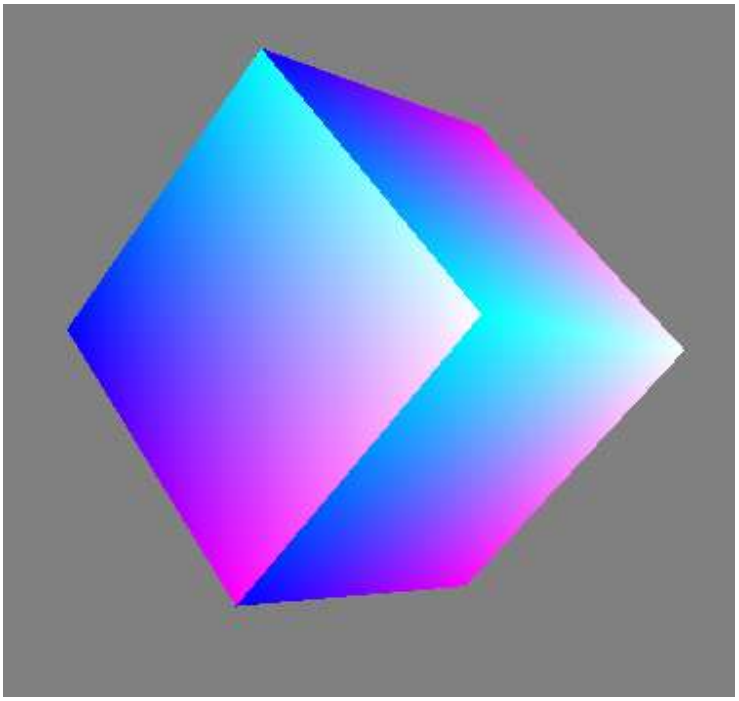
TEXTURING

- Partiendo del ejemplo **fixed_camera_normals** convertir nuestra malla de vértice X,Y,Z,Xn,Yn,Zn en una malla con coordenadas de textura (en flotante): X,Y,Z,Xn,Yn,Zn,U,V. Tenemos que modificar el formato del vértice, y luego el objeto cubo existente para añadirle coordenadas de prueba.

- Adaptar la carga de la malla al nuevo tipo configurando y añadiendo las funciones **glVertexAttribPointer.**
En este punto, el cubo debe verse correctamente sin cambios en las normales.

- Adaptar el vertex shader para que lea las coordenadas de texturas de la malla y las pase al fragment shader.
  Para ello se debe:
  - Crear una variable "in" que use "layout" con el atributo 2
  - Crear una variable "out" con la coordenada
  - Copiar la var. “in” en la variable “out”, tal cual

- Modificar el fragment shader para asegurarnos de que hemos recibido la coordenada, mostrándolo como color del triángulo; para ello creamos una variable "in" de la coordenada UV, usando el mismo tipo y nombre de "out" del vertex shader y lo asignamos al FragColor, adaptando el tipo (vec2 -> vec4).



-Aquí comienza la parte complicada. Debemos crear e inicializar una textura de prueba.

Creamos una función “UploadTexture” que reservará memoria para pixels ARGB/BGRA de 32 bits, de dimensiones w*h y nos devolverá un identificador de textura.

static unsigned int UploadTexture (int w, int h)

La función debe:
-Reservar memoria
-Rellenarla con un patrón, por ej. (x,y,0,255), o (x^y,x,y,255)
-Generar un id. de objeto textura OGL con **glGenTextures**
-Enfocarlo y declarar su tipo con GL_TEXTURE_2D con **glBindTexture**
- Configurar los parámetros de textura basico tipicos

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

- Cargar la textura en VRAM usando **glTexImage2D.** El formato interno podría ser GL_RGBA
- Por último, generar los mipmaps automáticamente con
glGenerateMipmap(GL_TEXTURE_2D);
- Comprobar la textura en este punto es dificil, asi que al menos sería conveniente comprobar el **glGetError**

-Intentaremos ahora comprobar si la textura funciona. Para ello, antes de hacer el draw debemos enfocar la textura y activar un sampler de textura

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, TextureID);
```

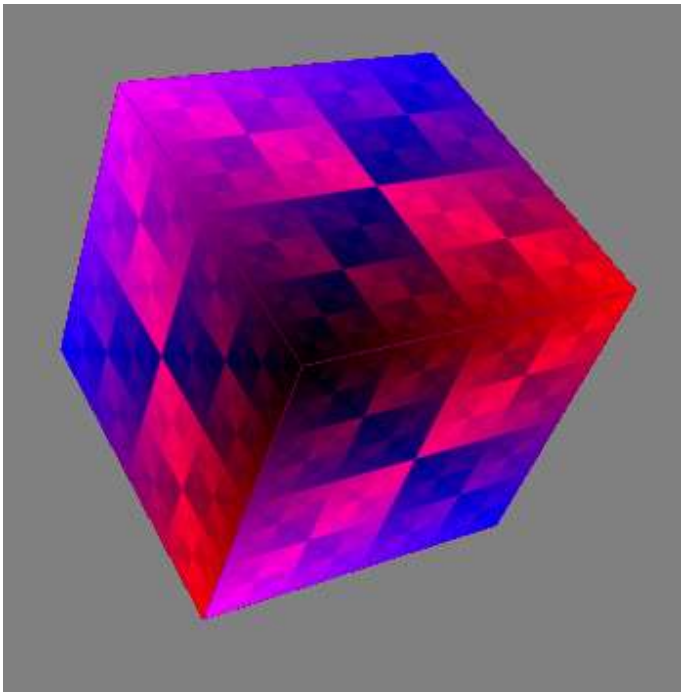
Dentro del fragment shader debemos ahora recibir un uniform de tipo sample2D.

```
uniform sampler2D texture1;
```

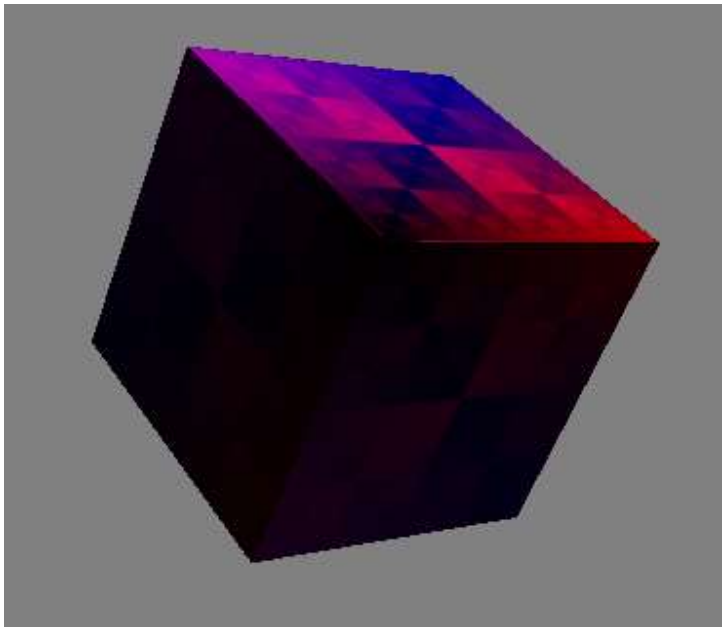
…que se asocia por defecto con la unidad TEXTURE0. Para usar unidades superiores a 0 ya es necesario usar un glUniform1i

Con esto podemos leer la textura:

```
FragColor =texture(texture1, TexCoord);
```



- Por último, multiplicamos la iluminación por el color del texel para obtener:



CARGANDO UNA TEXTURA

Una vez tenemos confirmación de que podemos pasar una textura OGL correctamente, podemos intentar cargar una imagen de disco.

Tenemos 2 opciones:

- Usar alguna librería de imágenes como **stbi_image** para cargar ficheros .**png**, .**jpg**, etc…
- Cargar texturas con formato de compresión específico para GPUs. Estos formatos ocupan poco espacio en disco y VRAM, y aceleran la ejecución de los shaders. En docs/ encontrareis un pequeño texto descriptivo de los diferentes formatos

Cargando una textura DXT1/DDS

En el caso de que queráis intentar la solución profesional, os propongo cargar una textura DXT1 que os paso ya convertida como ejemplo.

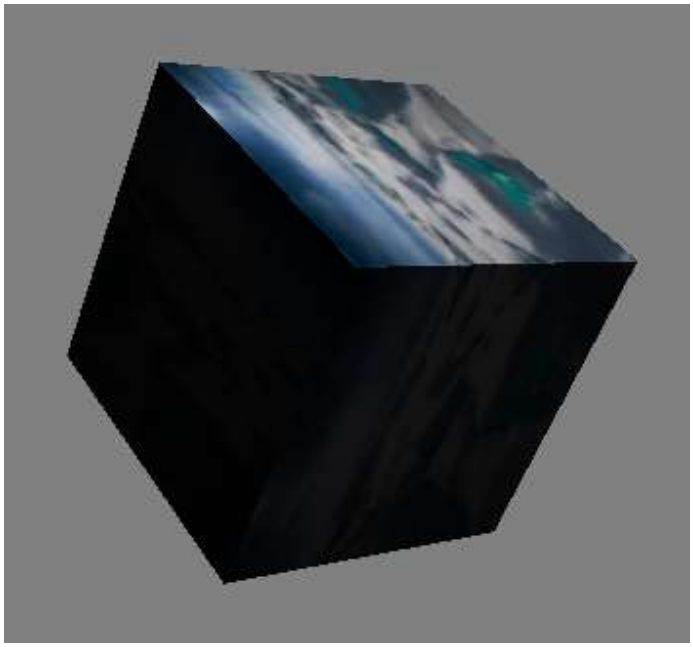
Tambien podeis convertir vosotros una imagen a .dds (dxt1) usando herramientas como gimp, imagemagik, photoshop o incluso conversores online.

Los pasos para cargar la textura serían:

- Leer el fichero (que debéis colocar en el directorio del proyecto o un subdirectorio) completo dejándolo en memoria. Para ello podéis usar una función como ReadSrcCode
- Aseguraros de que los primeros 4 caracteres son 'DDS '
- Saltandonos esa marca mágica, podemos castear la dirección al formato DDSHeader, que os aporto en un include
- Los datos a subir se encuentran justo después del header. Usando un cast a (unsigned char*) podemos empezar a subir datos a OGL, pero en lugar de usar **glTexImage2D** usaremos **glCompressedTexImage2D**.
- **glCompressedTexImage2D** es una función más sencilla, que solo requiere un formato de textura como argumento, el mismo para fuente y destino. Como argumento le daremos este #define(esto es un pequeño hack, no es la forma correcta de incluirlo).

```
#define GL_COMPRESSED_RGBA_S3TC_DXT1_EXT (0x83F0)
```

- Ya podemos comprobar si sale, pero no os olvidéis desactivar los mipmaps



- El último paso sería darle todos los mipmaps. El número de mipmaps podéis encontrarlo en la estructura del headers.
  Mediante un for debeis ir llamando **glCompressedTexImage2D** que recibe el número de mipmap y su dirección de memoria. Todos los mipmaps se encuentran consecutivos en el dds, por lo que la única dificultad estriba en calcular el tamaño de cada mipmap. **Recordad que el formato DXT1 usa solo 4 bits por pixel**.

Las texturas en formatos comprimidos tipo DXT1 funcionan por tiles (en este caso de 4x4). Por ello los últimos mipmaps de 2x2 y 1x1 producen un error al subirlos a VRAM. Para evitar el problema, los últimos mipmaps deben tener calcularse con tamaño mínimo de 8 bytes (4x4 pixels de 4 bpp).