

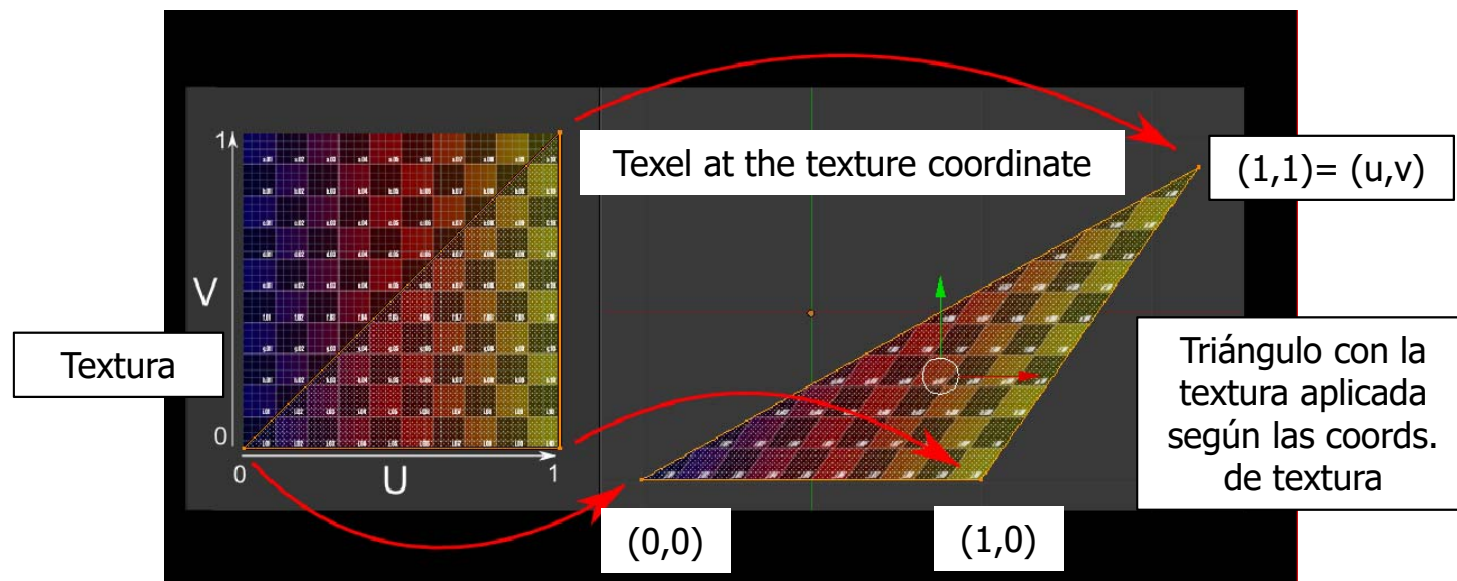
Introducción: Texturas

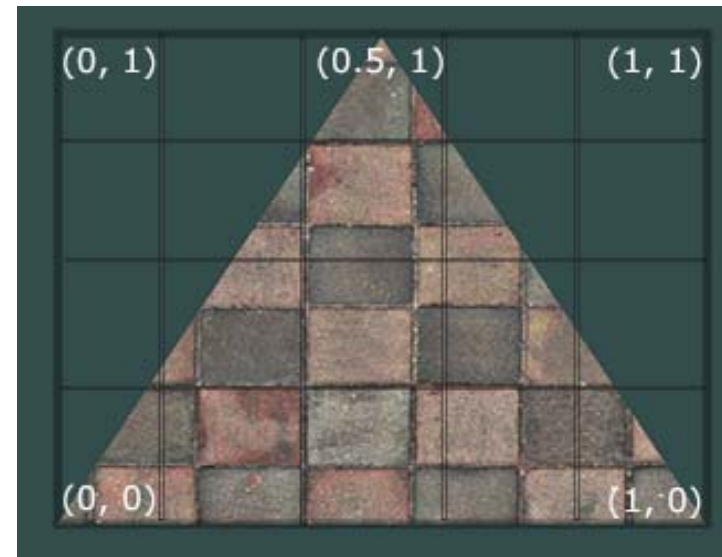
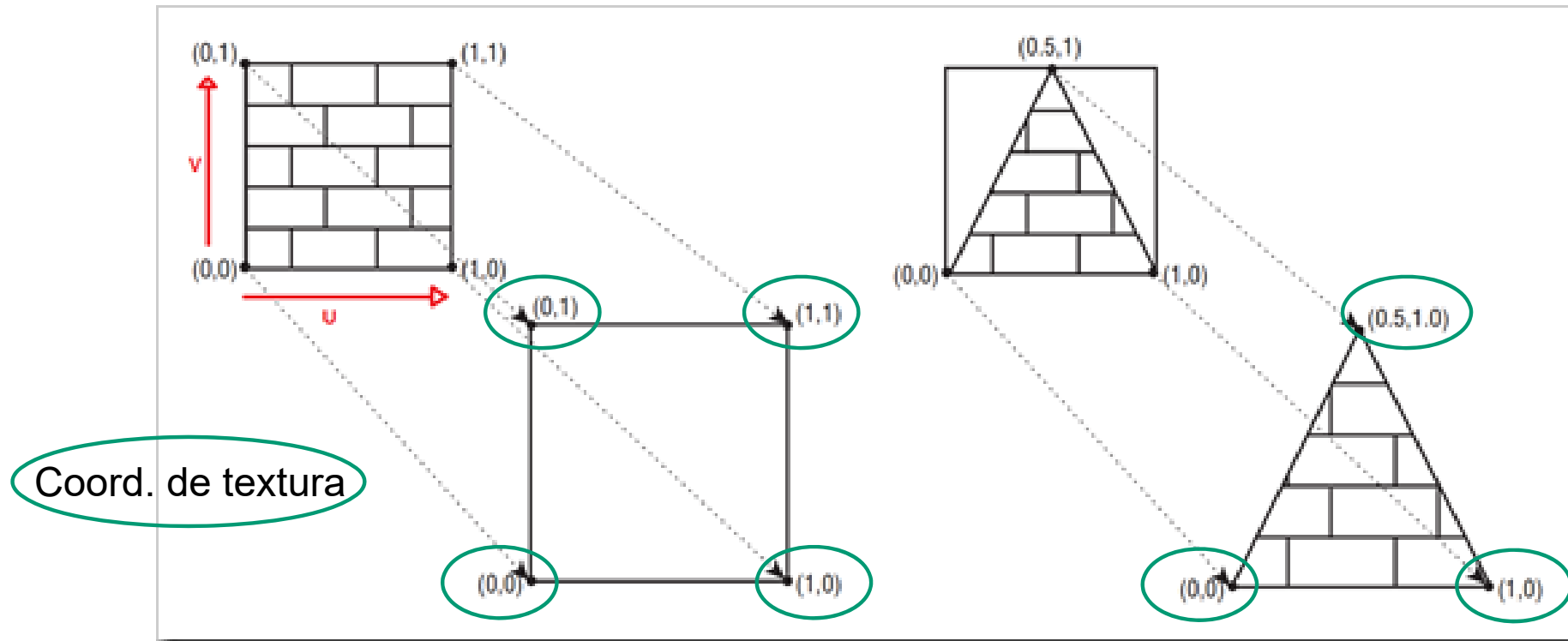
- ☐ Definición
- ☐ Aplicación a una malla
- ☐ Combinación de la textura con el color del fragmento
- ☐ Objetos de textura en OpenGL: la clase Texture

Ana Gil Luezas
Departamento de Sistemas Informáticos y Computación
Facultad de Informática
Universidad Complutense de Madrid

- ❑ Una textura es una imagen que se aplica a una malla para darle mayor realismo al objeto. Las texturas se pueden generar mediante algoritmos, o se pueden utilizar imágenes previamente rasterizadas.
- ❑ Cuando se le añade una textura a una malla, hay que indicar qué parte de la imagen tiene que ser usada para cada triángulo, y para esto se emplean las coordenadas de textura (S,T) o (U,V).

Cada vértice de la malla tiene que tener, además de su posición, sus coordenadas de textura: un dvec2. Estas coordenadas se utilizan para obtener el téxel de la imagen:





Definición de texturas

- Una textura 2D es una función de dos parámetros

$\text{Tex}(s,t): \mathbb{R} \times \mathbb{R} \rightarrow \text{Colores}$

Pero se utiliza la forma normalizada en los intervalos $[0,1]$

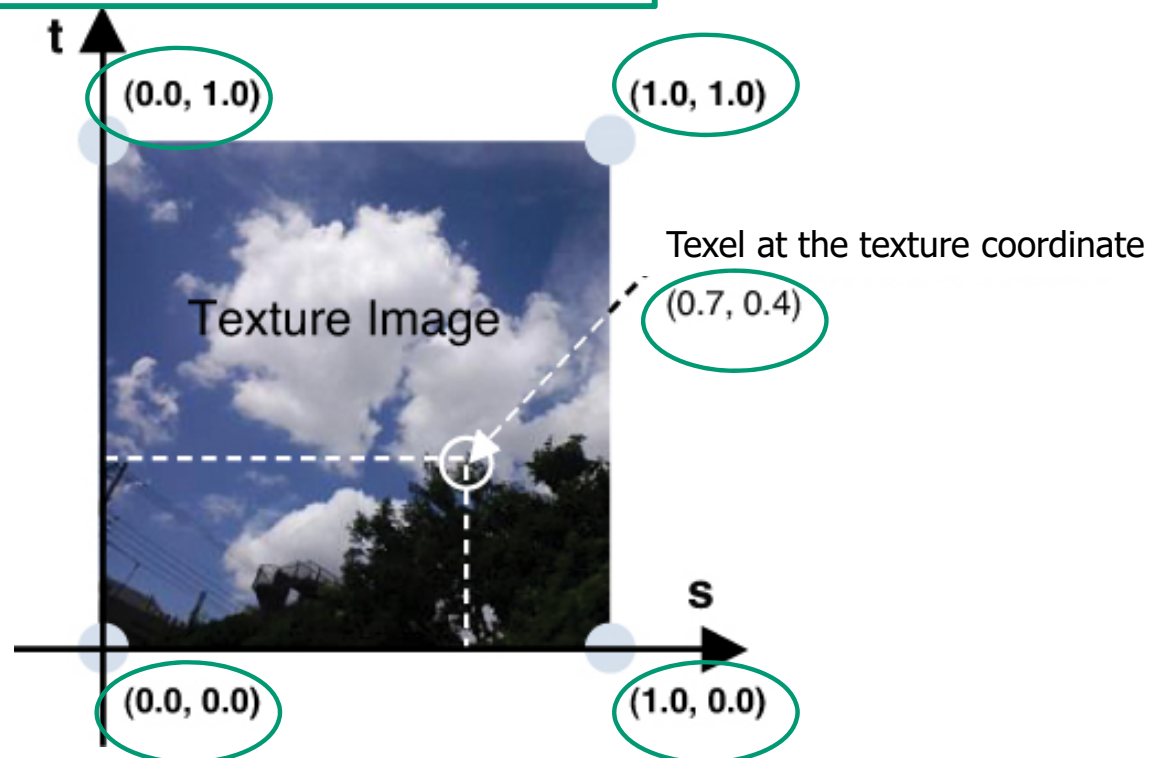
$T(s,t): [0,1] \times [0,1] \rightarrow \text{Colores}$

```
RGBA Texture (double s, double t) { // en [0,1]
    return img[trunc(s*WC)] [trunc(t*HF)];
} // suponiendo RGBA img[WC][HF]
```

Función de textura
 $T(s,t)$ = color para las coordenadas (s,t)

Coord. de textura

Las coordenadas de textura de los píxeles del interior se obtienen por interpolación

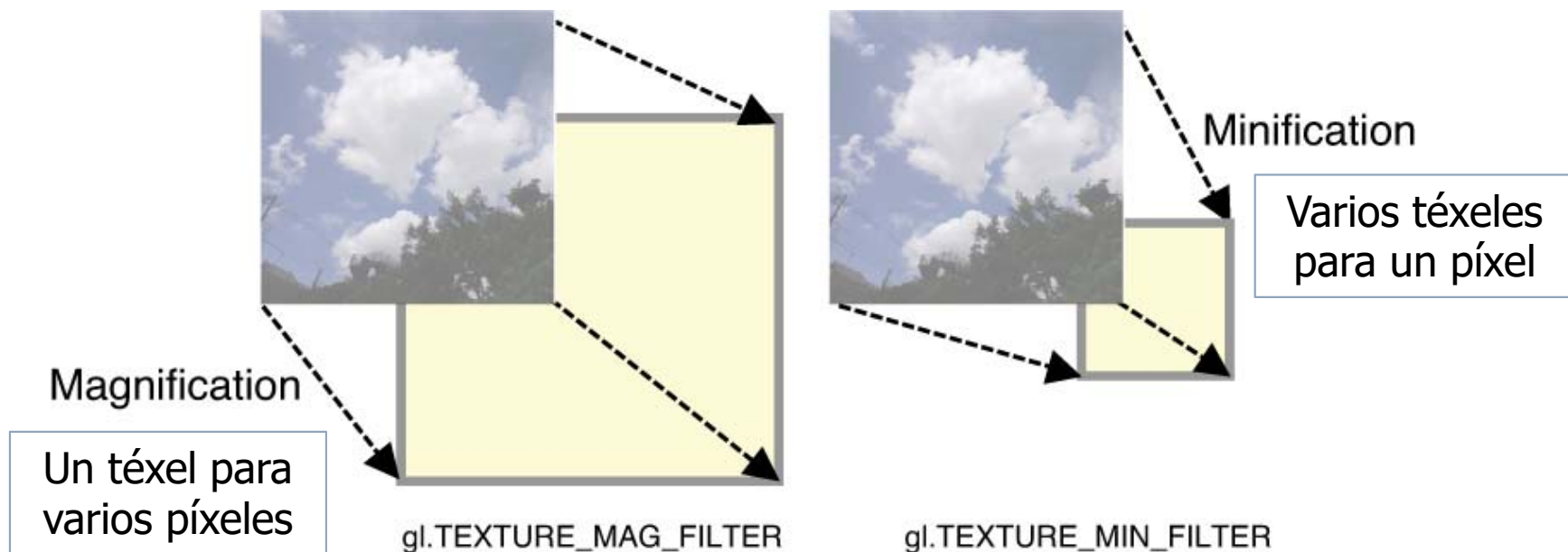


- Definición con imágenes rasterizadas.

```
RGBA Texture (double s, double t) { // en [0,1]
    return img[trunc(s*WC)] [trunc(t*HF)];
} // suponiendo RGBA img[WC][HF]
```

Función de textura
 $T(s,t)$ = color para las
coordenadas (s,t)

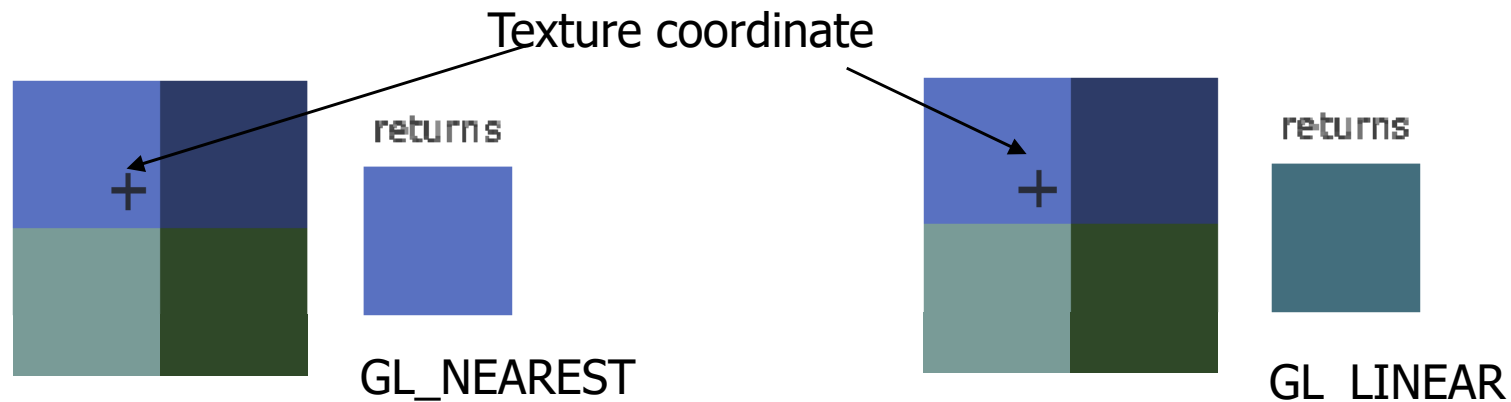
- En caso de tener que aumentar o reducir la imagen durante el renderizado, se pueden aplicar filtros .
También se utilizan imágenes de distintas resoluciones (mipmaps).



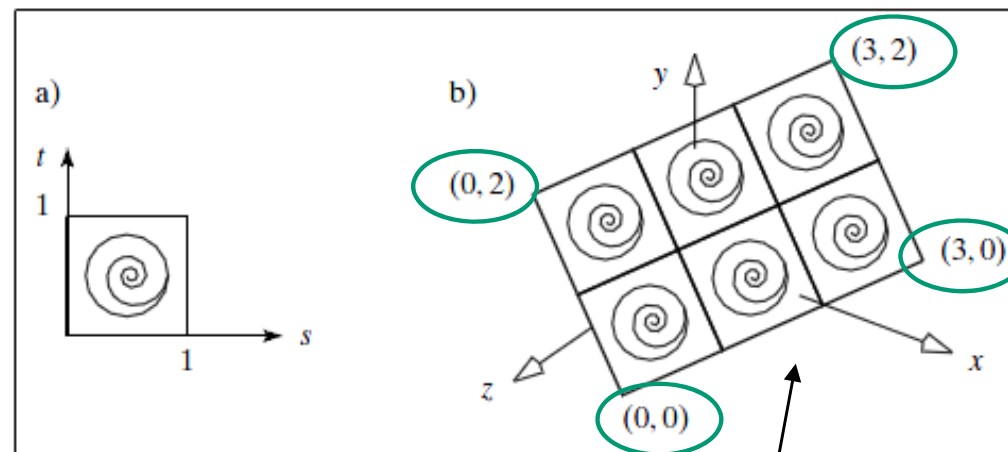
- ❑ **Filtros:** Función para determinar el color correspondiente a las coordenadas de textura de cada píxel

GL_NEAREST: El color del téxel más cercano a las coordenadas de textura

GL_LINEAR: La media de los colores de los cuatro téxeles más cercanos a las coordenadas de textura



- ❑ OpenGL permite asignar coordenadas fuera del intervalo $[0,1]$. ¿Qué ocurre cuando las coordenadas de textura caen fuera del rango $[0, 1]$? Por defecto **GL_REPEAT**: la textura se repite (tiling). Se ignora la parte entera de las coordenadas de textura.



1 Rectángulo

Aplicación de una textura a una malla

La textura puede aplicarse de varias formas:

- ☐ Recubriendo toda la superficie del objeto con la textura.
- ☐ Recortando parte de la textura.
- ☐ Pegando la textura en una zona concreta de la superficie.

Deben de preservarse las proporciones de la textura para evitar distorsiones de la imagen de la textura.

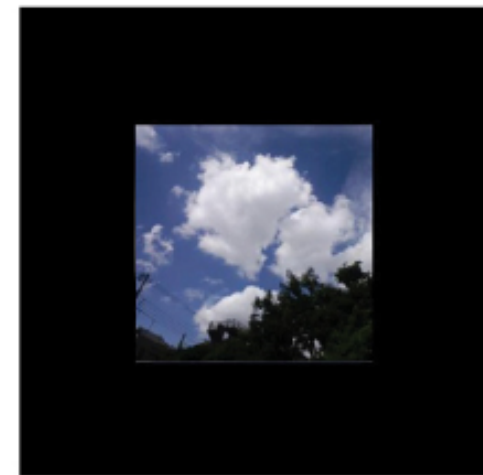
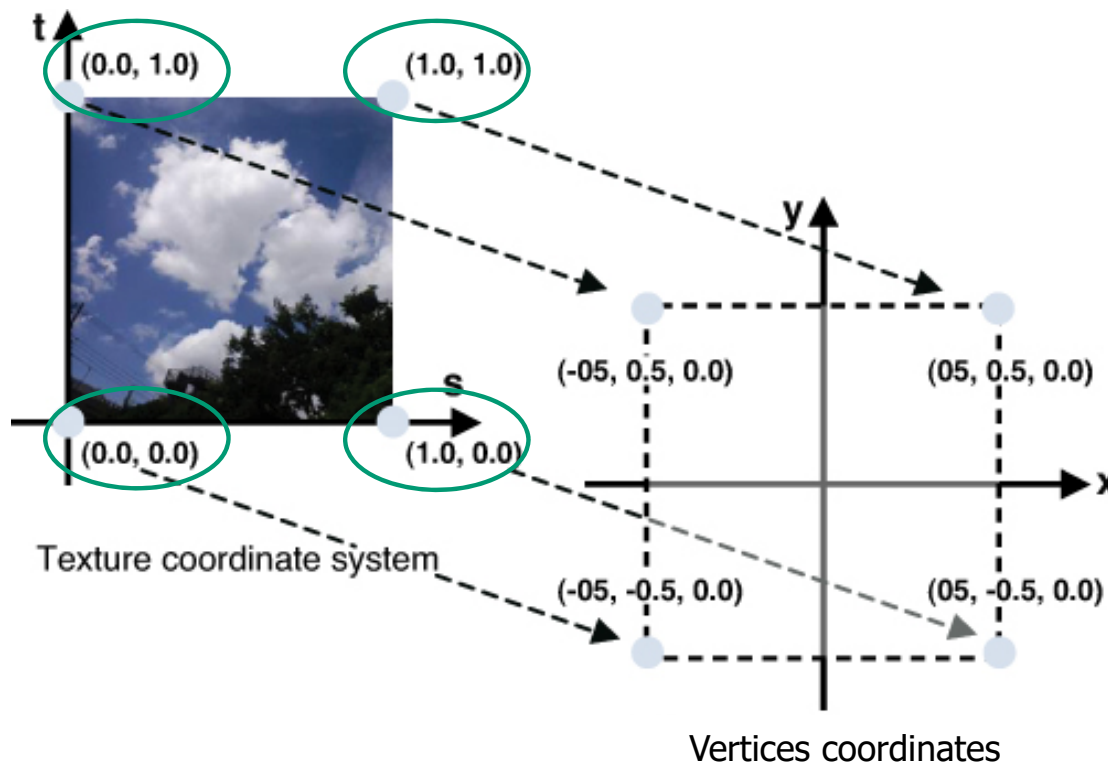
Aplicación de una textura a una malla

- ❑ **Texture mapping:** establecer las coordenadas de textura (s, t) de cada vértice

Map: Vértices de la malla $(\mathbb{R} \times \mathbb{R} \times \mathbb{R}) \rightarrow [0,1] \times [0,1]$

OpenGL permite asignar coordenadas fuera del intervalo $[0,1]$

(\rightarrow configurar **wrapping**)



The resulting image produced

Aplicación de una textura a una malla

- ❑ A cada vértice hay que asignarle sus coordenadas de textura (s, t) añadiendo a la **clase Mesh** un vector **de coordenadas de textura** (análogo al vector de colores pero de 2 coordenadas):

```
std::vector<glm::dvec2> vTexCoords; // vector de coordenadas de textura
```

El método **Mesh::render()** tiene que activar (y desactivar) el array de coordenadas de textura:

```
glEnableClientState(GL_TEXTURE_COORD_ARRAY);  
glTexCoordPointer(2, GL_DOUBLE, 0, vTexCoords.data());
```

Aplicación de una textura a una malla

Ejemplo: toda la textura en un rectángulo

```
Mesh* Mesh::generaRectanguloTexCor(GLdouble w, GLdouble h) {
```

```
    Mesh *m = generaRectangulo(w, h);
```

```
    m->vTexCoords.reserve(m->mNumVertices);
```

```
    m->vTexCoords.emplace_back(0, 1);
```

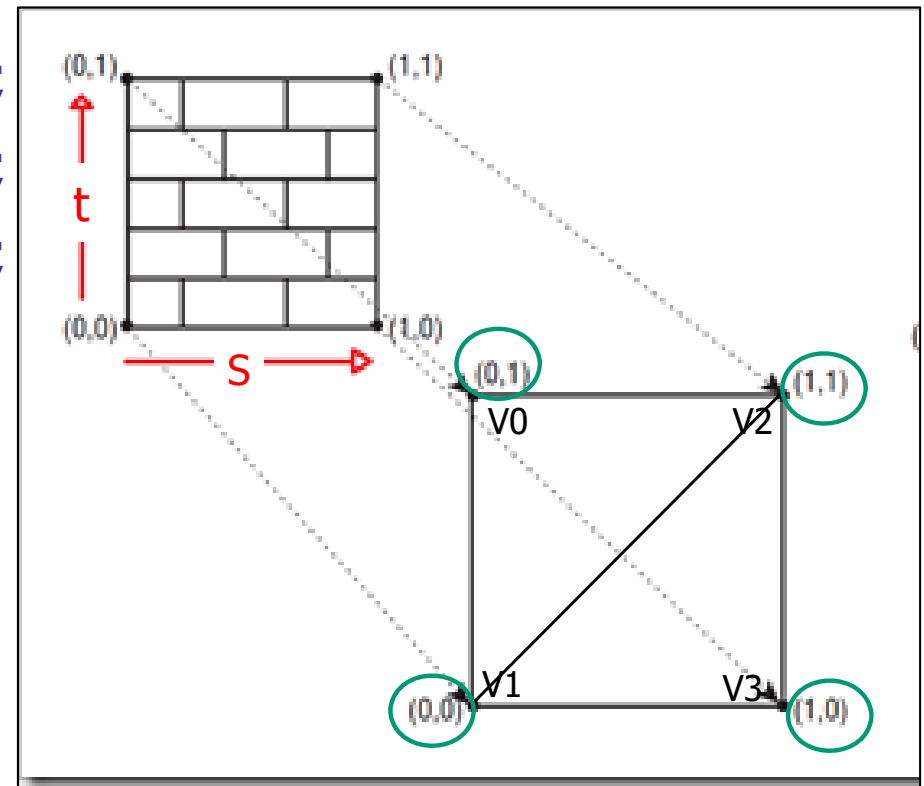
```
    m->vTexCoords.emplace_back(0, 0);
```

```
    m->vTexCoords.emplace_back(1, 1);
```

```
    m->vTexCoords.emplace_back(1, 0);
```

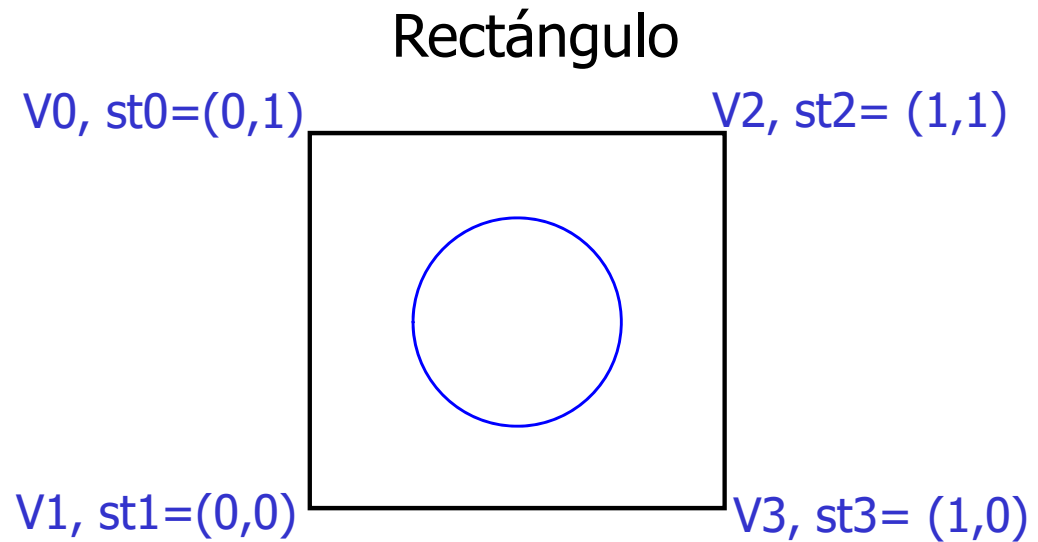
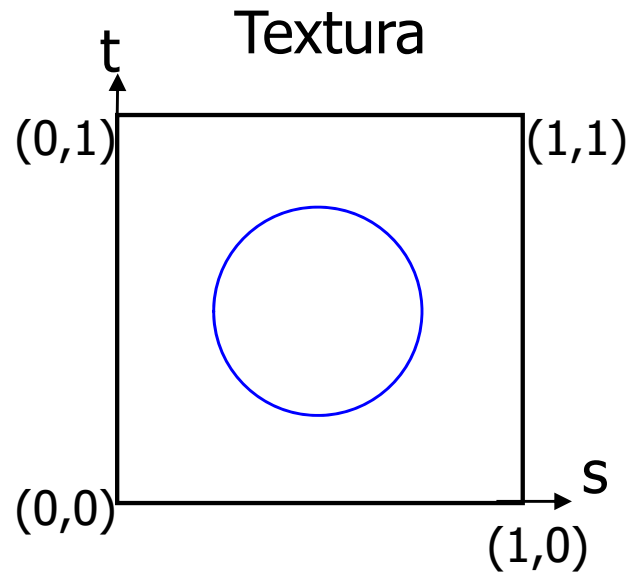
```
    return m;
```

```
}
```

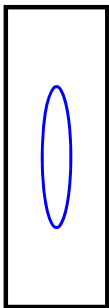


Aplicación de una textura a una malla

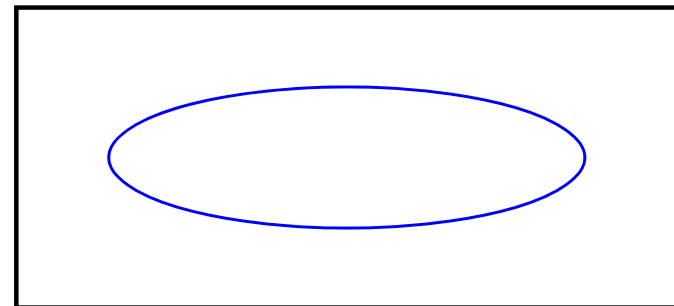
Ejemplo: dependiendo de las dimensiones del rectángulo (v_0, v_1, v_2, v_3)



Rectángulo



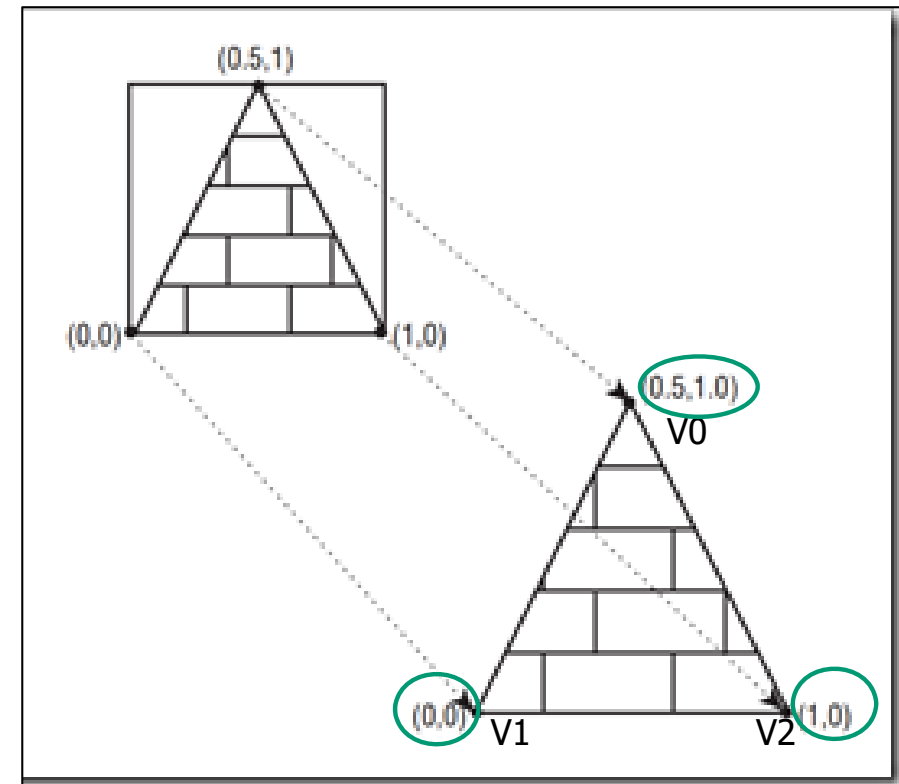
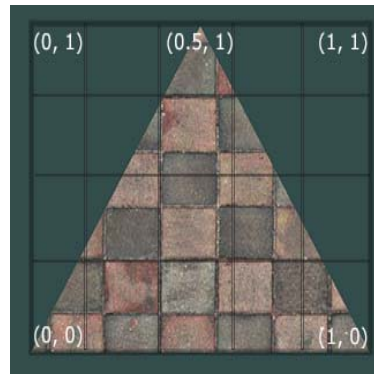
Rectángulo



Aplicación de una textura a una malla

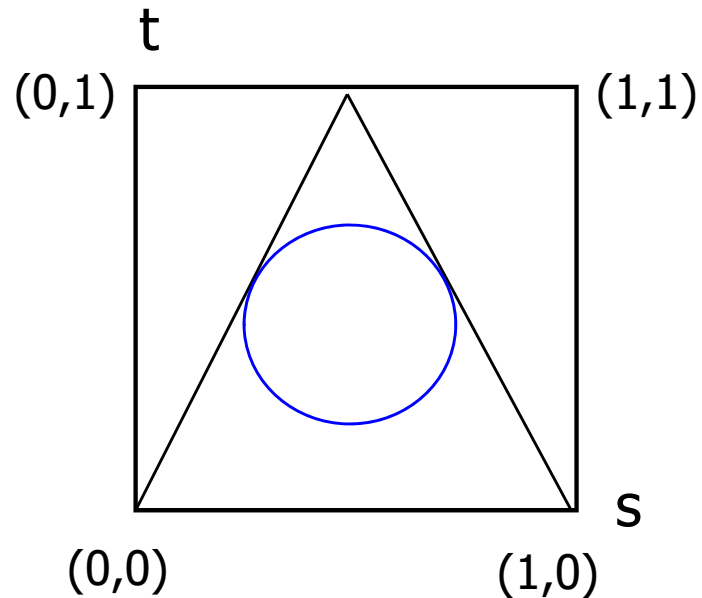
Ejemplo: Parte de una textura en un triángulo

```
Mesh* Mesh::generaTrianguloTexCor(GLdouble rd) {  
    Mesh *m = generaPoligono(3, rd);  
    ...  
    m->vTexCoords.reserve(3);  
    m->vTexCoords.emplace_back(0.5, 1);  
    m->vTexCoords.emplace_back(0, 0);  
    m->vTexCoords.emplace_back(1, 0);  
    return m;  
}
```

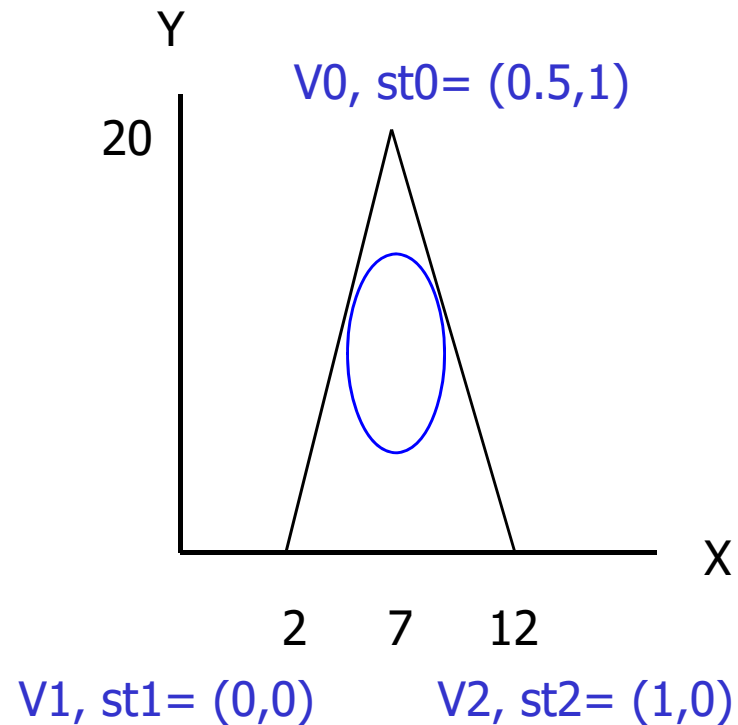


Aplicación de una textura a una malla

Ejemplo: dependiendo de las dimensiones del triángulo



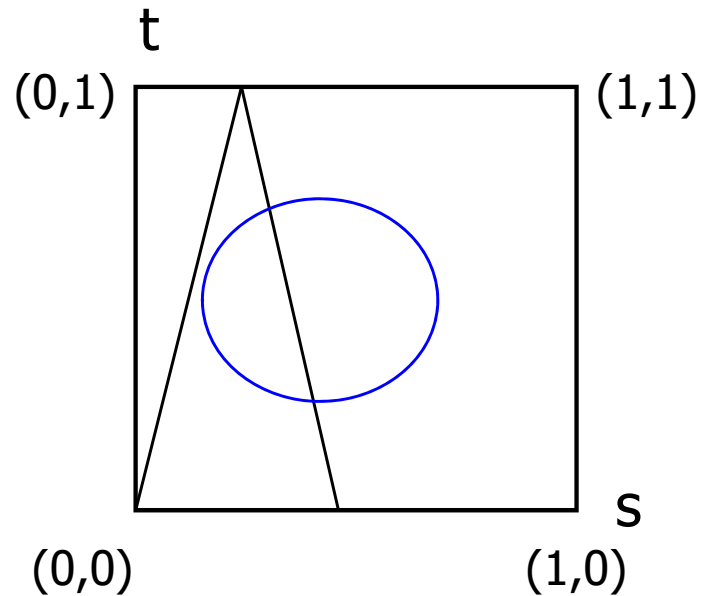
Textura



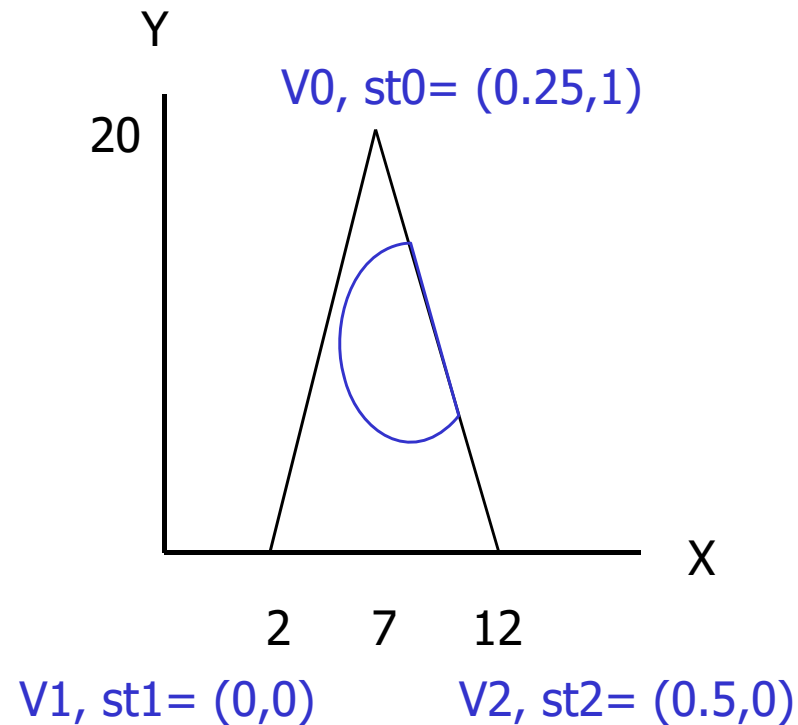
Triángulo

Aplicación de una textura a una malla

Ejemplo: Parte de una textura en un triángulo



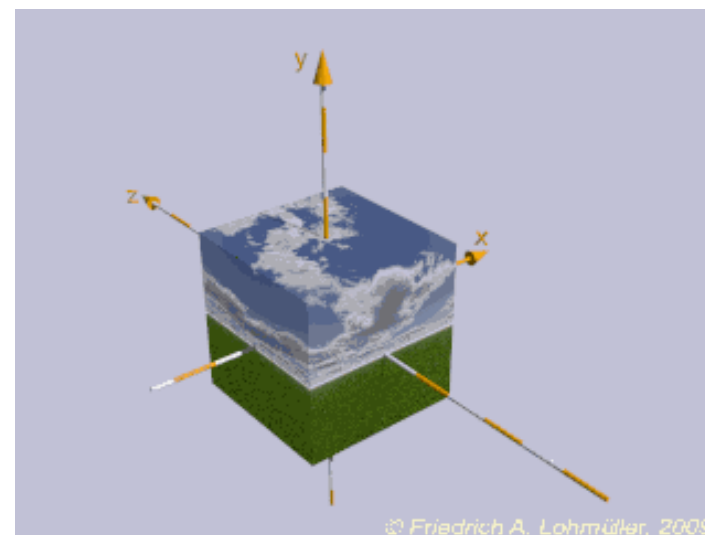
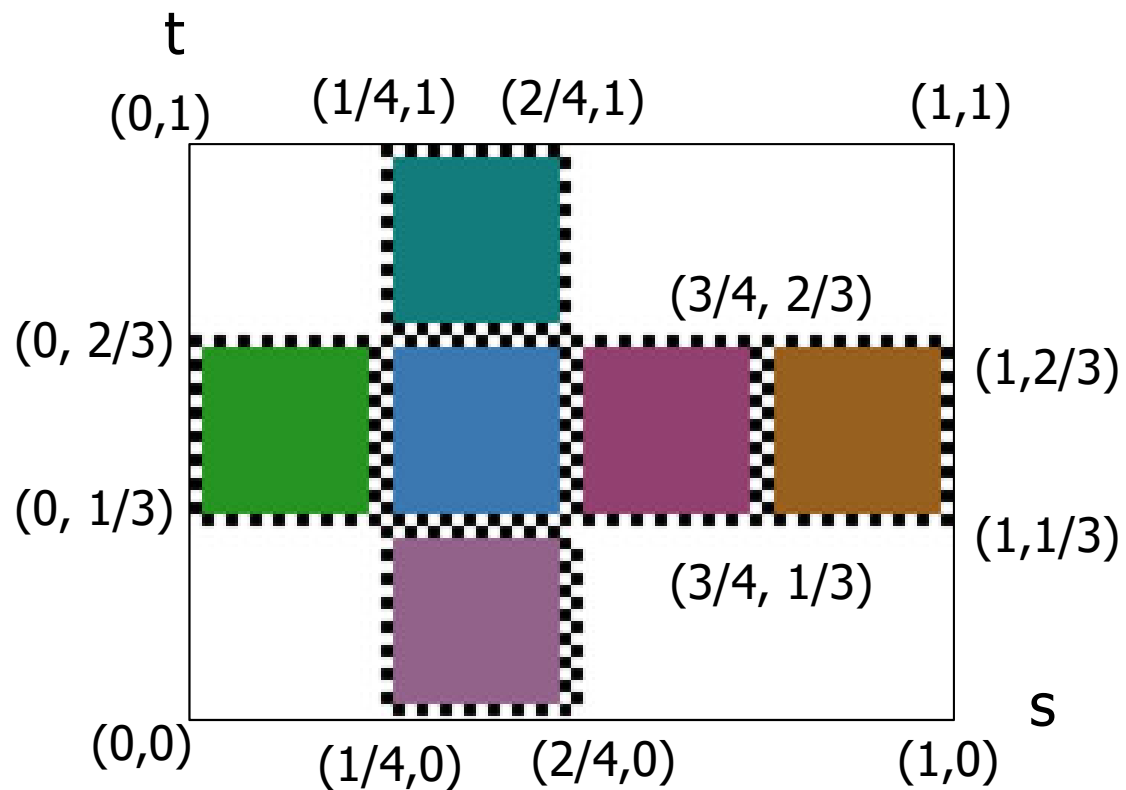
Textura



Triángulo

Aplicación de la textura a una malla

Ejemplo: cubo

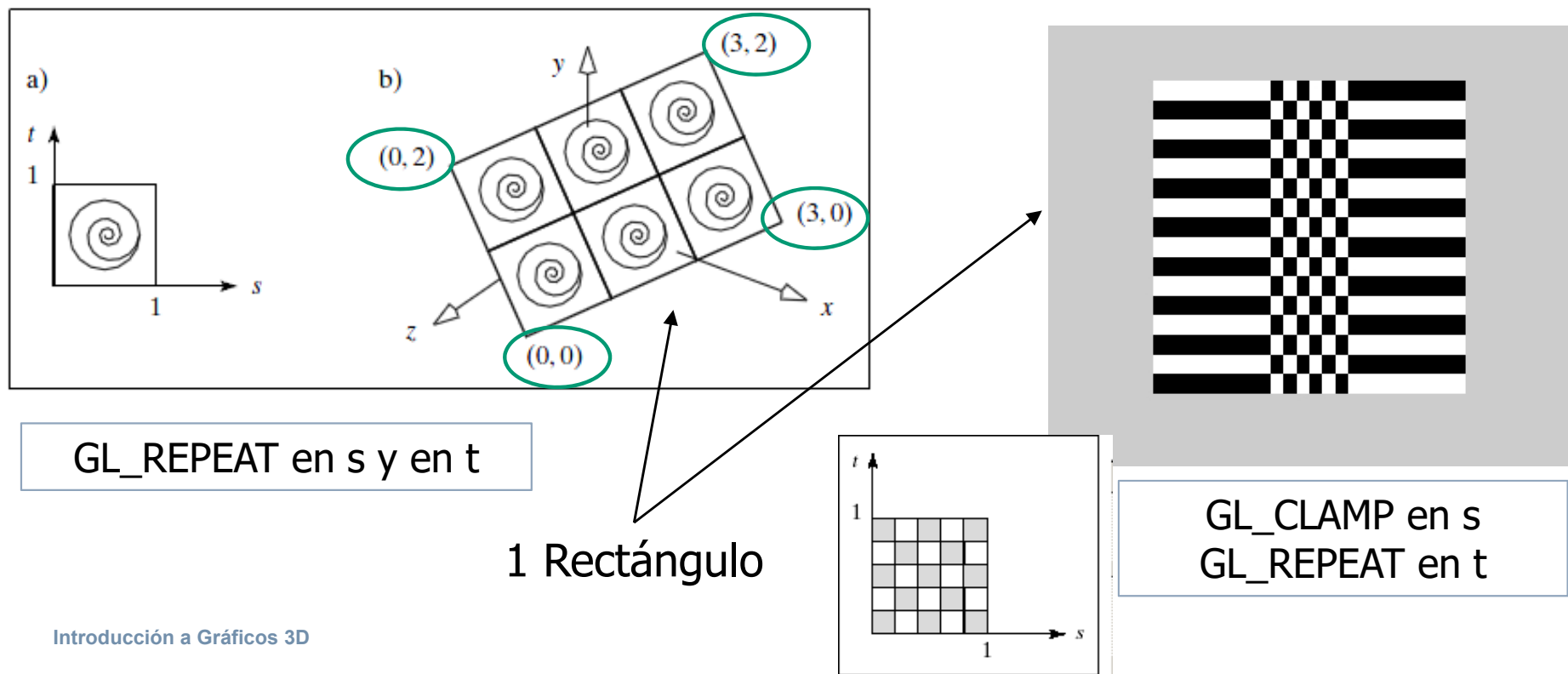


Aplicación de una textura a una malla

- ❑ OpenGL permite asignar coordenadas fuera del intervalo $[0,1]$:
 - > configurar texture wrap parameter

GL_REPEAT: la textura se repite (tiling). Se ignora la parte entera de las coordenadas de textura.

GL_CLAMP: coordenadas de textura superiores a 1 se ajustan a 1, y las coordenadas inferiores a 0 se ajustan a 0.



Mezcla de la textura con el color

Cada fragmento, del interior de un triángulo, consta de las coordenadas (x, y, z) del píxel, un color C y las coordenadas de textura (s, t) .

El color C se mezcla con el color de la textura $T(s, t)$: $C_r = \text{mix}(C, T(s, t))$

Las formas más habituales de combinar estos colores son:

- ❑ **GL_REPLACE**. Utilizar exclusivamente la textura: $C_r = T(s, t)$
- ❑ **GL_MODULATE**. Modular ambos colores: $C_r = C * T(s, t)$
- ❑ **GL_ADD**. Sumar ambos colores: $C_r = C + T(s, t)$

El color resultante se escribirá en el Color Buffer

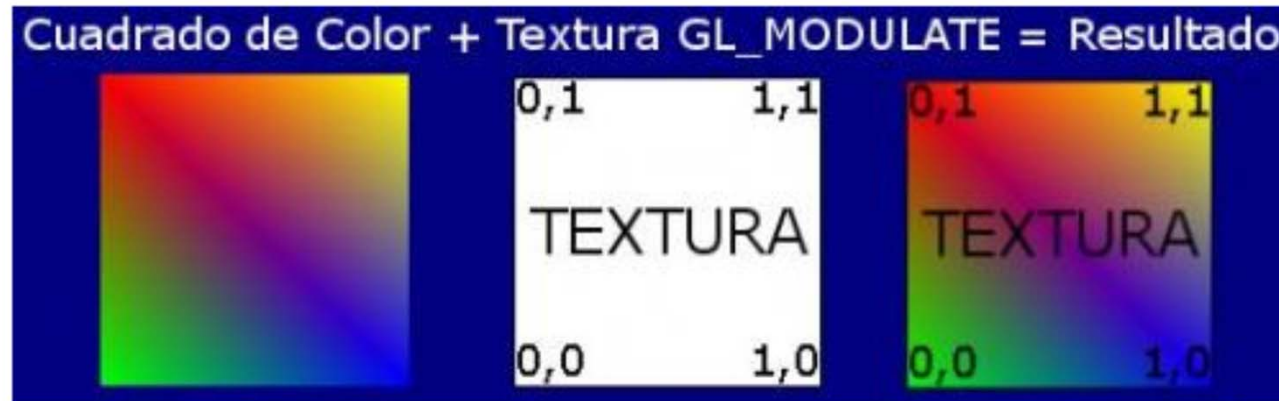
Función de textura
 $T(s, t)$ = color para las
coordenadas (s, t)

Mezcla de la textura con el color

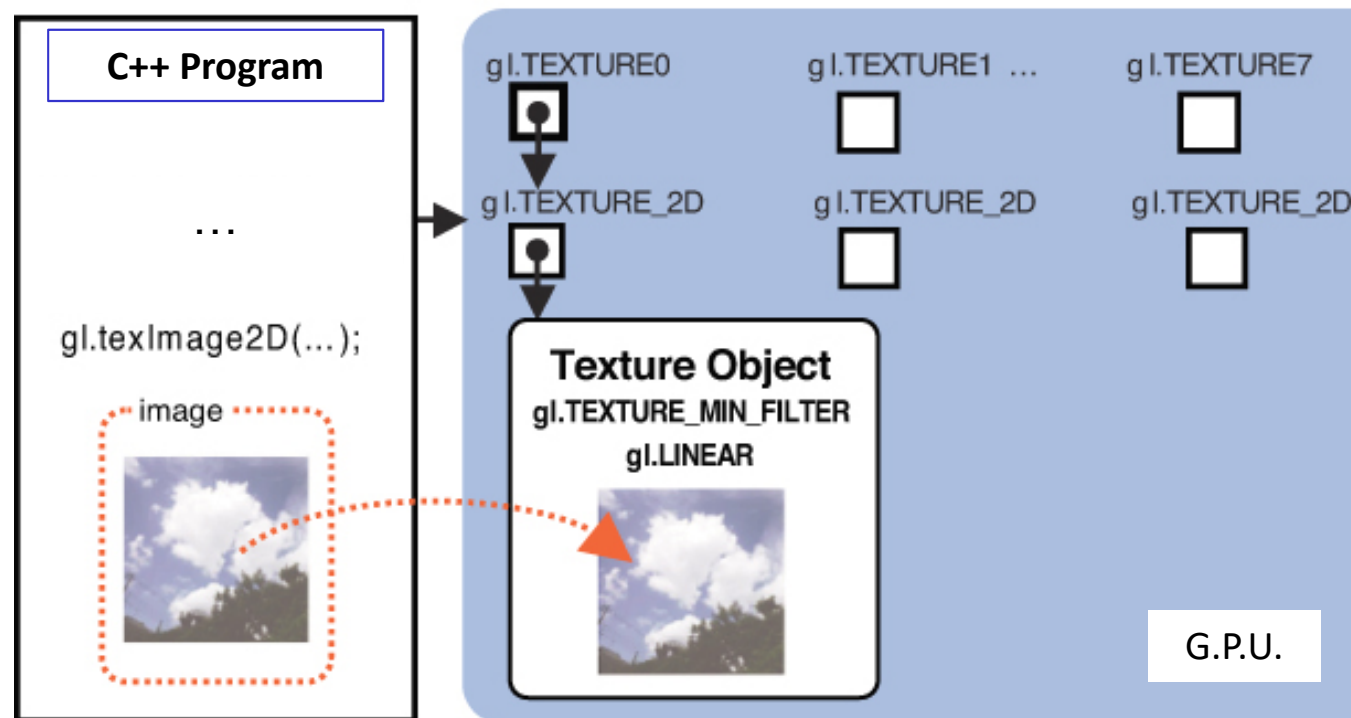
GL_REPLACE. Utiliza exclusivamente la textura: $C_r = T(s,t)$



GL_MODULATE. Modula ambos colores: $C_r = C * T(s,t)$



- ❑ En OpenGL las texturas se gestionan mediante **objetos de textura**: estructuras GPU que contienen la imagen y la configuración de la función de textura (filtros y wrapping, pero no el modo de mezcla)



- ❑ Hay que activar, en `scene::setGL`, el uso de texturas con `glEnable(GL_TEXTURE_2D);`
Y desactivarlo con `glDisable(GL_TEXTURE_2D);` (en `resetGL`)

❑ Gestión de objetos de texturas

- Crearlos y destruirlos: `glGenTextures(...)`, `glDeleteTextures(...)`
- Configurarlos (filtros y wrapping): `glTexParameter*(...)`
- Activarlos para que tengan efecto: `glBindTexture(...)`, `glTexEnvf(...)`
- Transferir la imagen (de CPU a GPU):

```
glTexImage2D (  
    GL_TEXTURE_2D, // 1D / 3D  
    0, // mipmap level  
    GL_RGBA, // Formato interno (GPU) de los datos de la textura  
    width, height, // Potencias de 2?  
    0, // -> border  
    GL_RGBA, // Formato de los datos de la imagen (data)  
    GL_UNSIGNED_BYTE, // Tipo de datos de los datos de data  
    data // puntero a la variable CPU con la imagen  
)
```

- ❑ Añadimos una nueva clase, `Texture`, con métodos para cargar de archivo una imagen y transferirla a la GPU (`load`), y para activar (`bind`) y desactivar (`unbind`) la textura en la GPU cuando la queramos usar.

- ❑ Añadimos a la clase `Scene` un atributo para las texturas:

```
vector<Texture*> gTextures;
```

- ❑ La entidad necesita una malla con coordenadas de textura y la textura que queremos usar.

Añadimos a la clase `Entity` un atributo para la textura (`Texture* mTexture`), que habrá que establecer al crear la entidad (método `setTexture`), y activar/desactivar al renderizarla.

- ❑ Generamos mallas con coordenadas de textura.

```
class Texture    // utiliza la clase PixMap32RGBA para el método load
{
public:
    Texture(){};
    ~Texture() {if (mId !=0 ) glDeleteTextures(1, &mId); };
    void load(const std::string & BMP_Name, GLubyte alpha = 255);
                                   // cargar y transferir a GPU
    void bind(GLuint mixMode); // mixMode: GL_REPLACE / MODULATE / ADD
    void unbind() { glBindTexture(GL_TEXTURE_2D, 0); };
protected:
    void init();
    GLuint mWidth =0, mHeight =0; // dimensiones de la imagen
    GLuint mId=0;    // identificador interno (GPU) de la textura
                    // 0 significa NULL, no es un identificador válido
};
```

```
void Texture:: init() {  
    glGenTextures(1, &mId); // genera un identificador para una nueva textura  
    glBindTexture(GL_TEXTURE_2D, mId); // filters and wrapping  
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);  
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);  
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);  
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);  
}  
  
void Texture:: bind(GLuint mixMode) { // modo para la mezcla los colores  
    glBindTexture(GL_TEXTURE_2D, mId); // activa la textura  
    // el modo de mezcla de colores no queda guardado en el objeto de textura  
    glTexEnv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, mixMode);  
    // modos: GL_REPLACE, GL_MODULATE, GL_ADD ...  
}
```



```
void Texture:: load(const std::string & BMP_Name, GLubyte alpha) {  
    if (mId == 0) init();  
    PixMap32RGBA pixMap;    // var. local para cargar la imagen del archivo  
    pixMap.load_bmp24BGR(BMP_Name);    // carga y añade alpha=255  
    // carga correcta ? -> exception  
    if (alpha != 255) pixMap.set_alpha(alpha);  
    mWidth = pixMap.width();  
    mHeight = pixMap.height();  
    glBindTexture(GL_TEXTURE_2D, mId);  
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, mWidth, mHeight, 0,  
                GL_RGBA, GL_UNSIGNED_BYTE, pixMap.data()); // transferir a GPU  
    glBindTexture(GL_TEXTURE_2D, 0); // la textura queda desactivada ?  
}
```

- ❑ La entidad necesita una malla con coordenadas de textura y la textura que queremos usar.

```
void Suelo::Suelo(...) {  
    mesh = Mesh::generateRectanguloTexCor(...);  
    // rectángulo con coordenadas de textura  
    ...  
}
```

- ❑ Añadimos a la clase `Abs_Entity` un atributo para la textura:

```
Texture * mTexture = nullptr;
```

Y un método para establecerla

```
void setTexture(Texture* tex) { mTexture = tex; };
```

- ❑ En el método `render`, las entidades que usen texturas, hay que activar la textura antes de renderizar la malla, y desactivarla después.

- ❑ Añadimos a la clase `Scene` un atributo para las texturas:

```
vector<Texture*> gTextures;
```

- ❑ En `init` creamos y cargamos (con el método `load()`) las texturas de los objetos de la escena. Y en `free` las liberamos.
- ❑ Al crear los objetos establecemos sus texturas.
- ❑ Adaptamos los métodos `setGL` y `resetGL` para activar/desactivar las texturas en OpenGL.

Para **activar las texturas** (en `setGL`): `glEnable(GL_TEXTURE_2D);`

Para desactivarlas (en `resetGL`): `glDisable(GL_TEXTURE_2D);`

- ❑ Copiar en la textura activa parte de la imagen del Color Buffer

```
glCopyTexImage2D(GL_TEXTURE_2D, level(0), internalFormat,  
                 xLeft, yBottom, width, height, border(0));
```

// en coordenadas de pantalla (como el puerto de vista)

Los datos se copian del buffer de lectura activo: GL_FRONT o GL_BACK

Para modificar el buffer de lectura activo:

```
glReadBuffer(GL_FRONT / GL_BACK); // por defecto GL_BACK
```

- ❑ Obtener (de GPU a CPU) la imagen de la textura activa

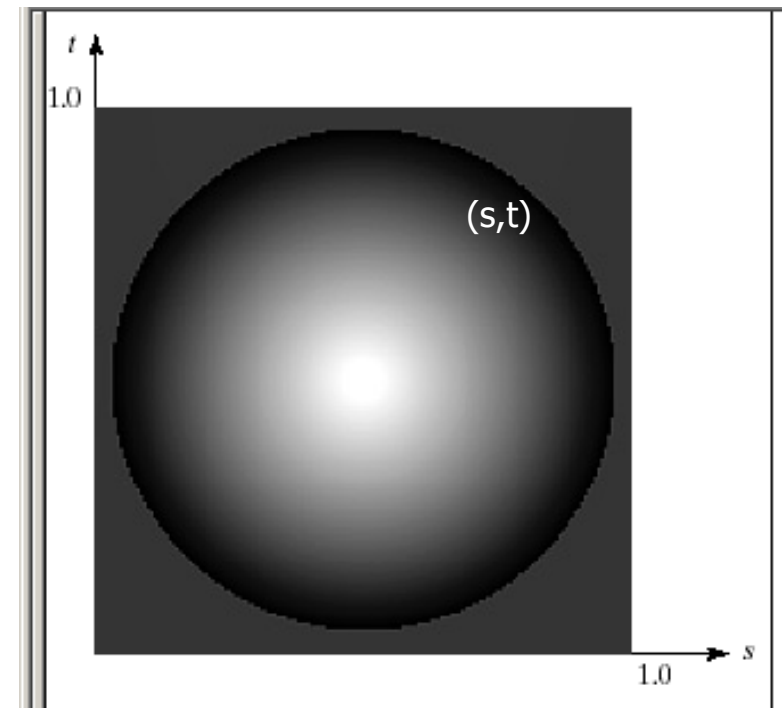
```
glGetTexImage(GL_TEXTURE_2D, level(0), imgFormat, imgType, pixels);  
// pixels-> array donde guardar los datos (de tipo y tamaño adecuado)
```

- Definición **procedimental**. Ejemplo para colores de una sola componente de tipo double (escala de grises):

```
double TexturaProc (double s, double t) { // en [0,1]

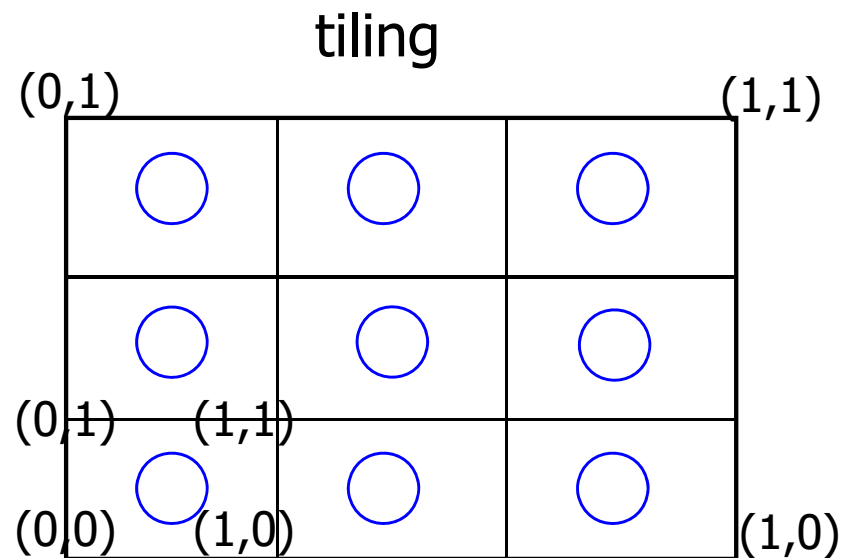
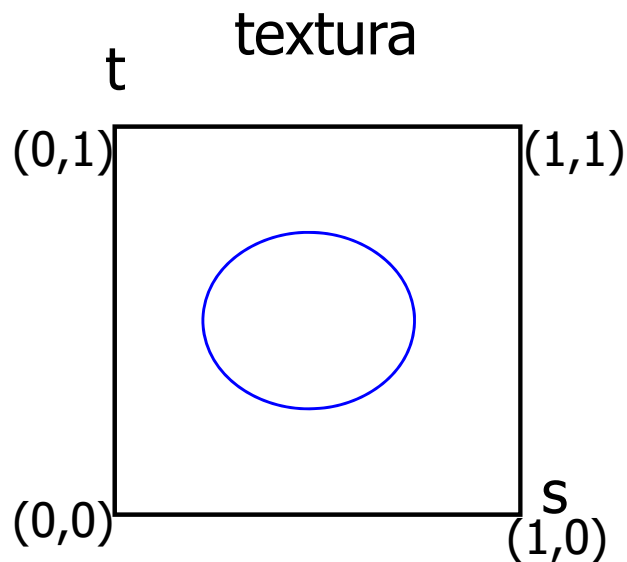
    double r = 0.4; // radio
    double d = sqrt((s-0.5)*(s-0.5) + (t-0.5)*(t-0.5)); // distancia al centro
    if (d < r) return 1 - (d / r); // intensidad del círculo
    else return 0.2; // background
}
```

Los puntos dentro del radio del círculo (r) son más oscuros en el borde ($d \approx r$) y más claros en el centro ($d \approx 0.0$)



Aplicación de la textura a una malla

Ejemplo: tablero formado por NDC x NDF rectángulos



$$\text{coordText}(i,j)=(i/\text{NDC},j/\text{NDF})$$

