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

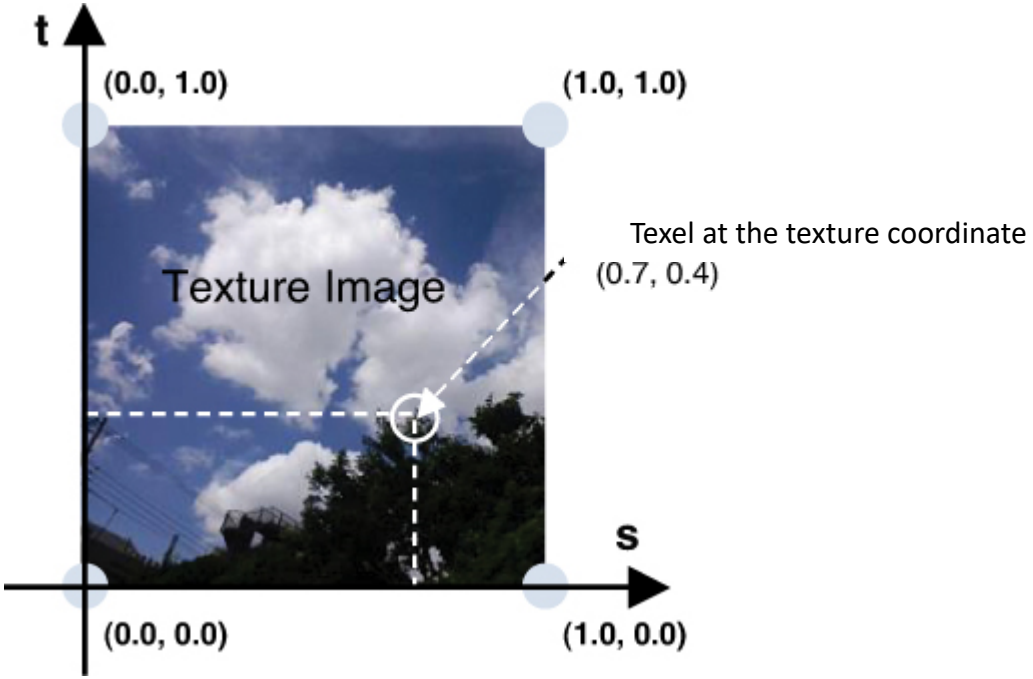
Definición de texturas

- Una textura 2D es una función de dos parámetros

  Tex(s,t): $\Re x \Re$ -> Colores (téxel)

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

  T(s,t): [0,1]x[0,1] -> Colores (téxel)

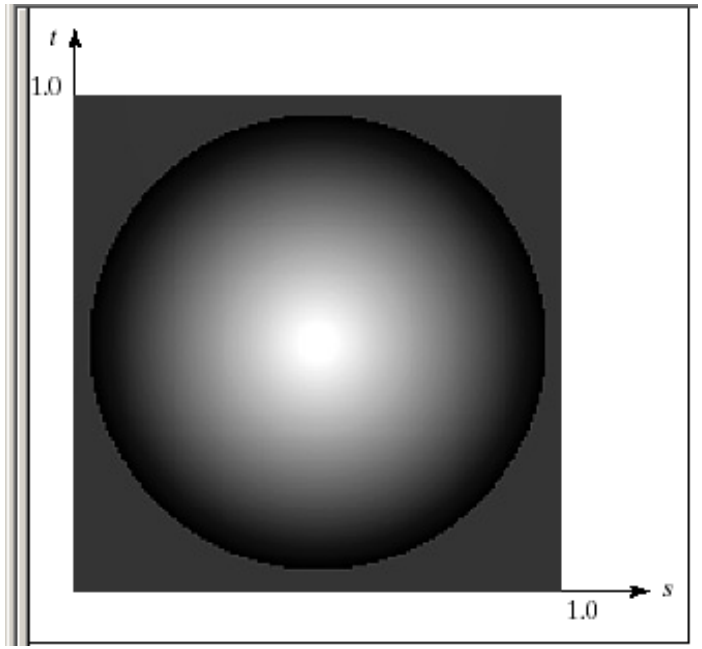


Definición de texturas

- Definición procedimental. Ejemplo para colores de una sola componente de tipo double:

```
double TexturaProc (double s, double t)  { // en [0,1]
   double r = sqrt((s-0.5)*(s-0.5) + (t-0.5)*(t-0.5));
   if (r < 0.3) return 1 - (r / 0.3);  // intensidad de la esfera
   else return 0.2;  // background
}
```

Los puntos dentro del radio de la esfera son más oscuros en el borde (r≈0.3) y más claros en el centro (r≈0.0)



Definición de texturas

- Definición con imágenes rasterizadas.

  Ejemplo para una imagen de NCxNF colores RGBA

```
RGBA TexturaBMP (double s, double t)  // en [0,1]
{  // suponiendo RGBA img[NC][NF]
  return img[trunc(s*NC)] [trunc(t*NF)];  (*)
}
```

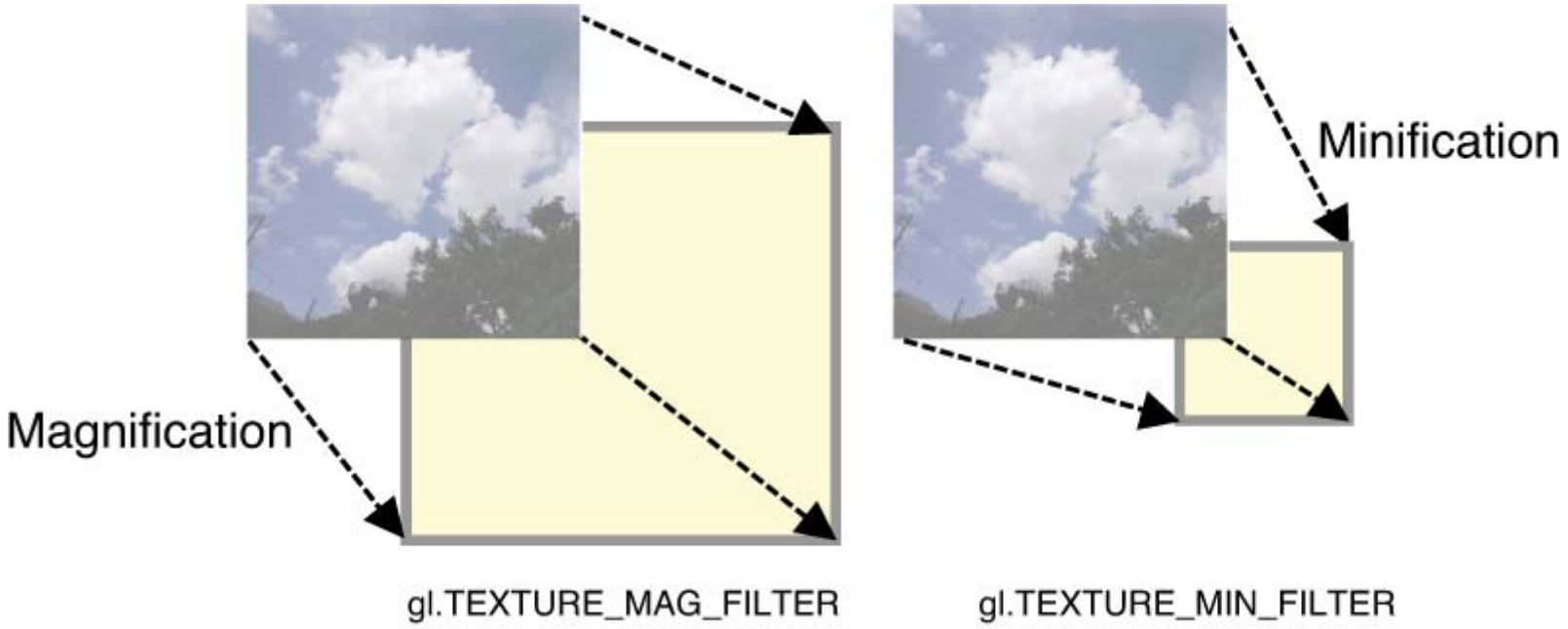
  Algunos valores se repetirán y otros se saltarán, dependiendo de que sea necesario estirar o encoger la textura al aplicarla sobre una malla.

  Para evitar estos problemas se aplican filtros en (*).

Aplicación de Filtros

- En caso de tener que aumentar o reducir la imagen durante el renderizado, se pueden aplicar filtros .

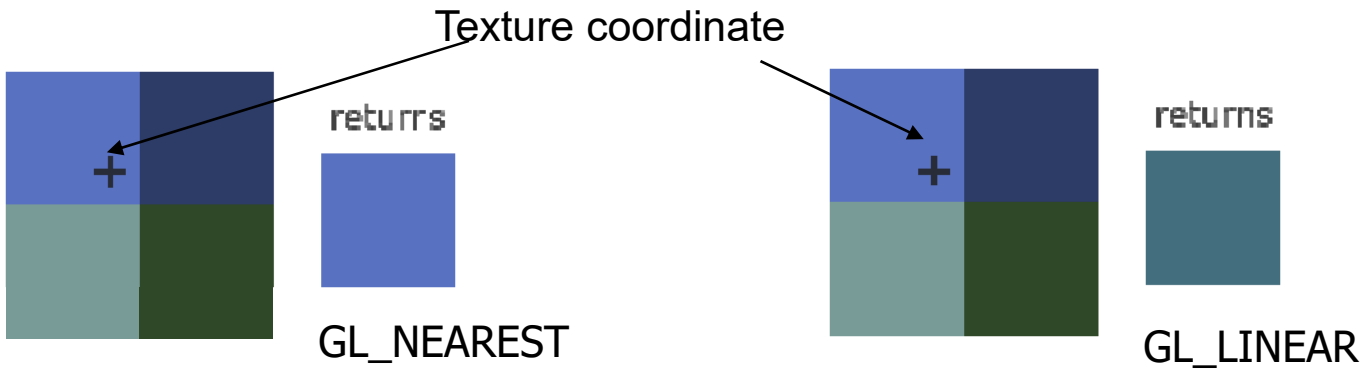
  Por ejemplo realizar una media de los cuatro valores más cercanos. También se utilizan imágenes de distintas resoluciones (mipmaps).



Aplicación de Filtros

- Magnificación del téxel: Un téxel -> varios píxeles
  Un píxel representa una fracción de un téxel
- Minificación del téxel: Un téxel -> una fracción de un píxel
  Un píxel representa una colección de téxels
- Filtros: qué téxels se utilizan para determinar el color de cada píxel

  GL_NEAREST: Se toma el téxel más cercano a las coordenadas de textura del píxel
  GL_LINEAR: Se toma la media de los cuatro téxels más cercanos



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.
- Pegándola en una zona concreta de la superficie.

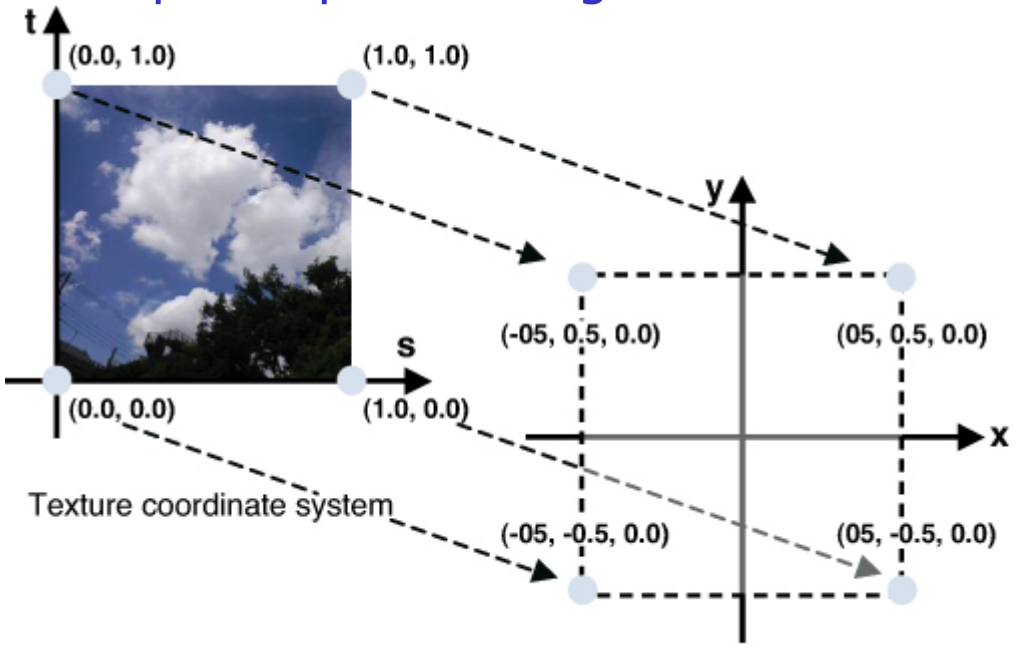
Deben de preservarse las proporciones de la textura para evitar distorsiones de la imagen de la textura.

Aplicación de una textura a una malla

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

  Map: Vértices de la malla ($\Re x \Re x \Re$) -> [0,1]x[0,1]

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



Vertices coordinates

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 array de coordenadas de textura (análogo al array de colores pero de 2 coordenadas):

```
glm::dvec2 * texCoords; // array de coordenadas de textura
```

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

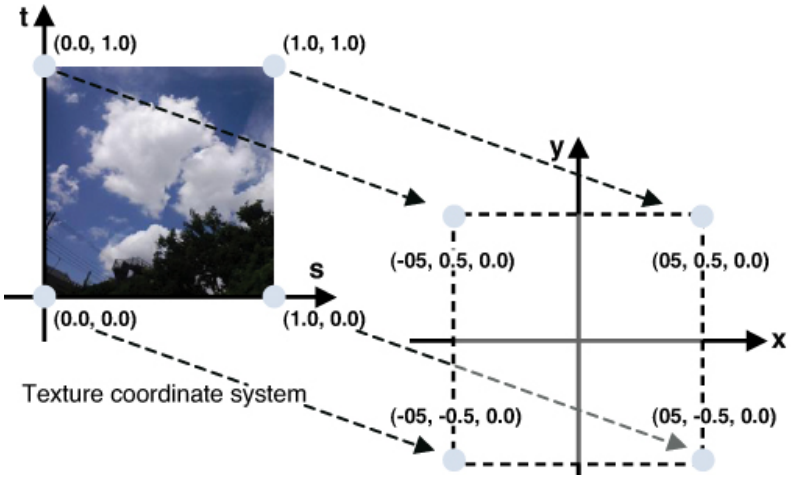
```
glEnableClientState(GL_TEXTURE_COORD_ARRAY);
glTexCoordPointer(2, GL_DOUBLE, 0, texCoords);
```

- Añadimos una nueva clase, Texture, con métodos para cargar de archivo una imagen (load), y activar (bind) y desactivar (unbind) la textura en la GPU.
- Añadimos a la clase Entity un atributo para la textura, que habrá que cargar al inicio, y activar/desactivar al renderizar.

Aplicación de una textura a una malla

- Ejemplo: toda la textura en un rectángulo

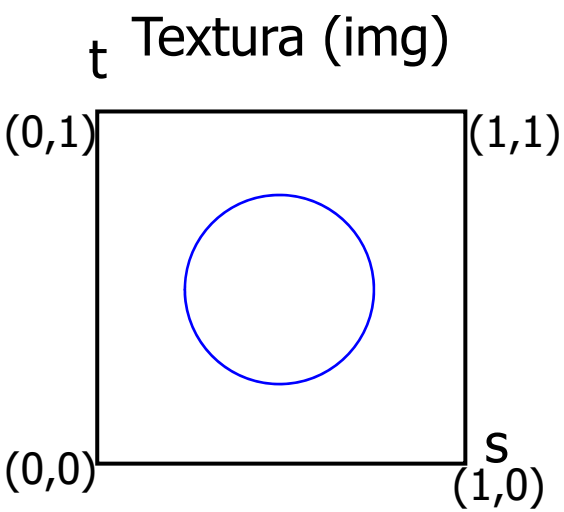
```
Mesh* Mesh::generateRectangleTex(GLdouble w, GLdouble h)
{
    Mesh *m = generateRectangle(w, h);
    m->texCoords = new dvec2[m->numVertices];
    m->texCoords[0] = dvec2(0, 1);
    m->texCoords[1] = dvec2(0, 0);
    m->texCoords[2] = dvec2(1, 1);
    m->texCoords[3] = dvec2(1, 0);
    return mesh;
}
```

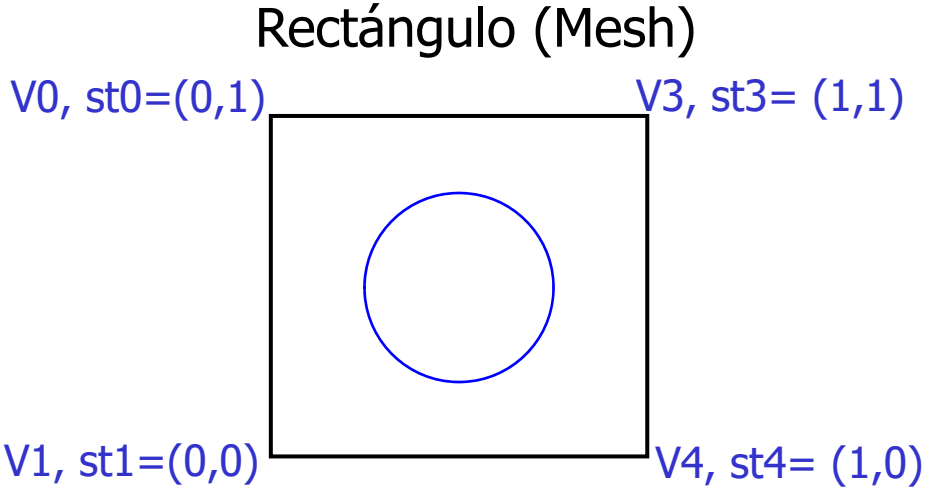


Mesh: vertices coordinates

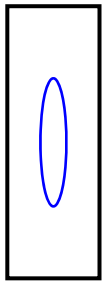
Aplicación de una textura a una malla

- Ejemplo: dependiendo de las dimensiones del rectángulo

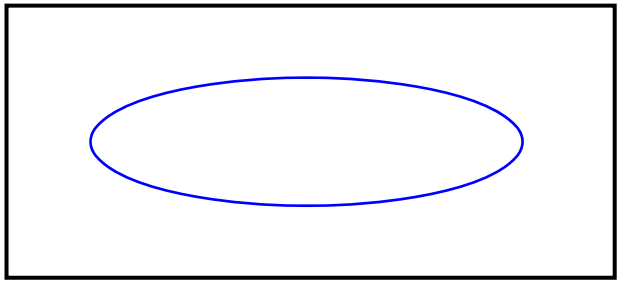




Rectángulo

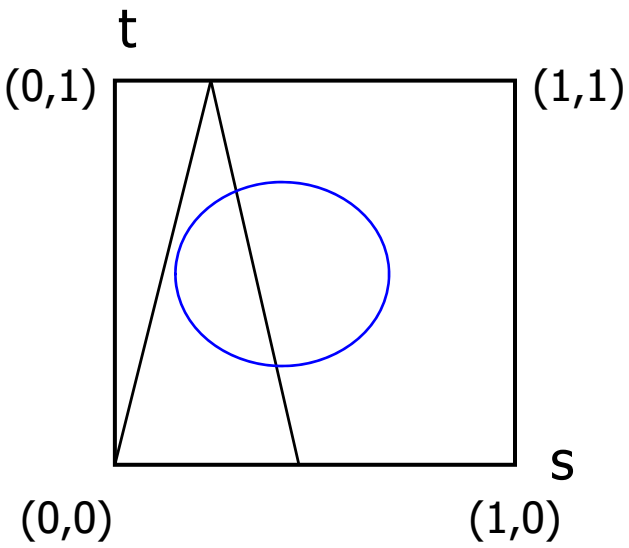


Rectángulo

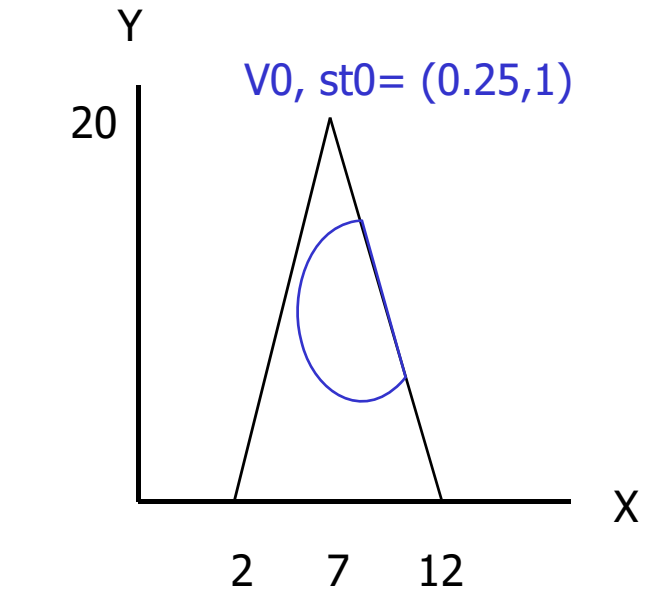


Aplicación de una textura a una malla

- Ejemplo: Parte de una textura en un triángulo



Textura

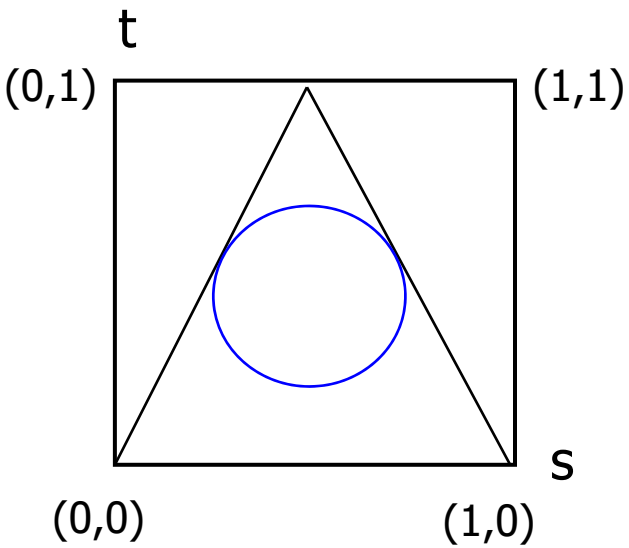


V1, st1= (0,0) V2, st2= (0.5,0)

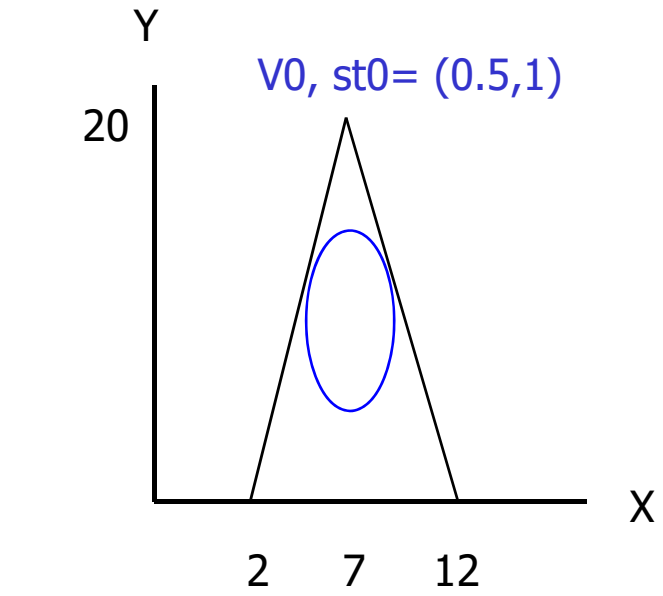
Triángulo

Aplicación de una textura a una malla

- Ejemplo: Parte de una textura en un triángulo



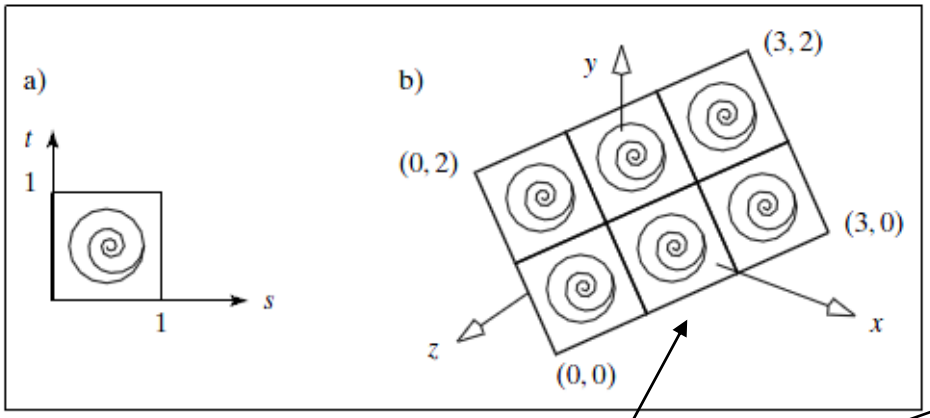
Textura



Triángulo

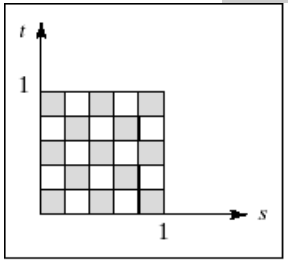
Aplicación de una textura a una malla

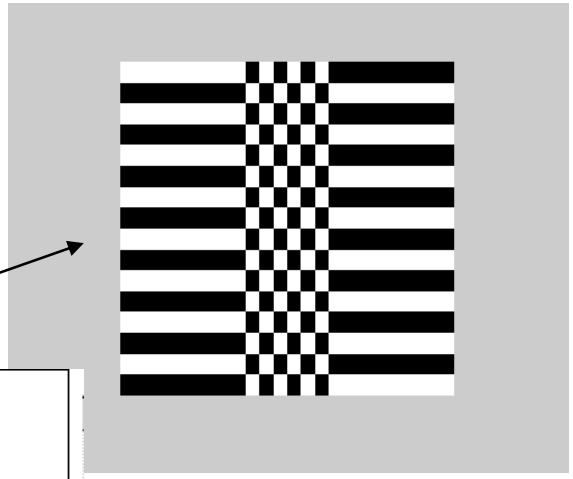
- Texture Wrapping: qué se hace cuando las coordenadas de textura caen fuera del rango [0, 1].
  GL_REPEAT: la textura se repite (tiling). Se ignora la parte entera de las coordenadas de textura.
  GL_CLAMP: coordenadas de textura superiores a 1 se ajustan a 1, y las coordenadas inferiores a 0 se ajustan a 0.



GL_REPEAT en s y t

Rectángulo

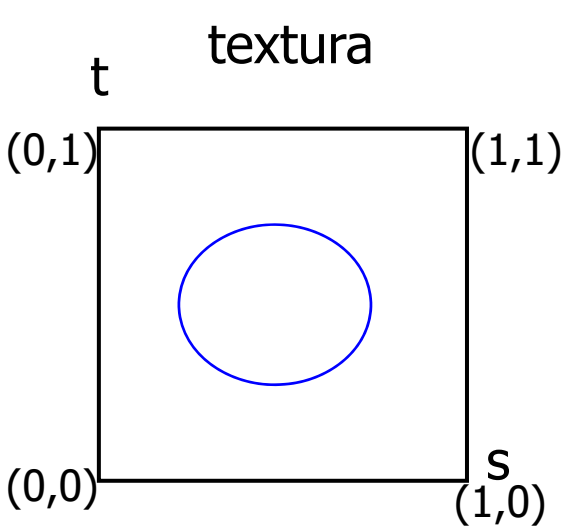


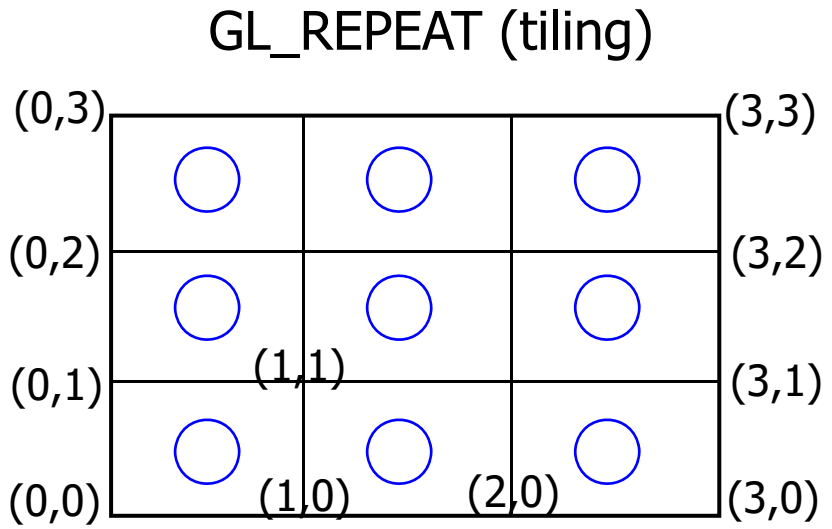


GL_CLAMP en s y GL_REPEAT en t

Aplicación de una textura a una malla

- Ejemplo: malla de una superficie





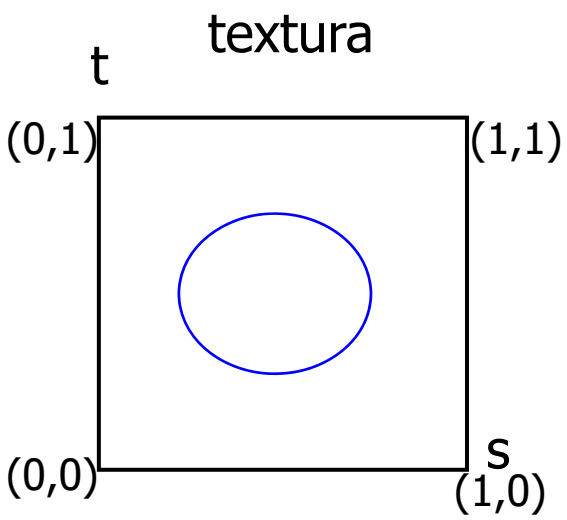
Malla de una superficie:

Tablero WxH dividido en NF x NC x 2 triángulos

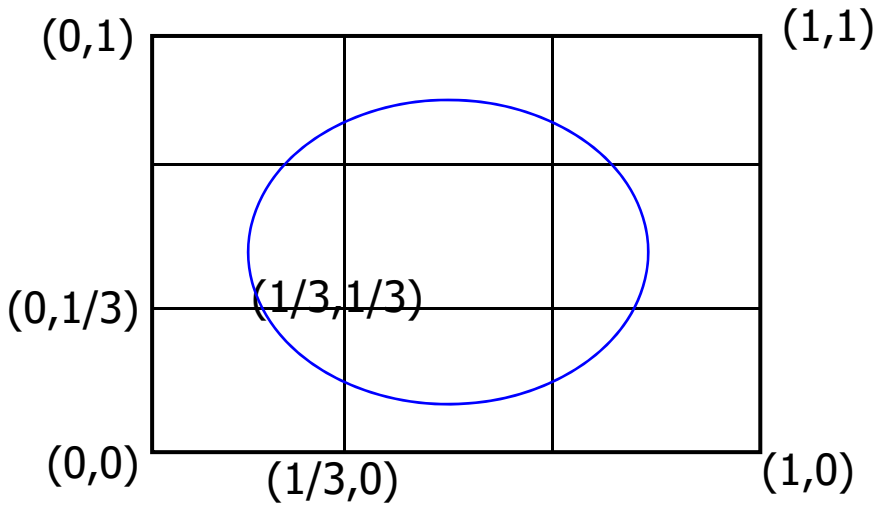
Cubo (contorno): malla de 1 x 4 x 2 triángulos

Aplicación de una textura a una malla

- Ejemplo: malla de una superficie



Proporcional:
textCoords(i, j)=(i/NC, j/NF)



Malla de una superficie:

Tablero WxH dividido en NF x NC x 2 triángulos

Cubo (contorno): malla de 1 x 4 x 2 triángulos

Mezcla de la textura con el color

En el Fragment Shader cada fragmento consta de las coordenadas (x, y) del píxel, y de un color C. Si además tiene coordenadas de textura (s,t), el color C se mezcla con el color de la textura T(s, t).

Las formas más habituales de combinar estos colores son:

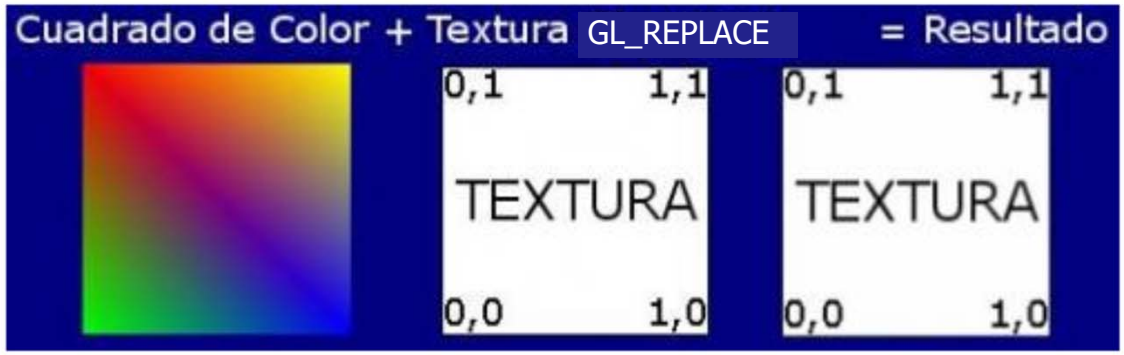
- GL_REPLACE. Utilizar exclusivamente la textura: C = T(s,t)
- GL_MODULATE. Modular ambos colores: C = C * T(s,t)
- GL_ADD. Sumar ambos colores: C = C + T(s,t)
- GL_DECAL (para texturas RGBA). C = (1-At).C + At.T(s,t)

  siendo At la componente alfa de T(s,t)

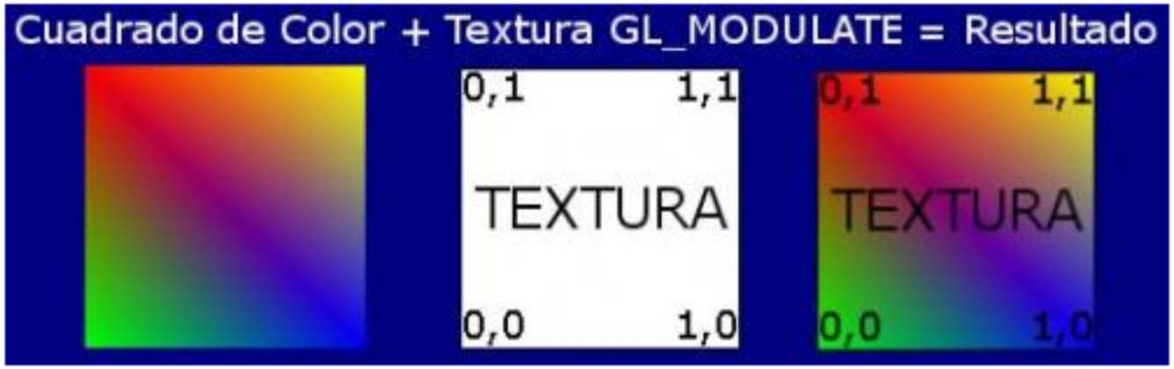
El color resultante se escribirá en el Color Buffer

Mezcla de la textura con el color

GL_REPLACE. Utilizar exclusivamente la textura: C = T(s,t)

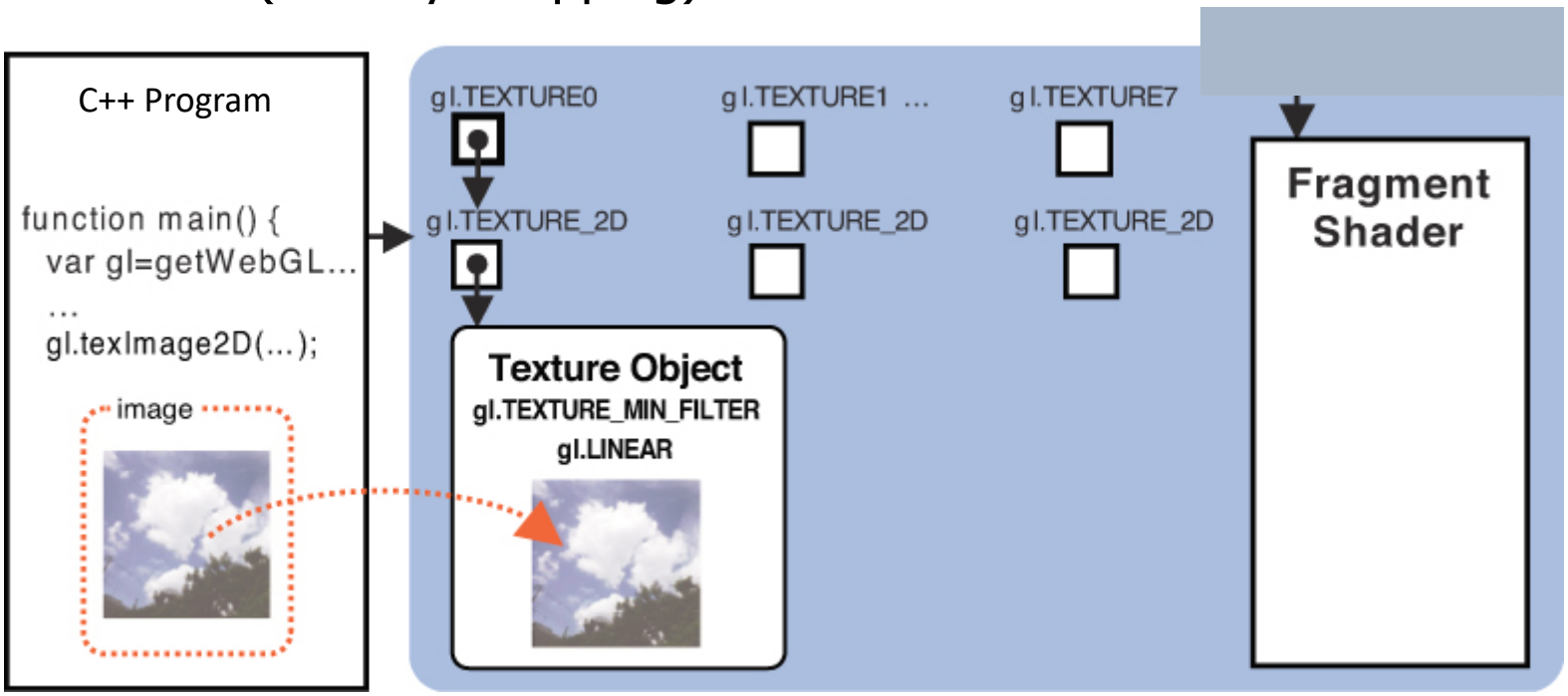


GL_MODULATE. Modular ambos colores: C = C * T(s,t)



Texturas 2D en OpenGL

- En OpenGL las texturas se gestionan mediante objetos de textura: estructuras GPU que contienen la imagen y la configuración de la textura (filtros y wrapping)



- Hay que activar el uso de texturas con
  glEnable(GL_TEXTURE_2D); // -> scene::init()

- Gestión de objetos de texturas
  - Crearlos y destruirlos: glGenTextures(…), glDeleteTextures(…)
  - Configurarlos (filtros, …): glTexParameter*(…)
  - Activarlos para que tengan efecto: glBindTexture(…), glTexEnvi(…)
  - 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,   // -> 0
    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
)
```

Clase Textura

```
class Texture     // utiliza la clase PixMap32RGBA para el método load
{
public:
   Texture(): w(0), h(0), id(0) {};
   ~Texture() {if (id !=0 ) glDeleteTextures(1, &id); };
   void load(const std::string & BMP_Name, GLubyte alpha = 255);
                                                  // cargar y transferir a GPU
   void bind(GLint modo = GL_REPLACE);   // para mezcla de colores
   void unbind() { glBindTexture(GL_TEXTURE_2D, 0); };
protected:
   GLuint w, h;  // dimensiones de la imagen
   GLuint id;     // identificador interno (GPU) de la textura
   void init();
};
```

Clase Textura

```
void Texture:: init() {
    glGenTextures(1, &id);  // genera un identificador para una nueva textura
    glBindTexture(GL_TEXTURE_2D, id);       // 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(GLint modo) {  // modo para la mezcla los colores
    glBindTexture(GL_TEXTURE_2D, id);   // activa  la textura
    // la función de mezcla de colores no queda guardada en el objeto
    glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, modo);
                                // modos: GL_REPLACE, GL_MODULATE, GL_ADD …
}
```

Clase Textura

```
void Texture:: load(const std::string & BMP_Name, GLubyte alpha) {
    if (id == 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);
    w = pixMap.width();
    h = pixMap.height();
    glBindTexture(GL_TEXTURE_2D, id);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA,
                        GL_UNSIGNED_BYTE, pixMap.data());
                                // transferir a GPU
}
```

Entidad con textura

- Añadimos a la clase Entity un atributo para la textura:

```
Texture texture;     // w, h, id
```

- La entidad necesita una malla con coordenadas de textura y cargar la imagen que queremos usar

```
void RectangleTex::RectangleTex(…) {
   mesh = Mesh::generateRectangleTex(….); // con coord. de textura
  textura.load("..\\Bmps\\Zelda.bmp");  // cargamos la imagen
  …
}
```

- Recuerda activar y desactivar la textura en el método render()

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

```
glCopyTexImage2D(GL_TEXTURE_2D, level, internalFormat,
                                    xleft, ybottom, w, h, border);
                          // 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, format, type, pixels);
      // pixels-> array donde guardar los datos (de tipo y tamaño adecuado)
```