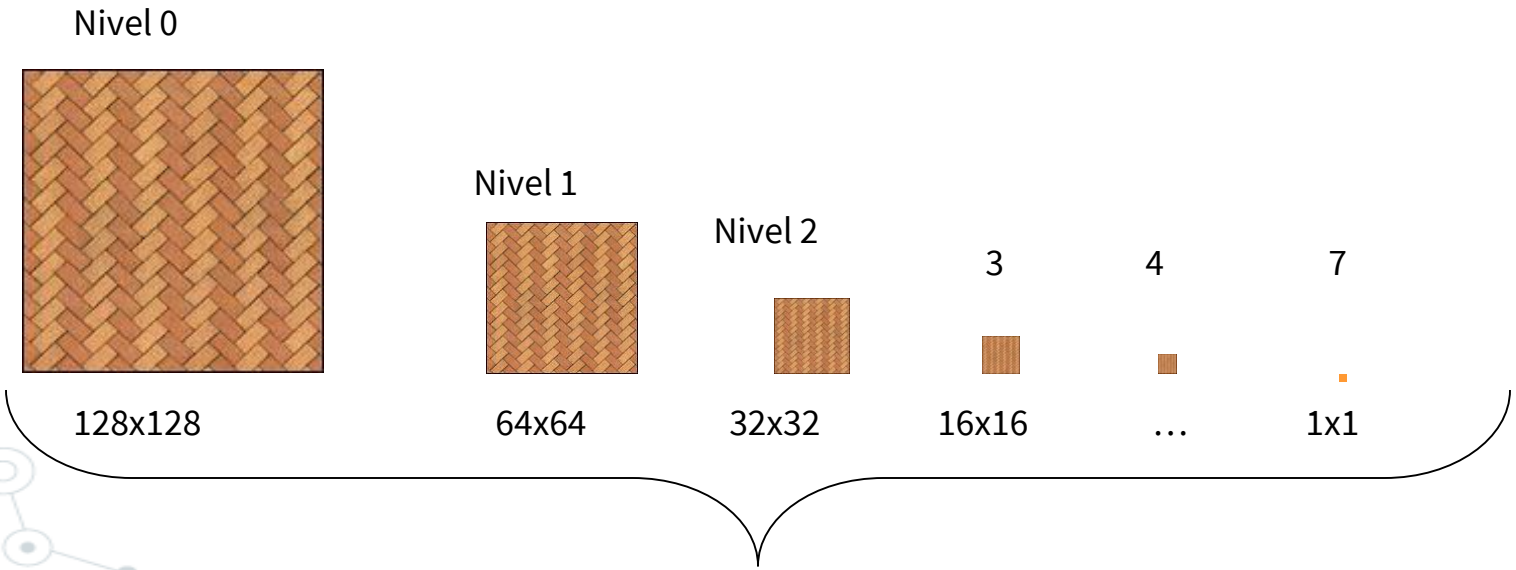
Mipmapping

- Al reducir la textura, incluso usando el filtro bilineal, únicamente se tienen en cuenta el color de 4 texels (aunque en un fragmento confluyan muchos más texels)
- El *mipmapping* es una técnica que permite reducir los artificios al precalcular un mejor filtrado de reducción, a costa de usar algo más de memoria
- Se precalculan distintos tamaños



Mipmapping

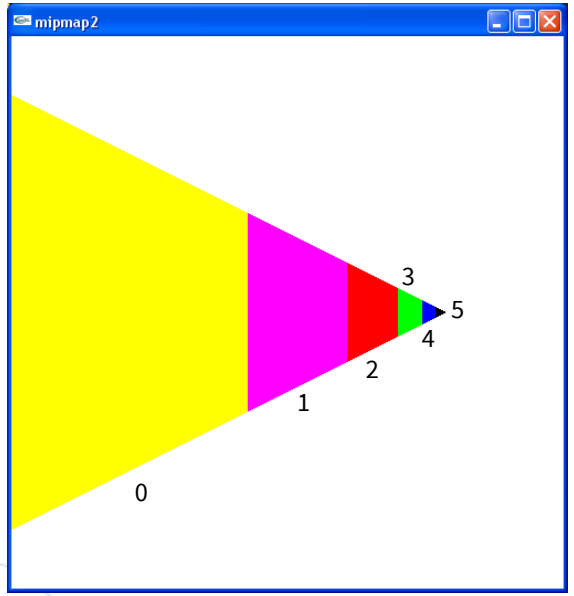
- Un mipmap está compuesto por las resoluciones desde el tamaño original hasta 1x1 píxel, donde cada nivel divide por 2 cada dimensión del anterior

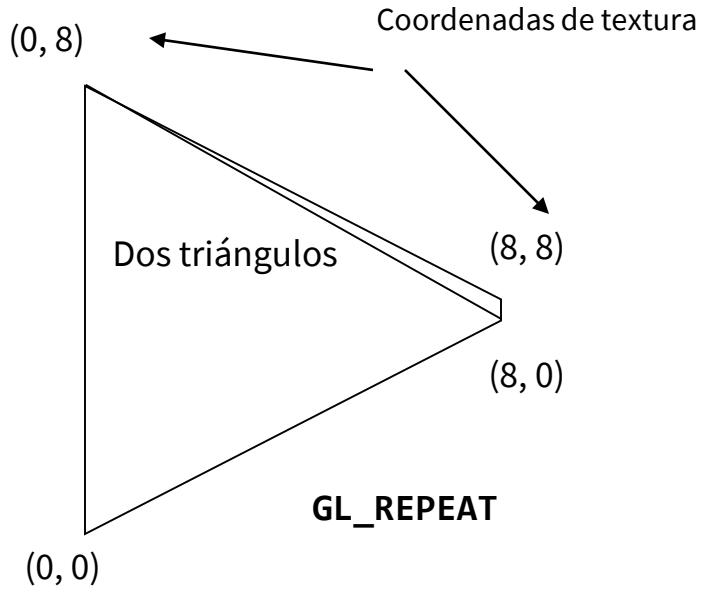


Construyendo *mipmaps*

- Opción 1 (larga)
  - Proporcionar uno por uno cada nivel a OpenGL
  - Esto implica tener que calcular el filtrado de cada nivel…
  - …pero da la flexibilidad de poder usar cualquier tipo de filtro…
  - e incluso que los niveles no tengan relación entre sí:

Ejemplo del *libro rojo*





Construyendo *mipmaps*

- Opción 1 (larga)
  - **`void glTexImage2D( GLenum` *`target`*`, GLint` *`level`*`, GLint` *`internalformat`*`, GLsizei` *`width`*`, GLsizei` *`height`*`, GLint` *`border`*`, GLenum` *`format`*`, GLenum` *`type`*`, const GLvoid *`*`texels` `);`**

  donde:
    - `target`, `internalformat`, `border`, `format`, `type`, `texels`: igual que antes
    - `level`: nivel del mipmap suministrado (0 el de mayor resolución)
    - `width`, `height`: tamaño del nivel suministrado

Construyendo *mipmaps*

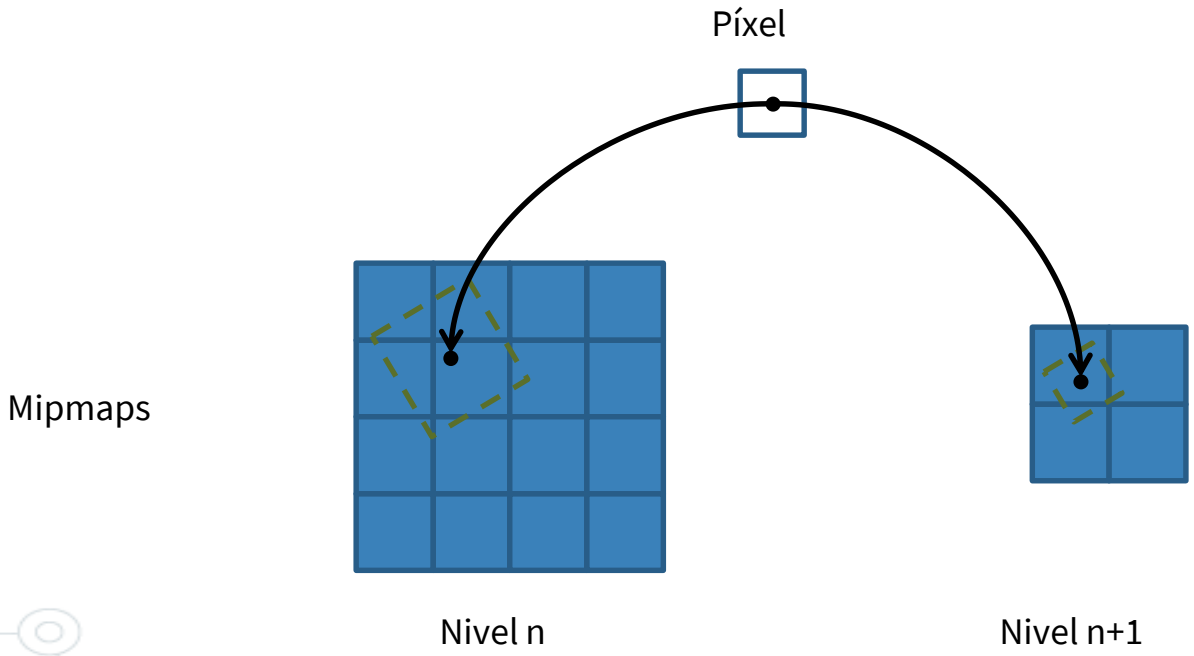
- Opción 2 (corta)
  - Usar **`glGenerateMipmap(GLenum target)`**
    - donde:
      - target: **`GL_TEXTURE_2D`**,…
    - Partiendo del nivel 0, genera el resto de niveles
    - No se asegura el tipo de filtro utilizado, y tiene una penalización en tiempo

Filtros y *mipmaps*

- OpenGL define 4 tipos de filtros de reducción para trabajar con mipmaps (`glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, <filtro>)`):
  - `GL_NEAREST_MIPMAP_NEAREST`, `GL_NEAREST_MIPMAP_LINEAR`, `GL_LINEAR_MIPMAP_NEAREST`, `GL_LINEAR_MIPMAP_LINEAR`
  - `GL_X_MIPMAP_Y`:
    - X: dentro de un nivel, usar NEAREST o LINEAR
    - Y: usar el nivel de mipmap más cercano (NEAREST) o usar los dos niveles más cercanos (LINEAR)
- Aunque una textura tenga definido un mipmap, si se usan los filtros `GL_NEAREST` o `GL_LINEAR`, sólo se usa la textura de nivel 0.
- Si se usa un filtro con mipmapping, pero no se ha construido uno, el objeto texturado aparecerá en negro

Filtros y *mipmaps*

- Interpolación entre dos niveles de mipmap (GL_X_MIPMAP_LINEAR)



Filtrado anisotrópico

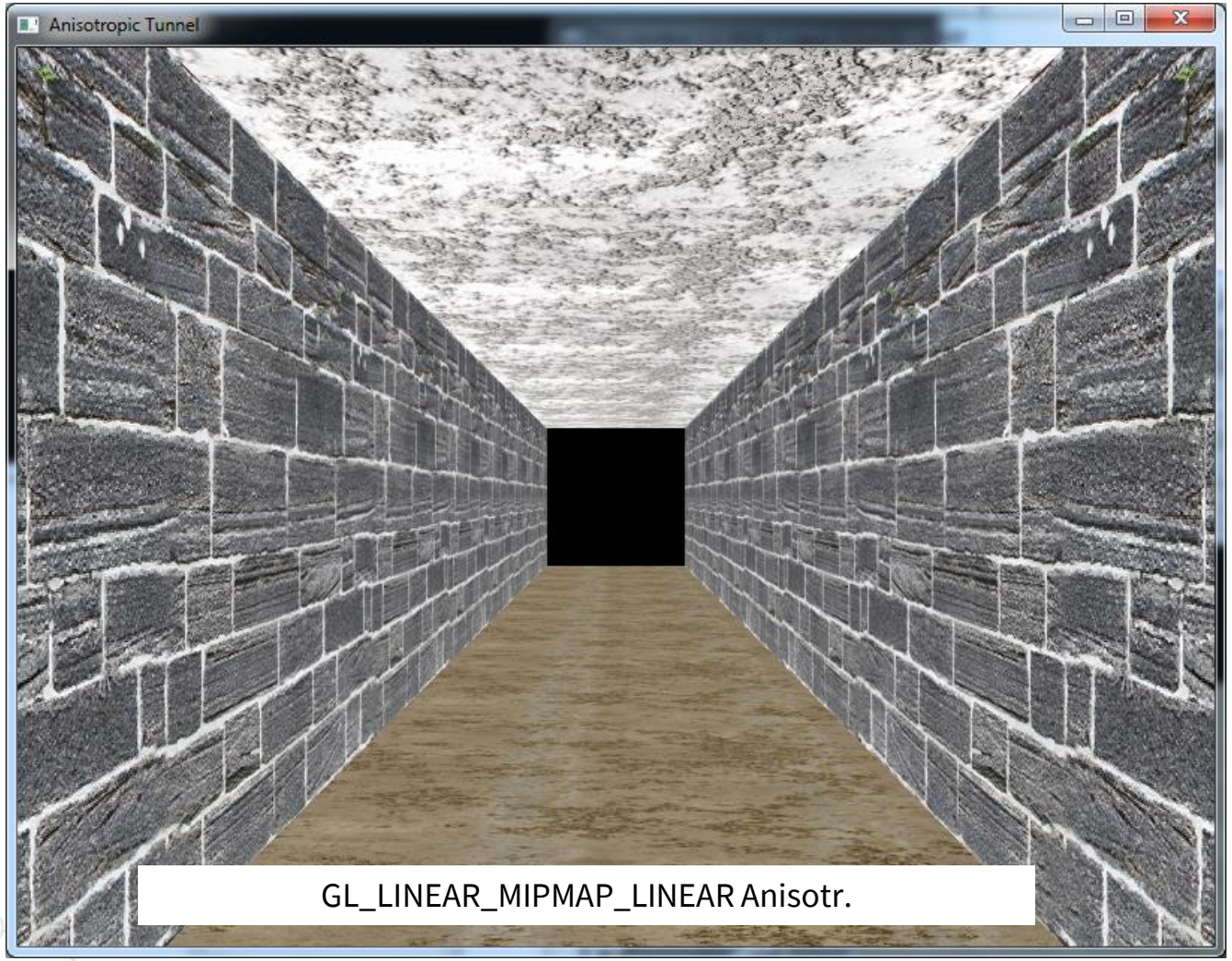
- Tiene en cuenta la orientación del polígono para seleccionar los texels a utilizar
- Se puede usar conjuntamente con cualquier filtro
- Reduce el emborronado de las texturas
- Incrementa el número de texels procesados
- No estaba en el núcleo de OpenGL, pero era una extensión muy popular
- **`if (GLEW_EXT_texture_filter_anisotropic) …`**
  - Comprobar si la extensión está disponible en la configuración actual
- **`glGetFloatv(GL_MAX_TEXTURE_MAX_ANISOTROPY_EXT, &max)`**
  - Devuelve el máximo de anisotropía soportado (1=isotropía)
- **`glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAX_ANISOTROPY_EXT, iso)`**

Novedad: core en GL 4.6

Filtrado anisotrópico

Anisotropic, Superbiblia

Filtrado anisotrópico



GL_LINEAR_MIPMAP_LINEAR Anisotr.

Usando varias texturas simultáneamente

- OpenGL permite usar más de una textura a la vez para renderizar una primitiva
- Cada textura está instalada en una unidad de textura
- **glGetIntegerv**(GL_MAX_COMBINED_TEXTURE_IMAGE_UNITS, &n) devuelve el número máximo de unidades de textura (mínimo 80 o 16 por tipo de shader)

Multitexture,
Cap. 7 Superbiblia 5ª ed

Usando varias texturas simultáneamente

- Cada unidad de textura ofrece toda la funcionalidad estudiada
- La función **glActiveTexture**(texUnit) activa una unidad específica, a la que harán referencia todas las llamadas a funciones de textura que se han a continuación
  - texUnit: GL_TEXTURE*i*, donde 0<=i<n
- Se pueden asignar objetos de textura a la unidad activa con **glBindTexture**

Novedad en GL 4.5:

```
void glBindTextureUnit(GLuint unit, GLuint texture)
```

Usando varias texturas simultáneamente

- Pasos:
  1. Llamar a **glActiveTexture** para cambiar la unidad de textura actual. Establecer la textura para cada unidad, tal y como se ha hecho hasta ahora.
  2. Al especificar los vértices, cargar más de una coordenada de textura (normalmente en distintos índices de atributo).
  3. También hay que indicar al shader en qué unidad de textura está cada textura:

```
GLint unit;
unit = glGetUniformLocation(programId, "texturaBase");
glUniform1i(unit, 0);
```

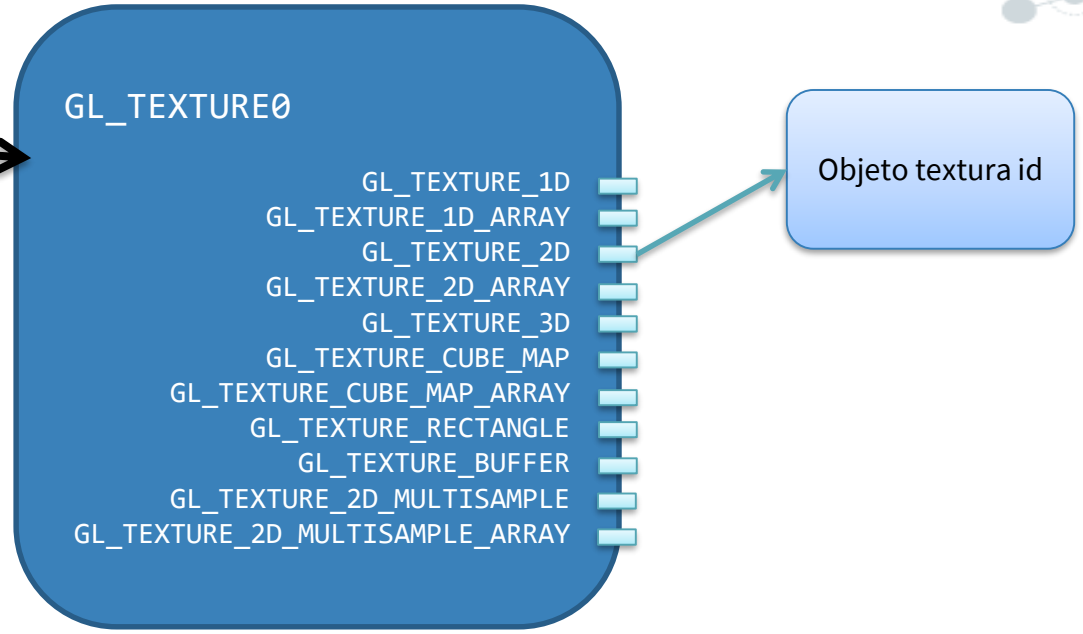
Usando varias texturas simultáneamente

- Ejemplo: (inicialización)

```
/* Construir dos texturas en texNames[]
(glGenTextures, glBindTexture, glTexImage2D, glTexParameteri…)  */
glActiveTexture (GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texNames[0]);
glActiveTexture (GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, texNames[1]);
```

Usando varias texturas simultáneamente

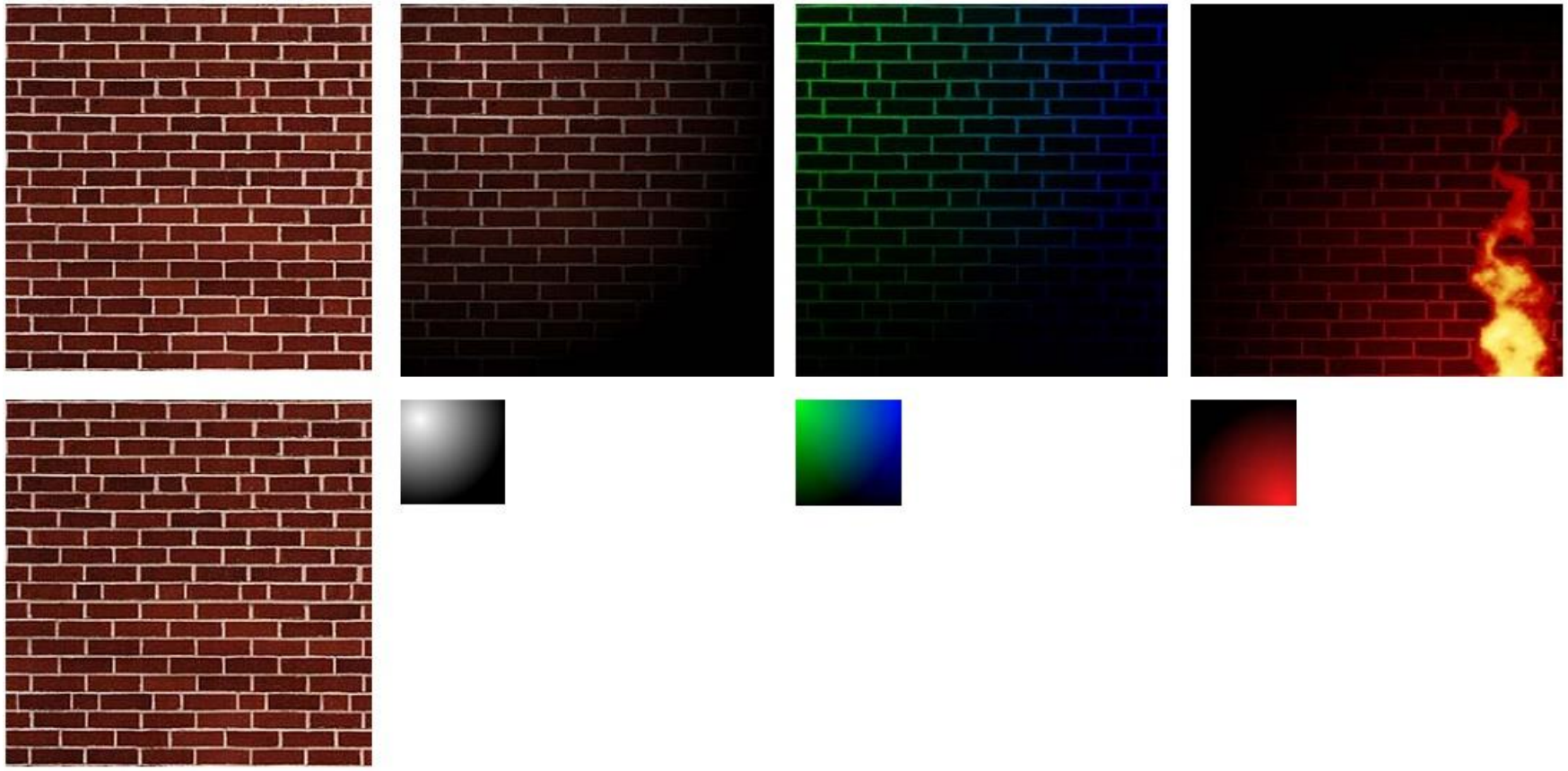
```
Gluint id;
glGenTextures(1, &id);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, id);
```



Unidad de textura

Usando varias texturas simultáneamente

- Ejemplo:



nvidia.com

Usando varias texturas simultáneamente

- Para desconectar la multitextura:

```
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, 0);
glActiveTexture(GL_TEXTURE2);
glBindTexture(GL_TEXTURE_2D, 0);
…
glActiveTexture(GL_TEXTURE0);
```

Otras cosas de texturas

- Otros tipos de textura:
  - 1D: fila de texels
  - 3D: representan volúmenes
- OpenGL puede trabajar con texturas comprimidas, para ahorrar memoria
- Vistas: permiten acceder a una textura como si tuviera un tipo distinto al que realmente tiene
- Texturado de puntos: sistemas de partículas

Array Textures

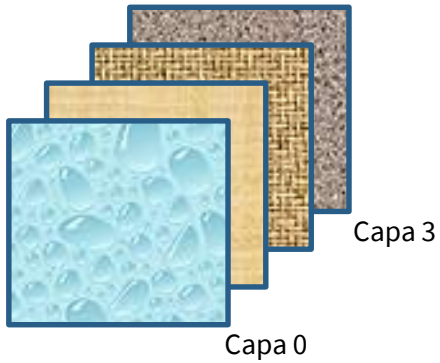
Desde OpenGL 3.0

- Permiten acceder a varias texturas (1D y 2D) a través de una unidad de textura
- Contiene varias “capas” del mismo tamaño y formato (igual que GL_TEXTURE_3D)
- Cada capa se asocia a un índice entero
- Se usa para minimizar el número de vinculaciones de textura en ejecución:
  - Atlas de texturas, animaciones, diferentes componentes de iluminación de un modelo, etc.

Array Textures

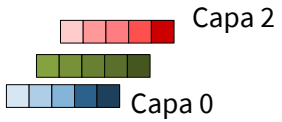
Desde OpenGL 3.0

GL_TEXTURE_2D_ARRAY



- Coordenadas de textura:
  `vec3(s, t, float(capa))`
- Cada capa se carga con:
  `glTexSubImage3D`

GL_TEXTURE_1D_ARRAY



- Coordenadas de textura:
  `vec2(s, float(capa))`
- Cada capa se carga con:
  `glTexSubImage2D`

Buffer textures

Desde OpenGL 3.1

- Texture target: GL_TEXTURE_BUFFER
- Son texturas asociadas a un Buffer Object.
- Permiten acceder a los datos del BO como si fuera un array unidimensional
- Pueden ser muy grandes (tamaño consultable con `GL_MAX_TEXTURE_BUFFER_SIZE`)
- Se accede a sus elementos mediante un entero
- No soportan filtrado, mipmapping, ni repetición de texturas
- Se pueden usar para dar acceso de lectura a los shaders sobre un BO

Texturas en PGUPV

- Clases implicadas:
  - `PGUPV::Texture2D`
  - `PGUPV::Mesh`

Texturas en PGUPV

```
class Texture2D { // Que hereda de Texture2DGeneric, Texture
public:
  Texture2D(GLenum minfilter = GL_LINEAR,
        GLenum magfilter = GL_LINEAR, GLenum wrap_s = GL_REPEAT,  GLenum wrap_t = GL_REPEAT);
  bool loadImage(std::string &filename);
  void bind(GLenum textureUnit = GL_TEXTURE0);
  void unbind();
  void generateMipmap();
  void setMinFilter(GLenum filter);
  void setMagFilter(GLenum filter);
  void setWrapS(GLenum wrap);
  void setWrapT(GLenum wrap);
  GLenum getMinFilter;
  GLenum getMagFilter();
  GLenum getWrapS();
  GLenum getWrapT();
  uint getWidth();
  uint getHeight();
…
```

Texturas en PGUPV

```
class Mesh {
public:
  Mesh();
  void addTexCoord(uint tex_unit, const std::vector<T> &t, GLenum usage)
  void addTexCoord(uint tex_unit, glm::vec2 *t, size_t n, GLenum usage);
  void addTexCoord(uint tex_unit, GLfloat *t, size_t n, GLenum usage);
  …
```

Proyecto ej3-1