Computação Gráfica

LEI / LCC
Departamento de Informática
Universidade do Minho

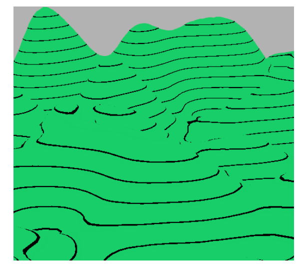
Texturing

Texturing: Definition and Application

Texturing

- Map 1D, 2D or 3D images to geometric primitives
- Applications:
  - Simulate materials: wood, granite, bricks
  - Replace complex geometry
  - Simulating natural phenomena (reflection, refraction, lens flares, etc...)

Textures



- 1D
  - *A* pixel line

- 2D
  - Regular image

- 3D
  - Volumes, as if the object was sculpted from a material

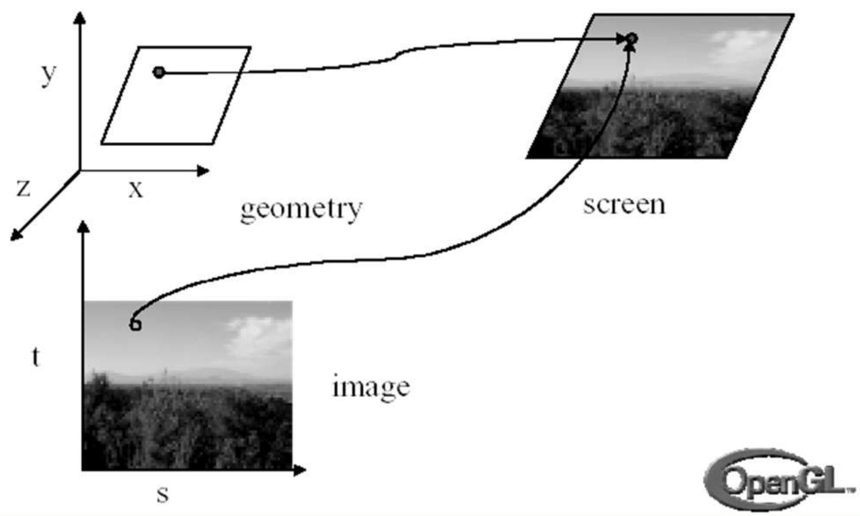
DEMO I – TEXTURE APPLICATION EXAMPLES

Textures - Usage

- Definition
  - Load an image
  - Create a texture in OpenGL
  - Define texture parameters
- Application
  - Define geometric transformation for texture (if applicable)
  - Define texture coordinates

Textures - Application

- Textures have their own coordinate system (s, t and r axes)
- Define a mapping between the vertices and coordinates in the texture.



Textures - Aplication

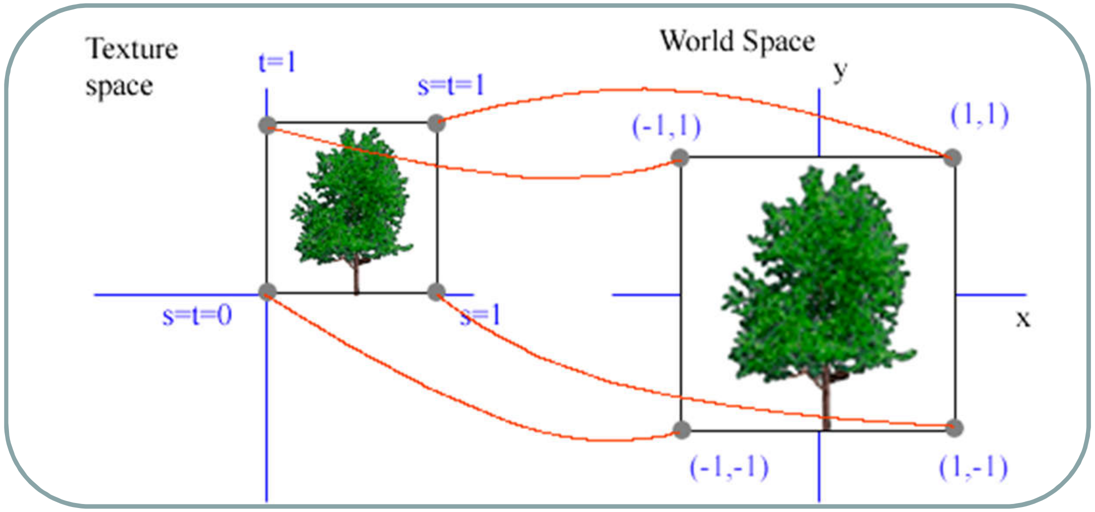
- When defining vertex coordinates, specify also the texture coordinates.

```
glBindTexture(GL_TEXTURE_2D,texID);
glBegin(GL_QUADS);
     glTexCoord2f(0,0);glVertex3f(-1.0f, -1.0f, 0.0f);
     glTexCoord2f(1,0);glVertex3f( 1.0f, -1.0f, 0.0f);
     glTexCoord2f(1,1);glVertex3f( 1.0f, 1.0f,  0.0f);
     glTexCoord2f(0,1);glVertex3f(-1.0f, 1.0f,  0.0f);
glEnd();
```

Note: for each vertex, texture coordinates must be defined BEFORE vertex coordinates.

Textures - Application

```
glBindTexture(GL_TEXTURE_2D,texID);
glBegin(GL_QUADS);
    glTexCoord2f(0,0);glVertex3f(-1.0f, -1.0f, 0.0f);
    glTexCoord2f(1,0);glVertex3f( 1.0f, -1.0f, 0.0f);
    glTexCoord2f(1,1);glVertex3f( 1.0f, 1.0f,  0.0f);
    glTexCoord2f(0,1);glVertex3f(-1.0f, 1.0f,  0.0f);
glEnd();
```



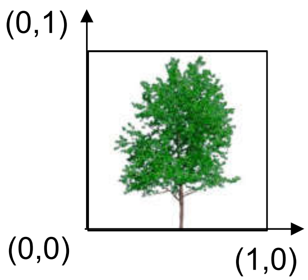
DEMO II – TEXTURE COORDINATES

Textures - Application

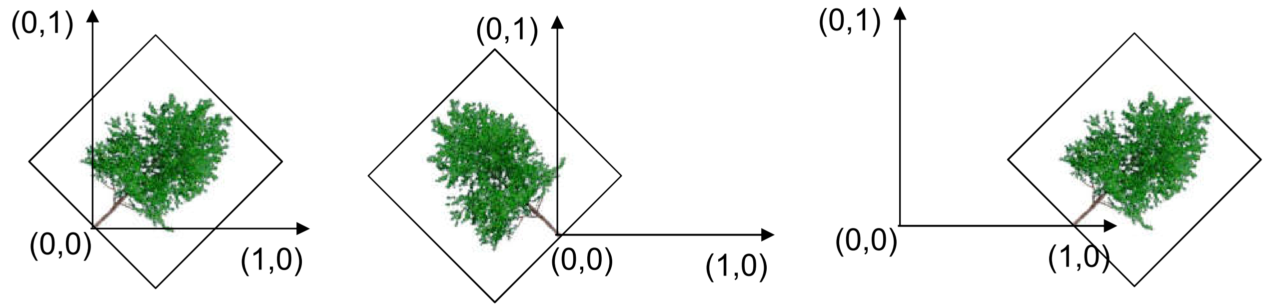
- Using VBOs
  - Setup:
    - Create an array with texture coordinates
    - Create a buffer and copy the array data to the buffer
  - Rendering
    - Bind buffer
    - Semantics
    - Draw

Textures - Application

- Consider the texture and the following code. Which result is correct?



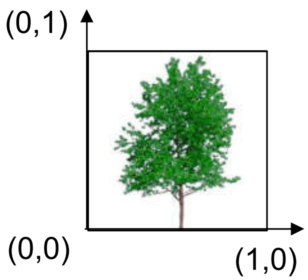

```
glMatrixMode(GL_TEXTURE);
glTranslatef(0.5,0,0);
glRotatef(45,0,0,1);

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



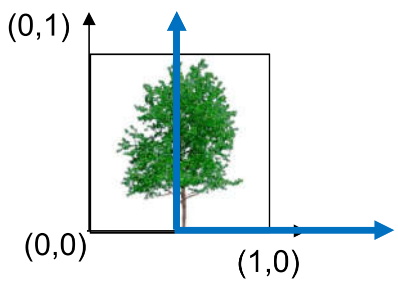
a) b) c)

Textures - Application

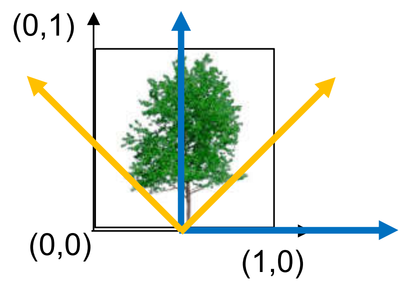
- Note that we are transforming texture coordinates.



Translate

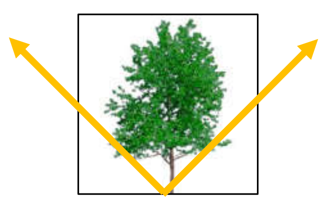


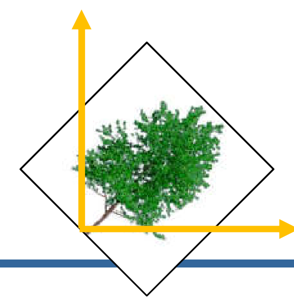
Rotate




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

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

Clear the diagram





Put it straight

DEMO III – GEOMETRIC OPERATIONS

Textures - Definition

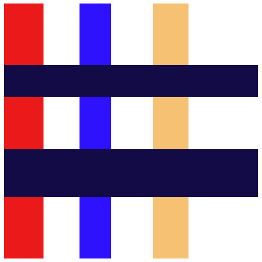
```
// Assume an image has been loaded and that w and h contain the width
// and height of the image respectively.
// Furthermore, assume that each pixel contains 4 unsigned bytes (RGBA)

int texName[1];

glGenTextures(1, texName);
glBindTexture(GL_TEXTURE_2D, texName[0]);
// wrapping parameters
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
// filtering
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h,
             0, GL_RGBA, GL_UNSIGNED_BYTE, imageData);
```

Textures - Wrap

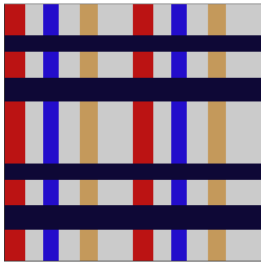
Clamp & Repeat

`GL_CLAMP`
`GL_REPEAT`

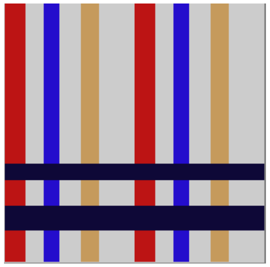
Original image



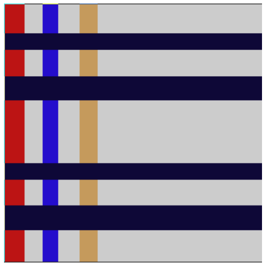
repeat both



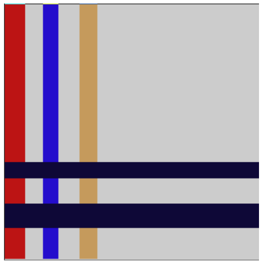
repeat s, clamp t



repeat t, clamp s



clamp both

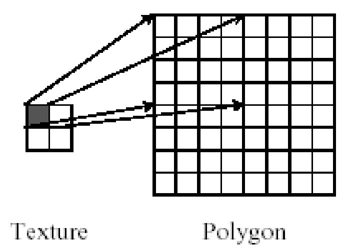


DEMO IV – TEXTURE PARAMETERS

Textures - Filters: Mag

- When the texture needs to be expanded to fit the triangles on screen

`GL_LINEAR` or `GL_NEAREST`

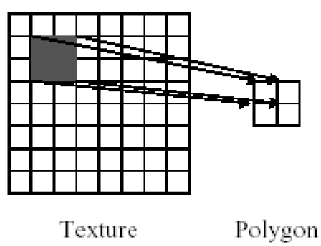


Magnification

Textures - Filters: Min

- When the texture is shrunk.

`GL_LINEAR` or `GL_NEAREST`



Minification

Textures - Filters

Mag:Nearest

May get too pixelated!

Textures - Filters

Mag: Linear

May get too blurry!

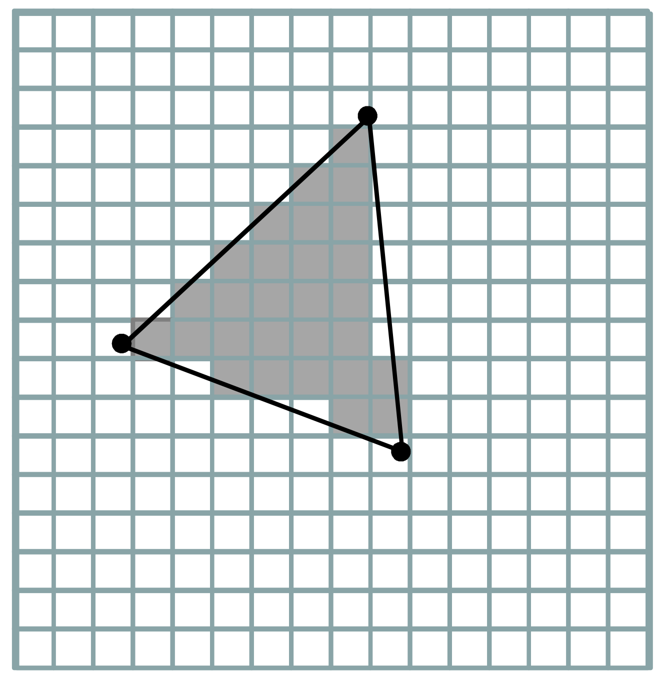
Textures – Flickering and Aliasing

Vertices (●) have texture coordinates specified by the application

Pixels (■) have texture coordinates interpolated based on distance to vertices

When the camera moves, triangle shifts in screen and pixel coordinates are updated

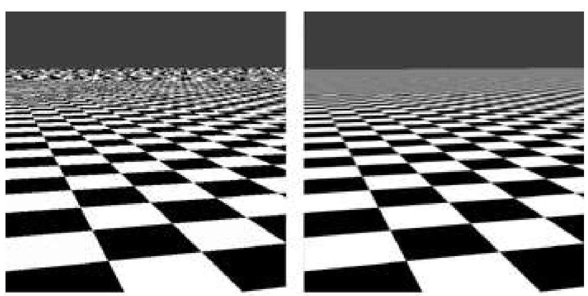
When a large image is used to cover a small portion of the screen, pixels may get totally different colors causing flickering



Textures - Mipmapping

- Issue: when the texture is severely shrunk it glitters when the camera or objects move.
- Issue: aliasing

http://www.tomshardware.com/reviews/ati,819-2.html



(a) (b)

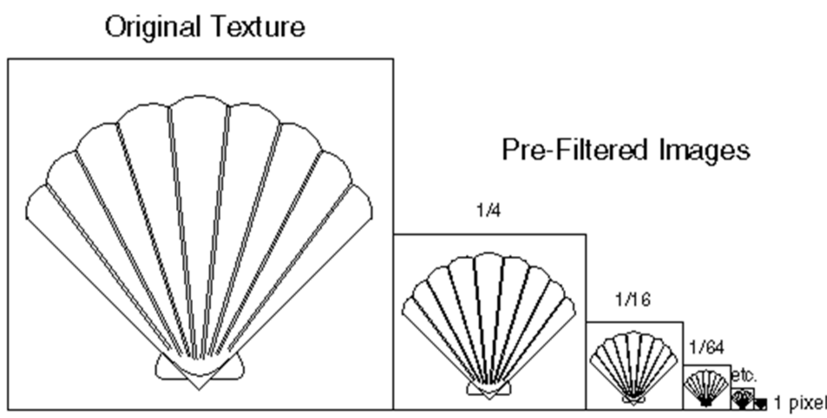
http://http.developer.nvidia.com/GPUGems/gpugems_ch25.html

DEMO V – MIPMAPPING IN USE

Textures - Mipmapping

- from Latin: "multum in parvo" (many things in a small place)
- Mipmapping: Create multiple textures at different scales, as in a pyramid.
- For instance: original texture is 32 x 16

  Provide also filtered textures: 16x8, 8x4, 4x2, 2x1, 1x1.



From the Red Book

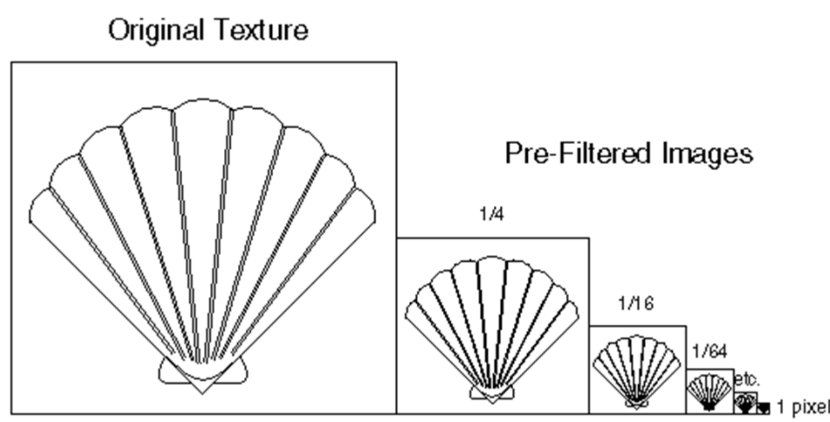
Textures - Mipmapping

- Question: How much memory is required to store all levels?

a) 2 times the original image

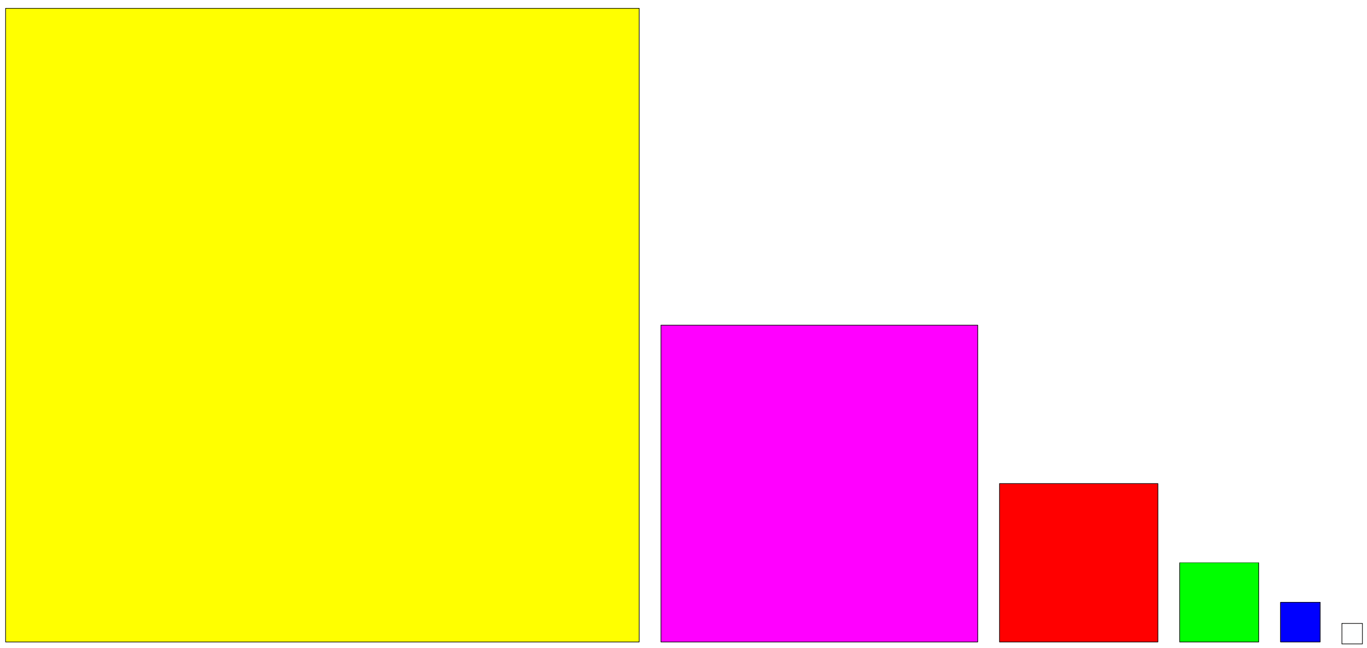
b) 1.5 x the original image

c) 1.33 x the original image



Textures - Mipmapping

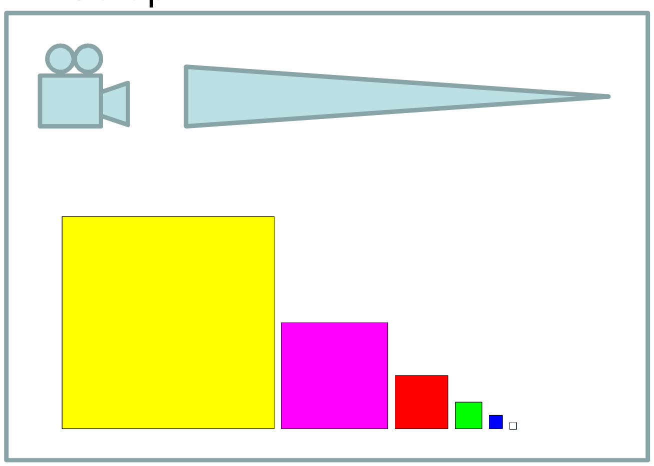
- Mipmapping: A visual example.

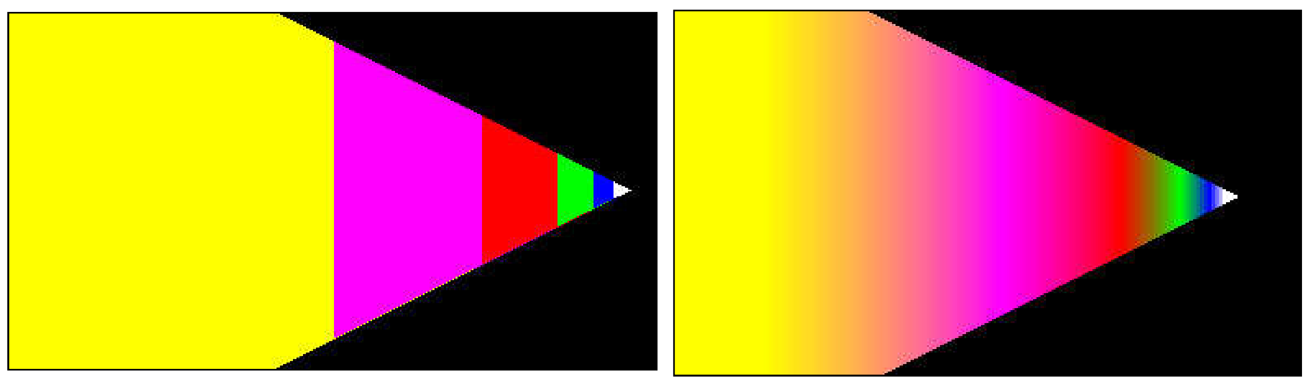


Textures - Mipmapping

- Mipmapping filtering:
  - choose more suitable level (NEAREST, on the left), or
  - A linear combination between the two more suitable levels (LINEAR, on the right)

Setup





Images taken from the Red Book

Textures - Mipmapping

4 filtering options for `GL_MIN_FILTER`:

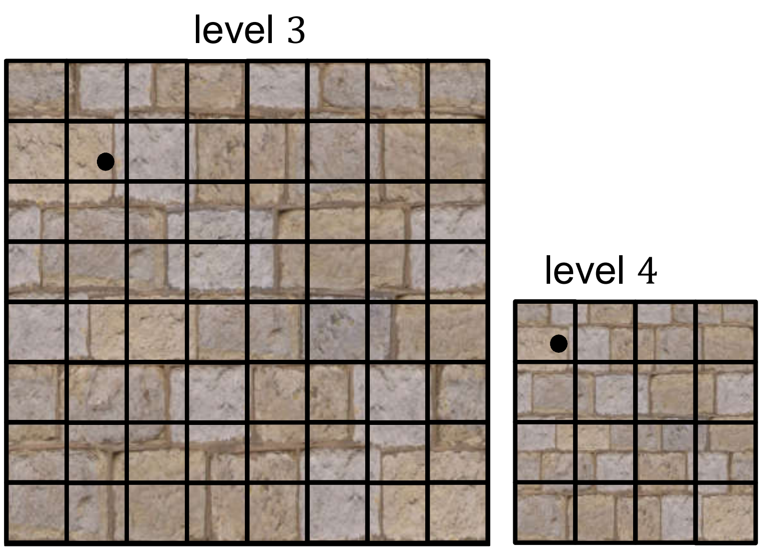
```
GL_NEAREST_MIPMAP_NEAREST
GL_LINEAR_MIPMAP_NEAREST
GL_NEAREST_MIPMAP_LINEAR
GL_LINEAR_MIPMAP_LINEAR
```

Que pixeis da textura

Que textura(s) usar

Textures - Mipmapping

Assume that:
- mipmap level is 3,25
- Texture coordinate = (0.2,0.8)

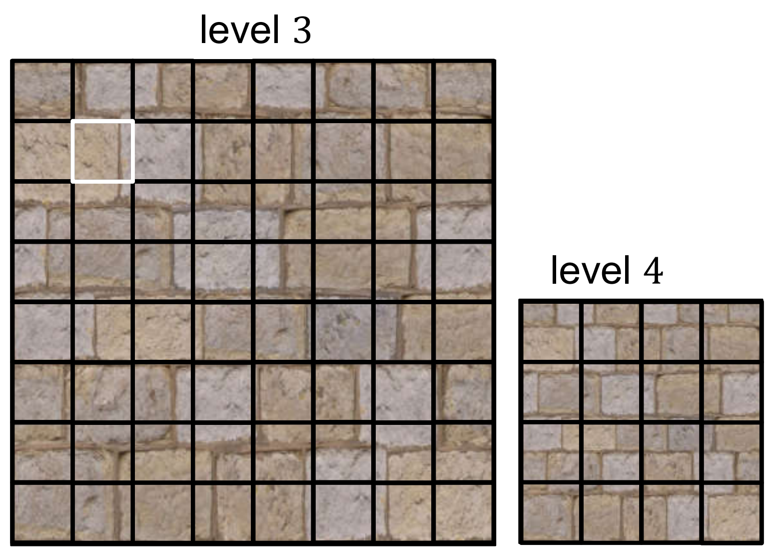


Textures - Mipmapping

Assume that:
- mipmap level is 3,25
- Texture coordinate = (0.2,0.8)

GL_NEAREST_MIPMAP_NEAREST

Pixel color = 



Textures - Mipmapping

Assume that:

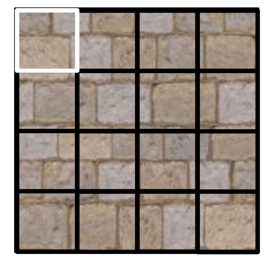
- mipmap level is 3,25
- Texture coordinate = (0.2,0.8)

GL_NEAREST_MIPMAP_LINEAR

Pixel color = 0.75 *  + 0.25 * 

level 3

level 4



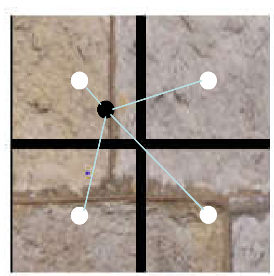
Textures - Mipmapping

Assume that:

- mipmap level is 3,25
- Texture coordinate = (0.2,0.8)

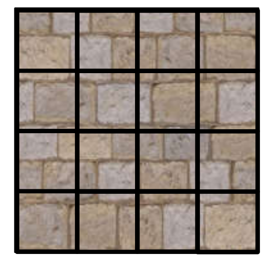
GL_LINEAR_MIPMAP_NEAREST

Pixel color = weighted average



level 3

level 4

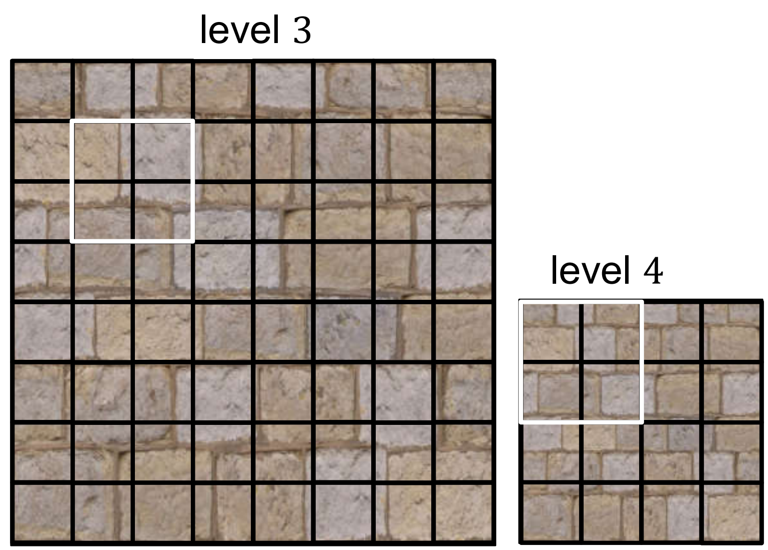


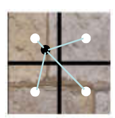
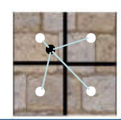
Textures - Mipmapping

Assume that:

- mipmap level is 3,25
- Texture coordinate = (0.2,0.8)

GL_LINEAR_MIPMAP_LINEAR



Pixel color = 0.75 * weighted average  + 0.25 * weighted average 

Textures - Mipmapping

- Advantages:
  - Better quality
  - Potentially faster due to cache use

- Disadvantages:
  - Memory required for mipmap levels (+- 33%)
  - Initial setup

Textures - Mipmapping

- GLU and GL (version 3.0+) allow the creation of mipmap levels.

  – With GLU

```
// instead of glTexImage2D
 gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGBA, imageWidth, imageHeight,
                   GL_RGBA, GL_UNSIGNED_BYTE, imageData);
```

  - With GL

```
// call after glTexImage2D
glGenerateMipmap(GL_TEXTURE_2D);
```

Textures - Mipmapping

```
glBindTexture(GL_TEXTURE_2D,texName);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                 GL_NEAREST_MIPMAP_NEAREST);

gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB, imageWidth, imageHeight,
                          GL_RGB, GL_UNSIGNED_BYTE, imageData);
```

Textures: Final Color

- Mixing Texture and triangle's color.

– `GL_REPLACE`	C = Ct	A = At
– `GL_MODULATE`	C = Ct * Cg	A = At * Ag
– `GL_BLEND`	C = Cg*(1-Ct) + Ce*Ct	A = Ag * At
– `GL_DECAL`	C = Cg*(1-At) + Ct*At	A = Af

$$\overbrace{\text{RGB}}^{\text{C}}\text{A}$$

g = geometry, t = texture, e = GL_TEXTURE_ENV_COLOR

```
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,param);

glTexEnvfv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_COLOR,param);
```

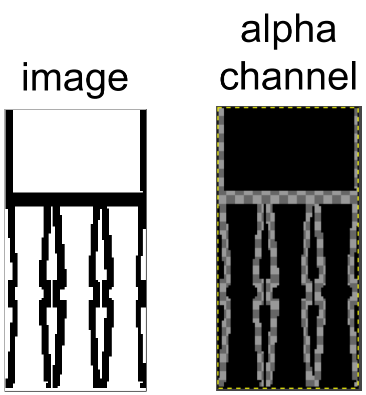
Textures: Transparency

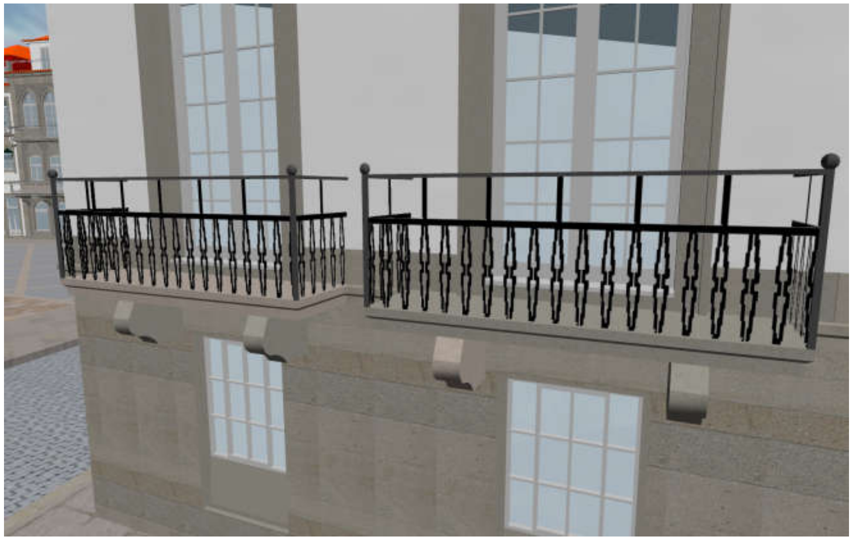
- Drawing order is **relevant** for **partial** transparencies
- For **total** transparency the **alpha channel test** is an appropriate solution.
  - The test is performed before the Z-buffer is written and eliminates every pixel which fails the test ...
  - ... Hence, these pixels do not affect the Z buffer

Textures: Transparency

- Total Transparency total in OpenGL

```
glEnable(GL_ALPHA_TEST);
glAlphaFunc(GL_GREATER, 0);
```





Textures: Transparency

- Partial transparency:
  - Transparency allows to combine a color with what was previously written in the framebuffer
  - Ordering is crucial. Opaque elements must be drawn first
  - Transparent elements must be ordered based on distance to camera or using BSP. Furthest elements drawn first
  - To compute the final color mix the two using weights for the fragment and new colors.

$$Final\ color = Cn * S + Cf * D$$

$$S = Alpha_n;\ D = 1 - Alpha_n$$

Textures: Transparency

- OpenGL

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
```

  - Alternatively:

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA,GL_ONE);
```

DEMO VI - TRANSPARENCY

Textures

- 1D
  - `glTexImage1D(GL_TEXTURE_1D,...)`

- 3D
  - `glTexImage3D(GL_TEXTURE_3D,...)`

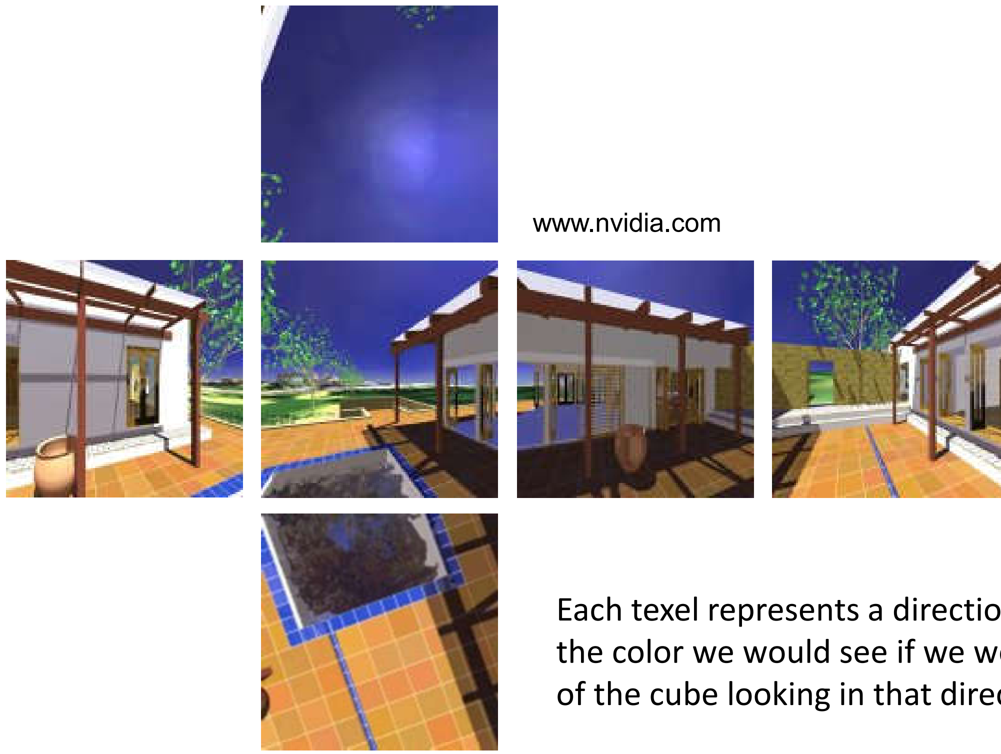
Textures

- OpenGL: The texturing functionality must be enabled.

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

OpenGL – Environment Map

Cube Mapping

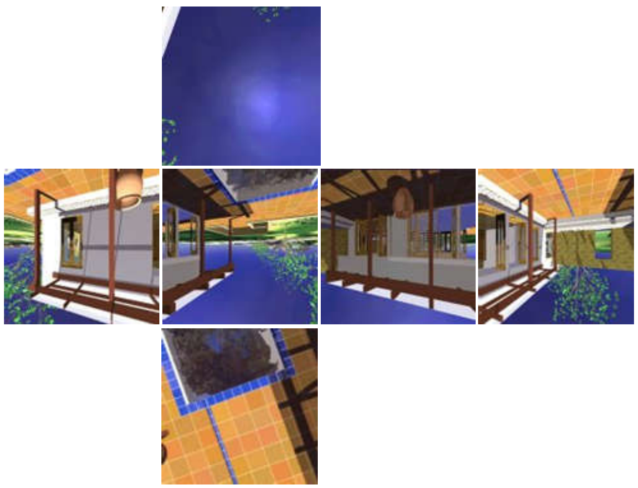


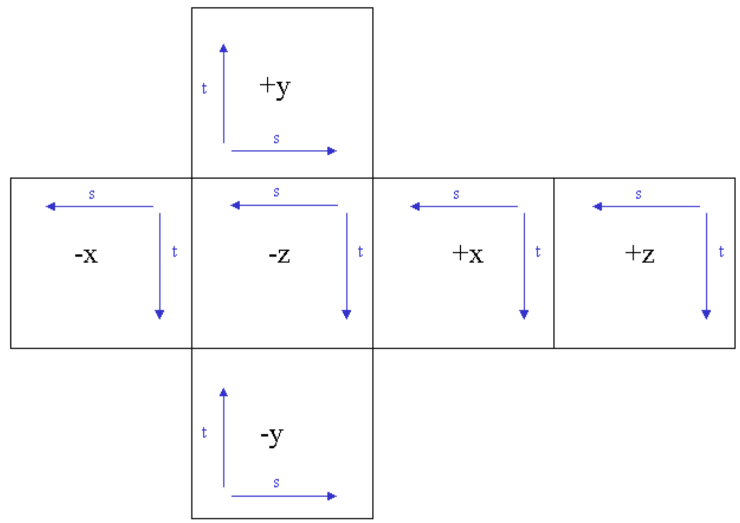
www.nvidia.com

Each texel represents a direction and its color is the color we would see if we were at the center of the cube looking in that direction

OpenGL – Environment Map

- Image Orientation

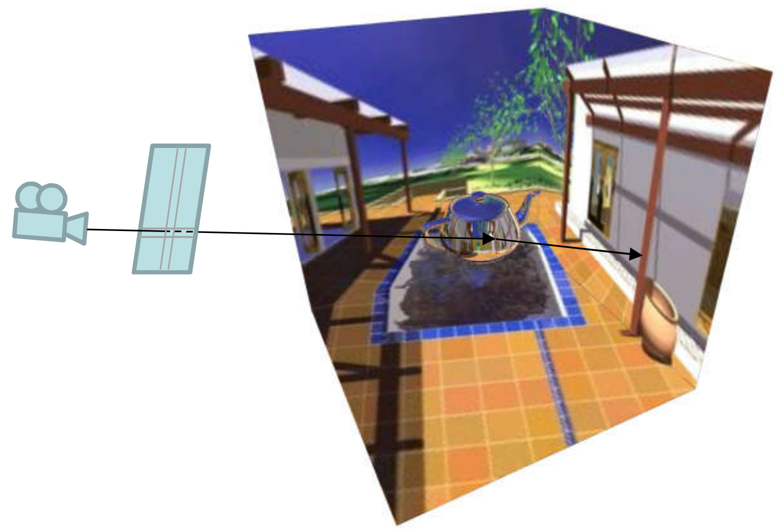




OpenGL – Environment Map

Based on the normal at the pixel of the object (teapot) a reflection vector and its intersection with the box are computed.

The texel at the point of intersection is used to shade the object.



OpenGL – Environment Map

DEMO VII – CUBE MAPPING

OpenGL – Environment Map

- Cube maps in OpenGL

```
static GLenum faceTarget[6] = {
  GL_TEXTURE_CUBE_MAP_POSITIVE_X,
  GL_TEXTURE_CUBE_MAP_NEGATIVE_X,
  GL_TEXTURE_CUBE_MAP_POSITIVE_Y,
  GL_TEXTURE_CUBE_MAP_NEGATIVE_Y,
  GL_TEXTURE_CUBE_MAP_POSITIVE_Z,
  GL_TEXTURE_CUBE_MAP_NEGATIVE_Z
};
```

```
glGenTextures(1, texName);
glBindTexture(GL_TEXTURE_CUBE_MAP,texName[0]);

for (i=0; i<6;i++) {

      glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
      glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
      glTexImage2D(faceTarget[i], 0, GL_RGB, imageWidth, imageHeight,
                0, GL_RGB, GL_UNSIGNED_BYTE, imageData);

}
```

OpenGL – Environment Map

- OpenGL: setup for reflective cube map

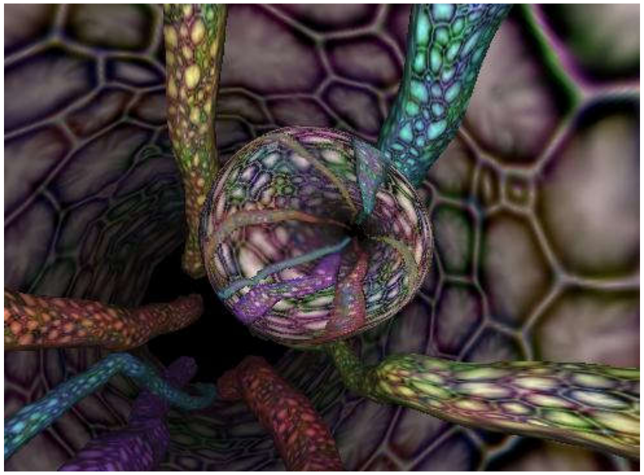
```
glEnable(GL_TEXTURE_CUBE_MAP);
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
glEnable(GL_TEXTURE_GEN_R);
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP);
glTexGeni(GL_R, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP);
```

OpenGL – Environment Map

- Runtime (or not) cube map generation for rendered scenes:
  - Define a camera with a field of view of 90 degrees.
  - Aim the camera along the positive X axis and capture the frame for the respective cube side
  - Repeate for the remaining 5 directions

OpenGL – Environment Map

- In real time



www.nvidia.com

DEMO VIII – DYNAMIC CUBEMAPPING

OpenGL – Environment Map

- "Ray Tracing"