Gráficos por Computador

- Texturas
  - Definición
  - Aplicación a una malla
  - Combinación de la textura con el color
  - Objetos de textura en OpenGL

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

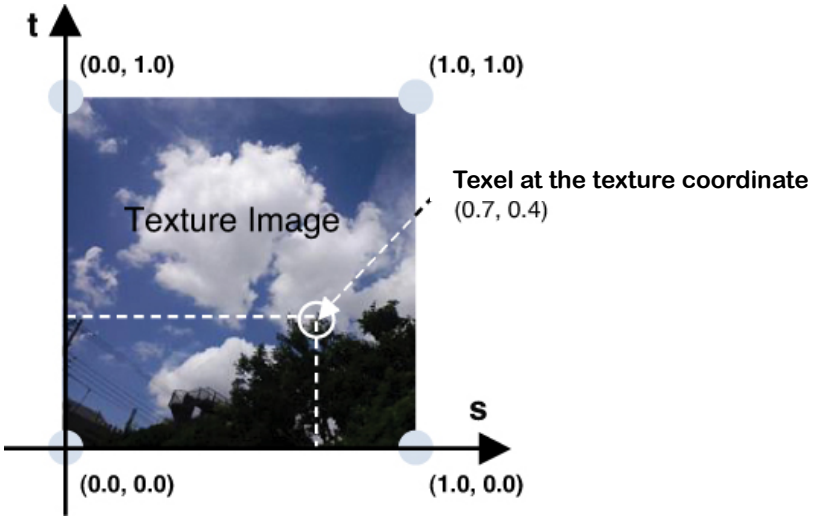
Definición de texturas

Una textura 2D es una función de dos parámetros

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

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

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



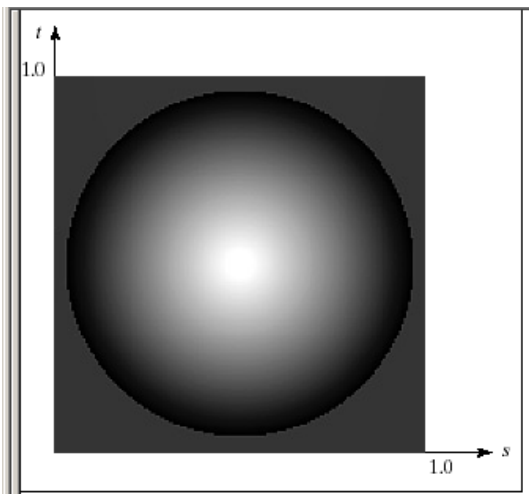
Definición de texturas

- Una textura se puede definir:
  - De forma procedimental
  - A partir de una imagen rasterizada
- Los Colores pueden ser
  - RGB, RGBA, L (brillo), ...

Definición de texturas

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

```
double TexturaProc (double s, double t)  {
  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
}
```



Definición de texturas

- Mediante imágenes rasterizadas. Ejemplo para imagen de NCxNF colores RGB

```
RGB TexturaBMP (double s, double t)
{  // suponiendo RGB mat[NC][NF]
  return mat [trunc(s*NC)] [trunc(t*NF)];  (*)
}
```

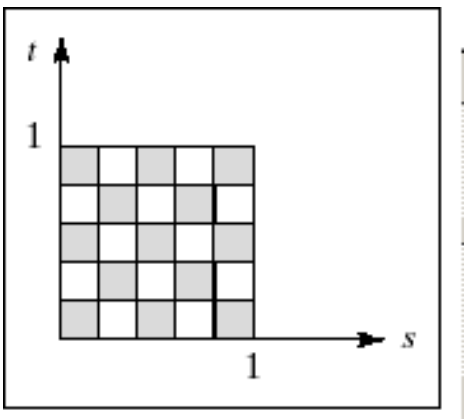
Algunos valores se repetirán y otros se saltarán, dependiendo de que sea necesario estirar o encoger la textura al aplicarla sobre una malla.

Para evitar estos problemas se aplican filtros en (*), como por ejemplo realizar una media de los cuatro valores más cercanos, o matrices de distintas resoluciones (multirresolución).

Definición de texturas

- También podemos definir de forma procedimental matrices. Ejemplo para colores de 4 componentes Glubyte

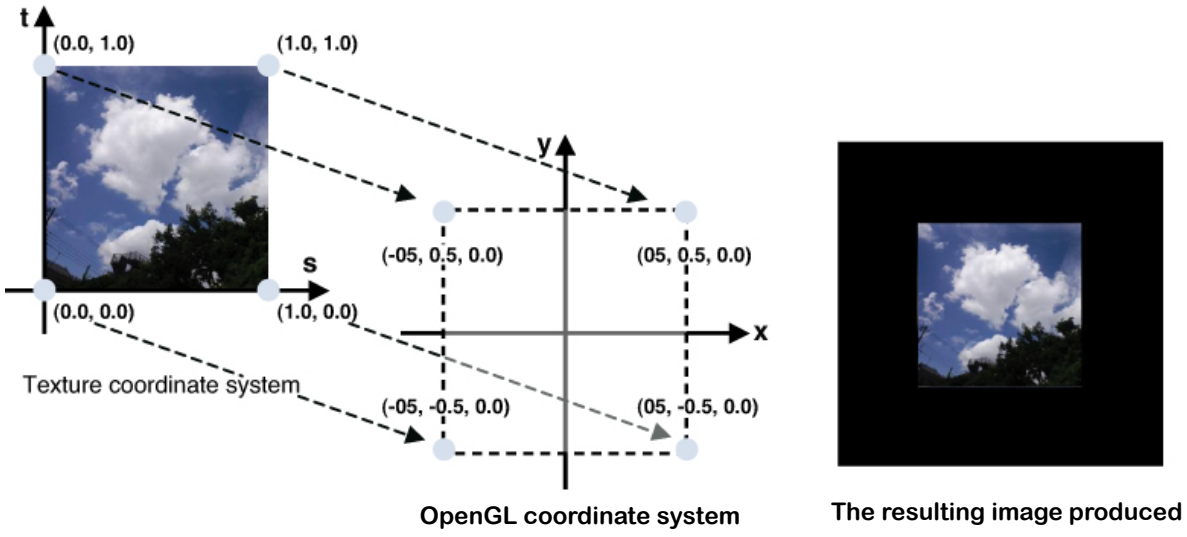
```
void TexturaProc (GLubyte mat[NC][NF][4]) {
  for (int i=0; i<NC; i++)
    for (int j=0; j <NF; j++) {
      int c=(par(i+j))*255;
      mat[i][j][0]= Glubyte (c);
      mat[i][j][1]= Glubyte (c);
      mat[i][j][2]= Glubyte (c);
      mat[i][j][3]= Glubyte(255);
    }
}
```



Aplicación de la textura a una malla

Texture mapping o inverse 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]



Aplicación de texturas a mallas

- A cada vértice hay que asignarle sus coordenadas de textura (s,t) añadiendo a la clase Malla una tabla de coordenadas de textura (análoga a la tabla de vectores normales pero de 2 coordenadas):

```
CTex2 * coorTex; // tabla de coordenadas de textura
```

Para

```
class CTex2 { public: Gldouble s, t; ...};
```

El método Malla::activar() tiene que activar la tabla de coordenadas de textura (GL_TEXTURE_COORD_ARRAY) y la textura que se quiere aplicar.

El método Malla::desactivar() tiene que desactivarlas.

Aplicación de la textura a una malla

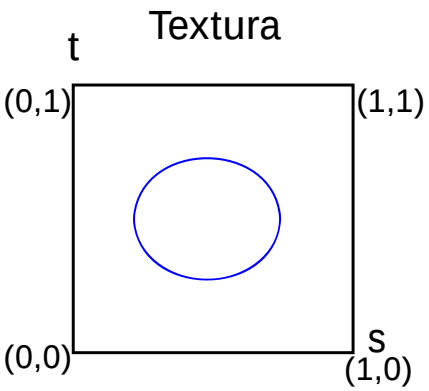
La textura puede aplicarse de dos formas:

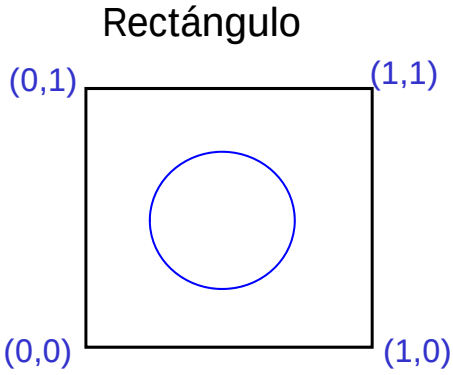
- Recubriendo toda la superficie del objeto con la textura.
- Pegándola en una zona concreta de la superficie.

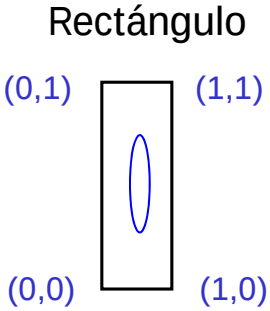
Deben de preservarse las proporciones de la textura para evitar distorsiones de la imagen de la textura.

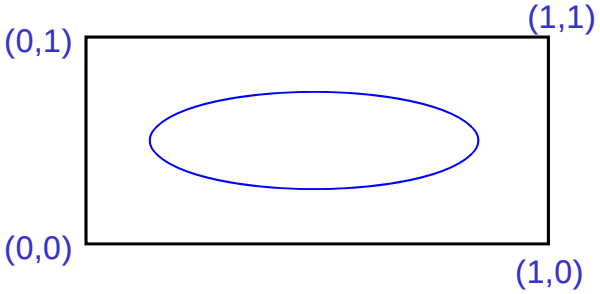
Aplicación de una textura a una malla

Ejemplo: toda la textura en un rectángulo



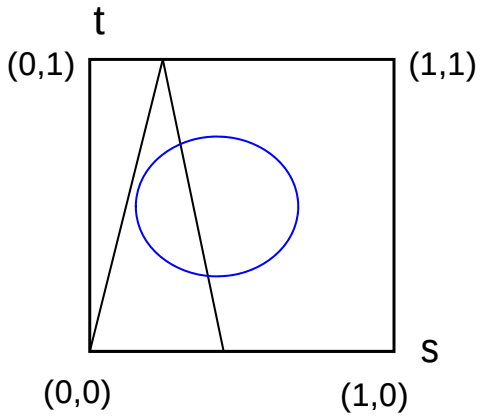




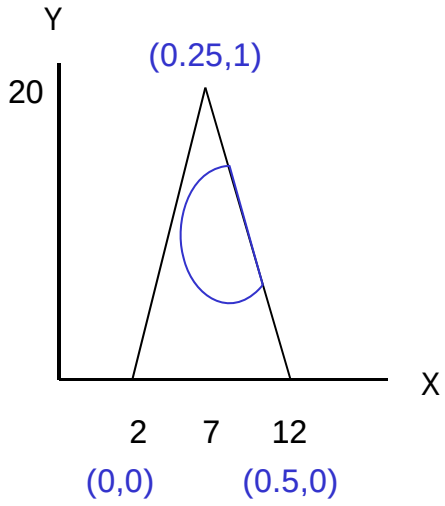


Aplicación de una textura a una malla

Ejemplo: Parte de una textura en un triángulo



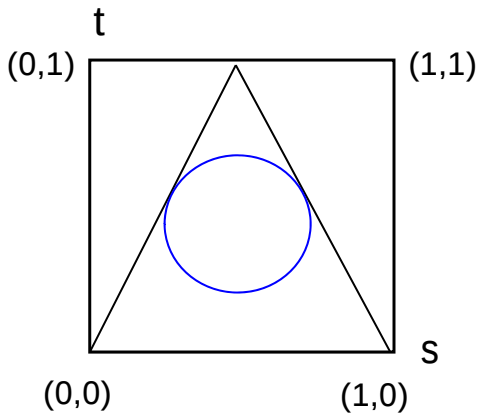
Textura



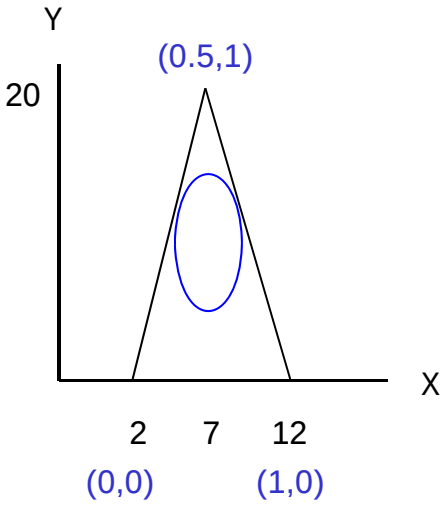
Triángulo

Aplicación de una textura a una malla

Ejemplo: Parte de una textura en un triángulo



Textura



Triángulo

Combinación de la textura con el color

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

Las formas más habituales de combinar estos colores son:

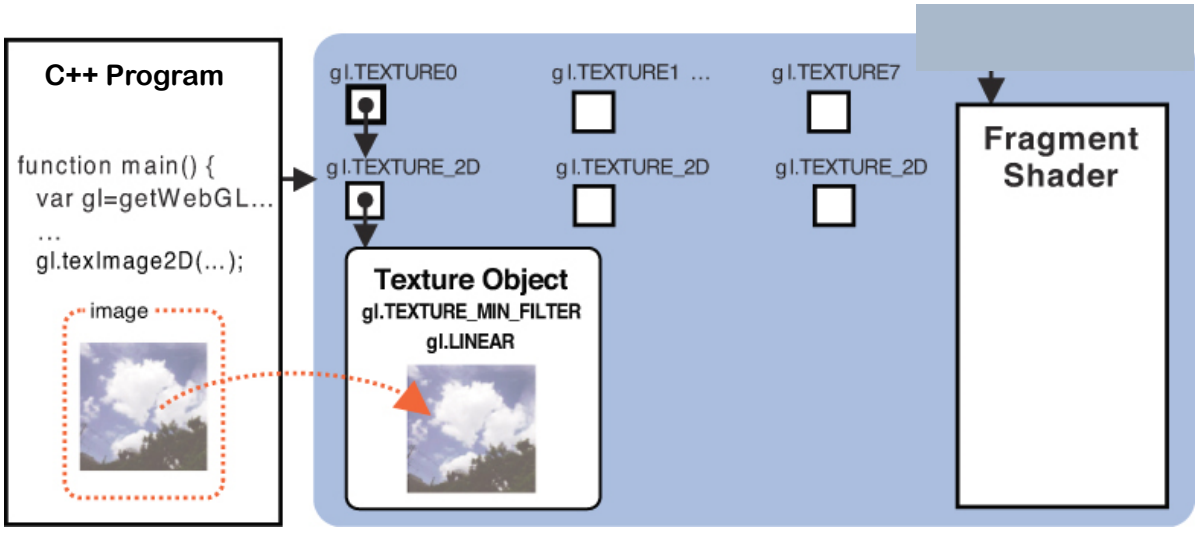
- Utilizar exclusivamente la textura: C = T(s,t)
- Modular ambos colores: C = C .T(s,t)
- Decal (para texturas RGBA): C = (1-At).C + At.T(s,t)

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

El color resultante se escribirá en el Color Buffer si la coordenada z pasa el test de profundidad (GL_DEPTH_TEST)

Texturas 2D en OpenGL

En OpenGL las texturas se gestionan mediante objetos de textura.



Texturas 2D en OpenGL

OpenGL incluye funciones para:

La gestión de objetos de texturas.

1- Generar nombres para los objetos de textura

2- Crearlos y destruirlos

3- Configurar sus propiedades y asociarles la imagen

4- Activarlos para que tengan efecto

Aplicar texturas a las mallas.

1- Activar la textura (objeto) que se quiere aplicar

2- Configurar la combinación con el color

3- Asignar coordenadas de textura a los vértices de los triángulos de la malla.

Clase Textura

```
class Textura {
public:
   Textura(): w(0), h(0), id(0) {};
   ~Textura() { glDeleteTextures(1, &id); };
   void init() { glGenTextures(1, &id);  // generar un nombre
                 activar(); glTexParameteri(...); }; // establecer filtros
   void activar() { glBindTexture(GL_TEXTURE_2D, id); glTexEnvf(...);};
   void desactivar() { glBindTexture(GL_TEXTURE_2D, 0); };
   bool load(const std::string & BMP_Name); // cargar y transferir a openGL
   void save(const std::string & BMP_Name); // obtener de openGL y guardar
public:
   GLuint w, h;  // dimensiones de la imagen
   GLuint id;     // identificador interno de la textura
};
```

Clase Textura

```
bool Textura::load(const std::string & BMP_Name) {
                    // la textura debe estar inicializada -> escena::init()
   PixMap24RGB pixMap;
   pixMap.load_bmpBGR(BMP_Name);          // cargar
   // carga correcta??
   w = pixMap.width();
   h = pixMap.height();
   glBindTexture(GL_TEXTURE_2D, id);          // transferir a openGL
   glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, w, h, 0, GL_RGB,
                    GL_UNSIGNED_BYTE, pixMap.map());
   ...
}
```


Clase Textura

```
void Textura:: save(const std::string & BMP_Name) {
   glGetTexImage(GL_TEXTURE_2D, 0, GL_RGB,      // obtener de openGL
                    GL_UNSIGNED_BYTE, pA);
            // pA-> array donde guardar los datos (de tipo y tamaño adecuado)

   ... // y guardar
}
```

Rectángulo con textura

- Para utilizar una textura en un rectángulo añadimos a la escena dos atributos:

  Textura textura; Rectangulo recTex;

- Activamos las texturas en OpenGL y cargamos la imagen

```
void Escena::init() {
   glEnable(GL_TEXTURE_2D);
   textura.init();
   textura.load(...);
   ...
}
```

- El rectángulo debe constar de las tablas de vértices y coordenadas de textura, vector normal y color. Recuerda activar y desactivar la textura en los métodos activar() y desactivar().

Texturas 2D en OpenGL

Lo primero de todo es activar las texturas con el comando:

glEnable(GL_TEXTURE_2D);

que podemos desactivar con:

glDisable(GL_TEXTURE_2D);

Generar nombres para los objetos de textura

```
GLuint Name;   GLuint Names[3];
glGenTextures(1, &Name);
glGenTextures(3, Names);
```

Los nombres que se generan se devuelven en el segundo parámetro. No son consecutivos.

El **0** nunca se devuelve como nombre, pero podemos utilizarlo para que no esté activo ningún objeto de textura.

Crear objetos de textura y activarlos

El comando para crear un objeto de textura es el mismo que para activarlo. Si se activa un objeto que no existe, se crea y queda activo. Los demás comandos sobre texturas se ejecutan sobre el objeto activo.

```
glBindTexture(GL_TEXTURE_2D,  Name);
glBindTexture(GL_TEXTURE_2D,  Names[i]);
```

Para desactivar la textura activa:

```
glBindTexture(GL_TEXTURE_2D,  0);
```

- **Liberar objetos de texturas**

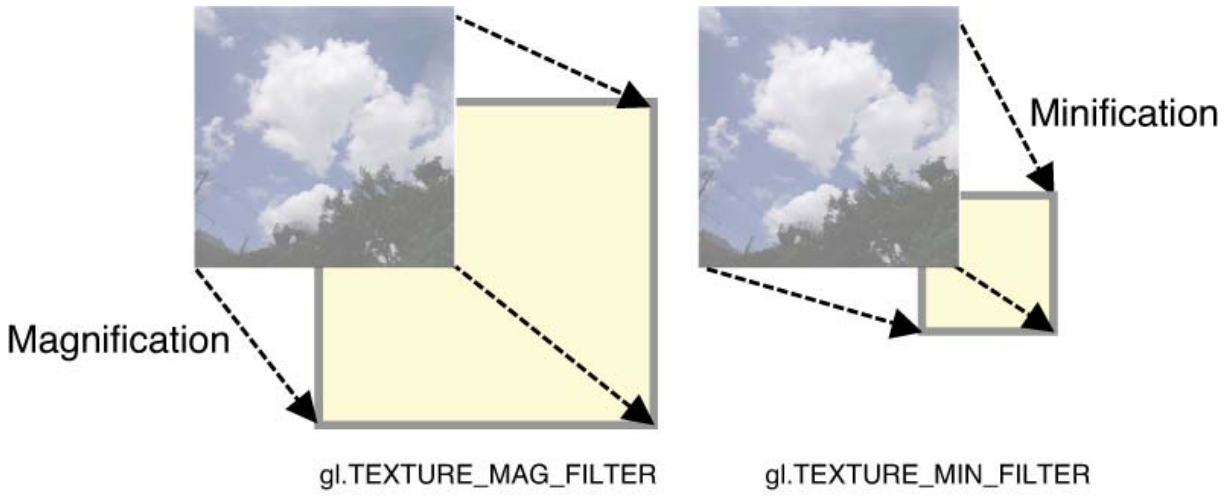
```
glDeleteTextures(1,  &Name);
glDeleteTextures(3,  Names);
```

- **Configurar los filtros para el objeto de textura activo**

Antes de asociarle la imagen debemos de especificar que proceso seguir en caso de tener que aumentar o reducir la imagen durante su aplicación.

Debe evitarse que sea necesario aumentar (en una dirección) y disminuir (en la otra) simultáneamente en la aplicación de la textura.

Filtros para aplicar en caso de tener que aumentar o reducir la imagen durante su aplicación:



Gestión de objetos de textura: Filtros

glTexParameteri(GL_TEXTURE_2D, TipoProceso, Proceso);

TipoProceso: **GL_TEXTURE_MAG_FILTER** (para aumentar)

Proceso: **GL_NEAREST** (el más cercano) **/**

GL_LINEAR (valor por defecto, una combinación de los 4 más cercanos)

TipoProceso: **GL_TEXTURE_MIN_FILTER** (para reducir)

Proceso: **GL_NEAREST / GL_LINEAR /**

GL_NEAREST_MIPMAP_LINEAR (valor por defecto, pero sólo funciona en el caso de multirresolución)

glTexParameteri(GL_TEXTURE_2D,

GL_TEXTURE_MIN_FILTER, GL_LINEAR);

Gestión de objetos de textura

- **Asociar la imagen de la textura al objeto activo**

glTexImage2D(GL_TEXTURE_2D, Level, GL_RGB[A], NCols, NFils, Border, GL_RGB[A], GL_UNSIGNED_BYTE, Data);

Level: el nivel de detalle (multirresolución). Para un único nivel debemos poner 0.

Border: es un booleano 0/1 que indica si la imagen tiene borde.

NCols, NFils: deben ser potencias de 2 si la imagen no tiene borde, o potencias de 2 más 2 si la imagen tiene borde.

Data: el array con la imagen que se quiere usar como textura. El formato del array debe ser el especificado y a continuación puede liberarse. Por ejemplo: GLubyte data[NCols * NFils * 3];

Combinación de la textura con el color

- **Configurar la combinación con el color**

  Hay que establecer la combinación cada vez que se usa la textura.

  glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, FunMix);

  FunMix: puede tomar los valores (entre otros):

  GL_DECAL (C=(1-At).Cf + At.Ct y A=Af)

  GL_REPLACE (C=Ct y A=At)

  GL_MODULATE (defecto, C=Ct.Cf y A=At.Af)

  Donde (C,A) es el RGBA resultado de la combinación, (Ct,At) es el RGBA de la textura y (Cf,Af) es el RGBA del color del fragmento.

Objetos de textura

- **Copiar en la textura activa la imagen del Color Buffer**

  glCopyTexImage2D(GL_TEXTURE_2D, level, internalFormat, xl, yb, w, h, border);

  // en coordenadas de pantalla (puerto de vista)

- **Obtener 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)

Aplicación de texturas a mallas

Una textura 2D normalizada es una función de dos parámetros

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

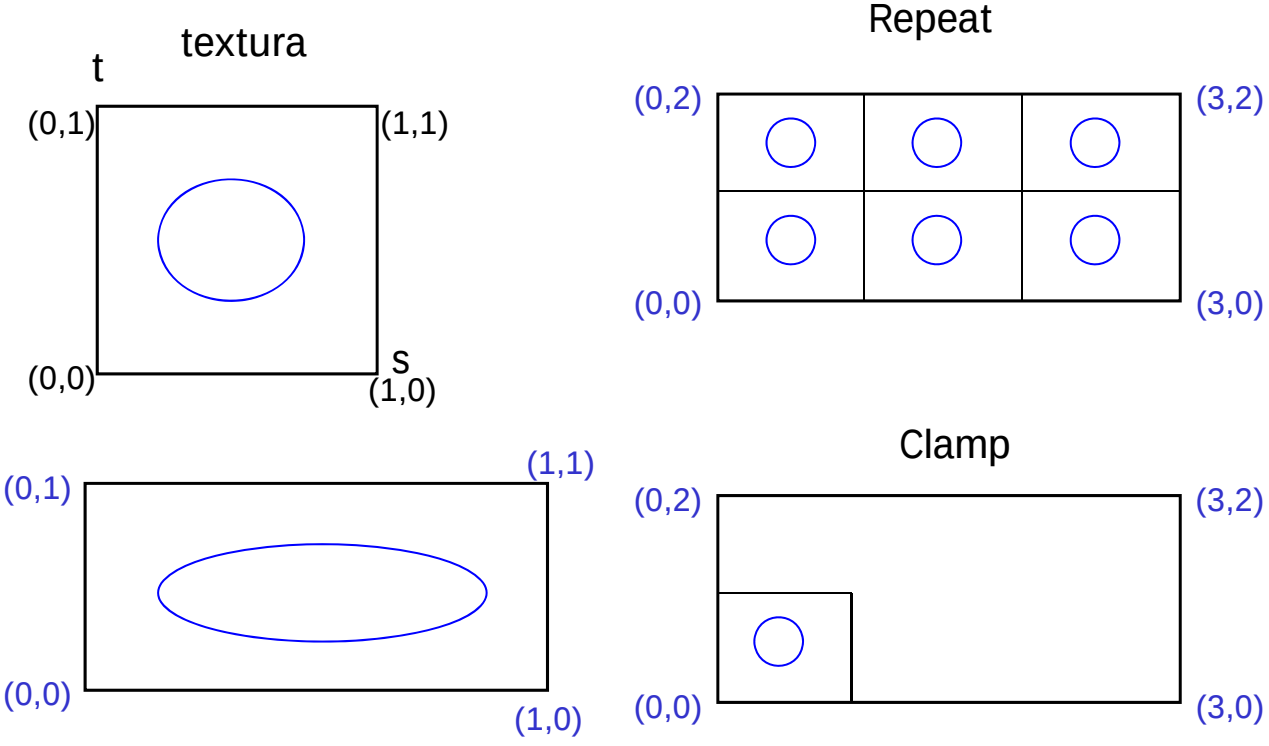
En caso de utilizar coordenadas de textura (s,t) fuera de los intervalos [0,1], podemos indicar como transformarlas a [0,1].

Las formas más simples de pasar (wrap) $\Re$->[0,1] son:

- Quedarse con la parte decimal del valor dado, generando una repetición de la imagen (*REPEAT*).
- Llevar los valores mayores que 1 a 1 y los menores que 0 a 0 (*CLAMP*).

Aplicación de una textura a una malla

Ejemplo: rectángulo



glTexParameteri(GL_TEXTURE_2D, TipoProceso, Proceso);

En caso de utilizar coordenadas de textura (s,t) fuera de los intervalos [0,1], podemos indicar como transformarlas a [0,1]. Se especifica independientemente para cada una de las coordenadas s y t.

TipoProceso: **GL_TEXTURE_WRAP_S** (para la coordenada s)

GL_TEXTURE_WRAP_T (para la coordenada t)

Proceso: **GL_CLAMP / GL_REPEAT**

El valor por defecto, en ambos casos, es *GL_REPEAT*, que se queda con la parte decimal del valor dado, generando una repetición de la imagen (*tiling*). *GL_CLAMP* lleva los valores mayores que 1 a 1 y los menores que 0 a 0.