Activar el modo textura

Es lo primero que se hace (desde la clase Scene) con el comando:

```
glEnable(GL_TEXTURE_2D);
```

Este modo se desactiva con el comando:

```
glDisable(GL_TEXTURE_2D);
```

Generar nombres para los objetos de textura

GLuint Name; GLuint Names[3];

glGenTextures(1, &Name);

glGenTextures(3, Names);

Los nombres se generan en init() de la clase Texture.

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 filtros y envoltorios para el objeto de textura activo

Están configurados en init() de la clase Texture.

Filtrado. especificar qué hacer en caso de tener que aumentar o reducir la imagen durante la aplicación de una textura.

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

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

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 (CUIDADO: 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: Envoltorios

glTexParameteri(GL_TEXTURE_2D, TipoProceso, Proceso);

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.

- **Pasar la imagen a la textura activa** (se invoca en el método load(BMP_Name, alpha) de la clase Texture)

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: tamaño de la imagen (Data).

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** (se invoca en el método render() de cada entidad que la necesite)

  Hay que establecer la mezcla 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.

- **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 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)
```

Fotos

- Se hace una foto cargando el buffer de color como valor del atributo mTexture de la entidad Foto
- Se carga un buffer de color mediante un nuevo método loadColorBuffer(width, height, buffer) de la clase Texture
- Este nuevo método consiste en:
  - Se activa el buffer de color que se va a cargar mediante glReadBuffer()
  - Se activa el objeto de textura de la clase Foto mediante glBindTexture()
  - Se copia la imagen del color buffer activado mediante glCopyTexImage2D()
  - Se reactiva el buffer de color que había mediante glReadBuffer()
- Se guarda la foto mediante un método de Scene que crea una textura, hace la foto con ella (como se acaba de decir) y se guarda la imagen desde GPU hasta CPU mediante un nuevo método save(BMP_Name) de la clase Texture que hace uso del método glGetTexImage(), con GL_UNSIGNED_BYTE como **type**, y del método save_bmp24BGR() de la clase Pixmap32RGBA