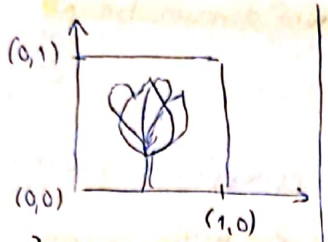
Notes on "Textures" (CG)

- Define a mapping between vertices and coordinates in the texture.

NOTE: For each vertex, texture coordinates must be defined BEFORE vertex coordinates.

We have this texture:

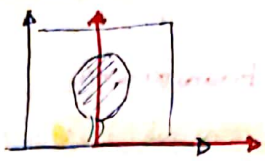


and the code:

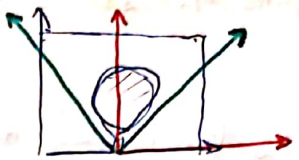
```
glMatrixMode(GL_TEXTURE);
glTranslatef(0.5, 0.0, 0.0);
glRotatef(45, 0, 0, 1);

glMatrixMode(GL_MODELVIEW);
glBegin(GL_QUADS);
   ...
glEnd();
```

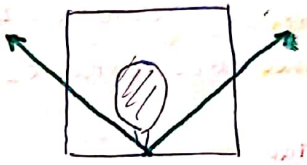
- translate



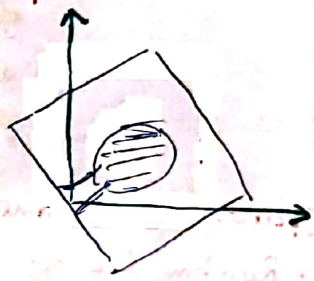
- Rotate



- clear the diagram



- put it straight



- Flickering and Aliasing.
  - Vertices have texture coordinates specified by the application
  - Pixels have texture coordinates interpolated based on distance to vertices.

- Mipmapping: Create multiple textures at different scales, as in a pyramid.

- Filter options for GL_MIN_FILTER:

GL_NEAREST_MIPMAP_NEAREST
GL_LINEAR_MIPMAP_NEAREST
GL_NEAREST_MIPMAP_LINEAR
GL_LINEAR_MIPMA_LINEAR

↑ Que pixels de textura — ↑ Que texturas) usar

(MIPMAPPING)

> ADVANTAGES
- Better quality
- Potentially faster due to cache use

> DISADVANTAGES
- Memory required for mipmap levels (+-
- Initial setup

- TRANSPARENCY → Allow us to combine a color w/ what was previously written in the frame buffer

> partial transparency:

- Ordering is crucial. Opaque elements must be drawn first
- Transparent elements must be rendered based on distance to camera

or using BSP. Furthest element drawn first.

> total transparency

The Alpha channel test is an appropriate solution,

↳ is performed before the z-buffer is written and eliminates every pixel that fails the test. → Hence, these pixels do not affect the z buffer.

—//—

- The texturing functionality must be enabled.

```
glEnable (GL_TEXTURE_1D);
glEnable (GL_TEXTURE_2D);
glEnable (GL_TEXTURE_3D);
```

Ex. OpenGL - Environment Map

- Runtime (or not) cube map generation for rendered scenes:

→ Define a camera w/ a field of 90 degrees.

→ Aim the camera along the positive X axis and capture the frame for the respective cube side

→ Repeat for the remaining 5 directions

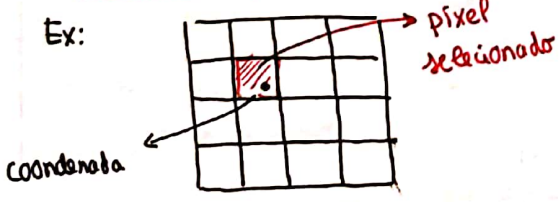
—// EXERCICES //—

④ Descreva o processo de amostragem utilizando o filtro GL_LINEAR e GL_NEAREST.

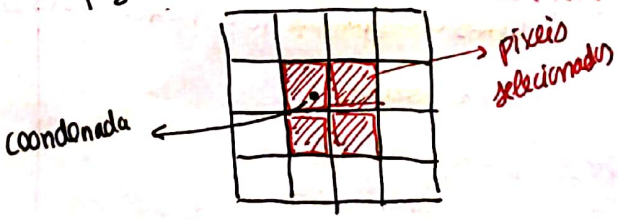
R:

- GL_NEAREST: Vamos procurar o pixel cujo centro esta mais proximo da nossa condenada

Ex:



- GL_LINEAR: Vamos buscar os 4 centros que estao mais proximos da nossa textura e fazemos uma média entre eles.



⑤ Descreva o mecanismo de Mipmapping, indicando as suas vantagens e desv.

R: O mecanismo de Mipmapping tem a ver c/ o facto de que em certas circunstâncias (cf. 22 IMAGEM A) as texturas funcionam bem numa parte mas noutra não.

NOTA: Quando vou buscar pixeis a uma imagem e a câmara mexe, passo a ir buscar pixéis a outro lado.

A ideia p/ criar o Mipmapping é que buscar poucos pixéis a texturas muito grandes "dá asneira."

cf. 24) Vamos encolhendo a imagem até um pixel. A imagem toda, e não pixeis. Poucos pixeis não seriam representativos da imagem grande mas a imagem reduzida é!

• NOTA: É mais preciso do que fazer a interpolação dos pixels escolhidos.

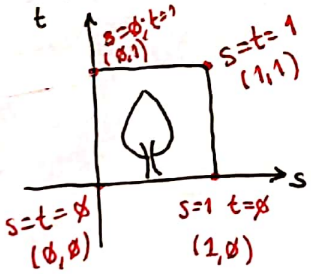
Apesar de contar todos que o mipmapping precisa, acaba por poder ter mais vantagens em termos de acesso à memória!

• O mipmapping retira bastante aquele efeito de "brilho" (cf. 22) quando se encolhe serenamente uma textura (o que acaba por acontecer sempre, por isso o mipmapping é sempre aconselhado

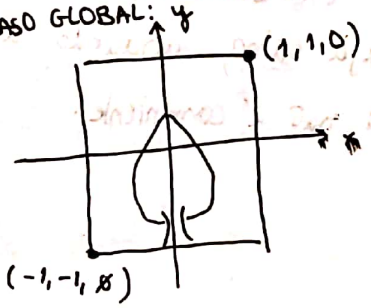
- DESVANTAGENS: pré-processamento → memória usada p/ guardar os níveis de mipmap

• O pré-processamento implica ou estar a criar no (arranque) início da aplicação aqueles níveis todos (cf. imagem p. 24) ou guardá-los previamente e carregá-los. Porém, isto em termos de memória não é muito significativo... (Isto acaba por valer a pena)

⑥ ESPAÇO TEXTURA:



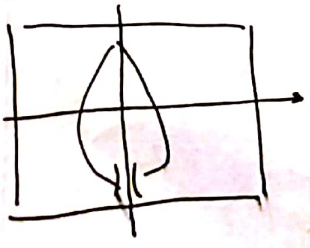
ESPAÇO GLOBAL: y



NOTA: Independentemente do tamanho da imagem, no espaço textura é sempre lado 1 x lado 1 !!!

```
glBindTexture(GL_TEXTURE_2D, 2D, texID);
glBegin(GL_QUADS);
    glTexCoord2f(0, 0); glVertex3f(-1, -1, 0)
                (1, 0);          (1, -1, 0)
                (0, 1);          (-1, 1, 0)
                (1, 1);          (1, 1, 0)
```

Esse espaço global fosse:



(1) Para obter transparências parciais é necessário ordenar os triângulos de modo a que os △s transparentes sejam desenhados só no final, ordenados por distância à câmara. PORQUÊ?

R: Esta necessidade vem do facto de que se um objeto transparente estiver à frente de um não transparente, esse será visível. Se não pararmos que o de trás é desenhado 1º, ele não passará no teste do z buffer, logo não será desenhado. Se desenharmos de trás p/ a frente o prob. desaparece.

(2) Explique o mecanismo de transparência total utilizando o teste do canal alfa.

R: O mecanismo de transparência total utiliza uma componente p/ a transparência de um pixel. O teste do canal alfa testa se um pixel será desenhado ou não segundo a sua transparência, caso o pixel não passe no teste, não é escrito no z-buffer e consequentemente não é desenhado.

NOTA: cf. 39

```
glEnable(GL_ALPHA_TEST);
glAlphaFunc(GL_GREATER, Ø);
```

→ isto é o valor do alpha
Se alpha = 1, o objeto é opaco.
Portanto alpha é mais a opacidade do que a transparência.

(3) Qual o problema de amostragem resultante de projetar uma textura no ecrã numa área cl um nº de pixéis mt inferior à dimensão da textura.

R: Como essa área tem um nº de pixéis mt inferior à da textura, uma pequena alteração (na câmara por exemplo) pode fazer cl que o pixel correspondente seja outro, andando a saltar de um p/ o outro de forma brusca, o que não é conveniente.