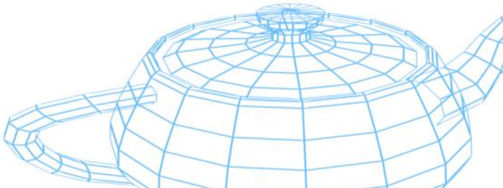
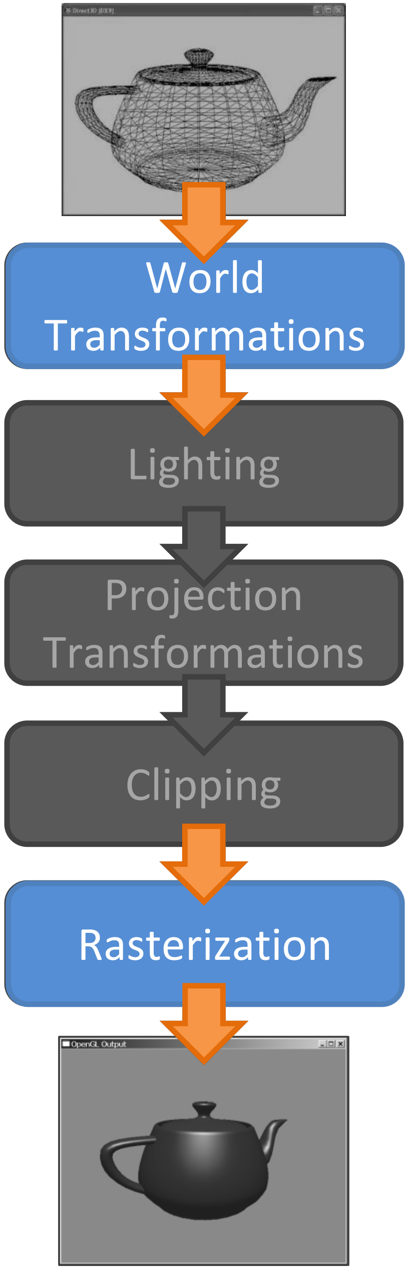
SUPSI

Computer Graphics

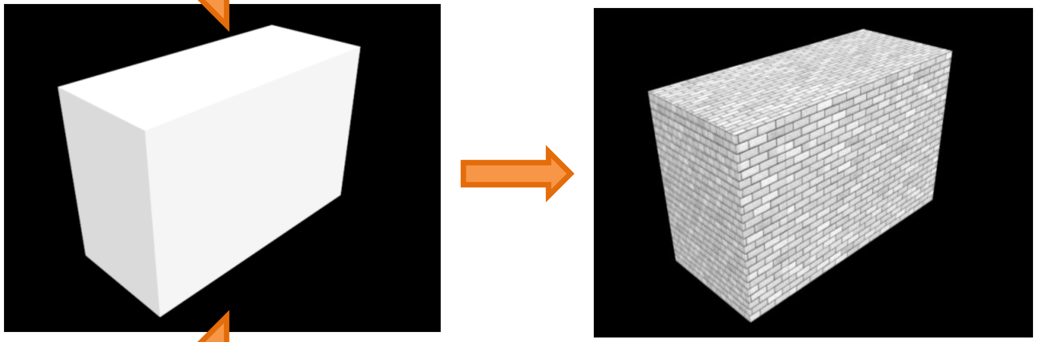
OpenGL (3): Texture mapping

Achille Peternier, lecturer





Texture coordinates



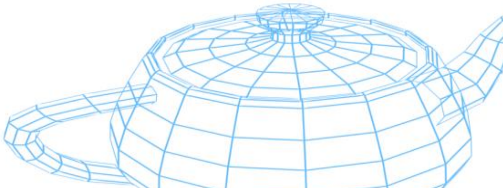
Texture interpolation
Primitive filling

Texture mapping

- Textures are images used to paint primitives during rasterization to provide additional detail without requiring additional geometry.

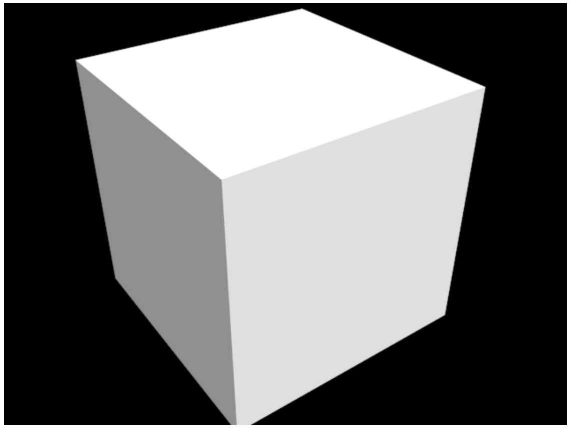
- Introduced by Edwin Catmull, Utah University, 1974 (now president of Walt Disney and Pixar animation studios).

Edwin Catmull
1945



Texture mapping

Geometry

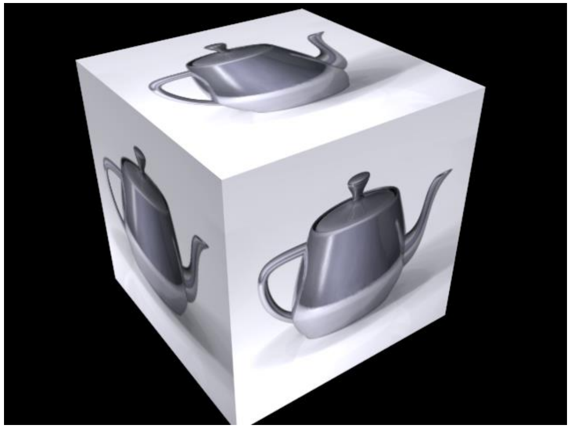


+

Texture

=

Textured geometry



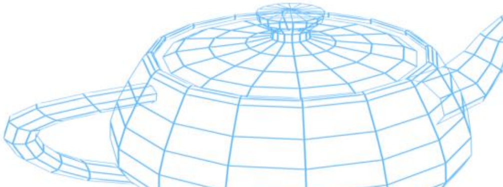
Texture mapping

- Textures are basically images:
  - Acquired through an optical device (camera, scanner, etc.).
  - Designed by 3D artists using graphic design tools.
  - Procedurally generated (fractals, noise functions, texture generators, etc.).
  - A screenshot of a previous frame or taken from a different camera position.
  - http://opengameart.org/

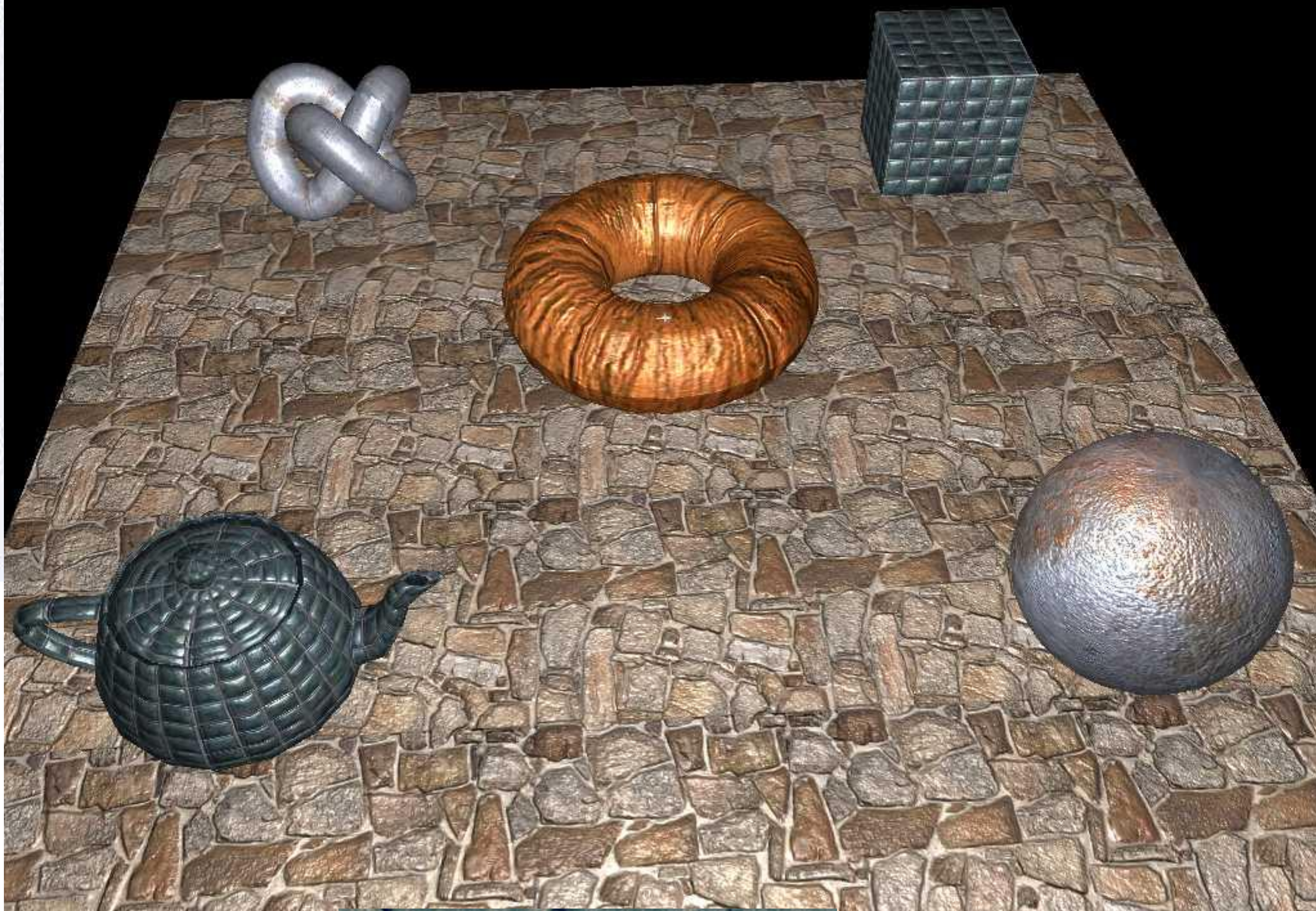
- Typically an RGB bitmap:
  - Alpha channel used for transparency or other special effects.

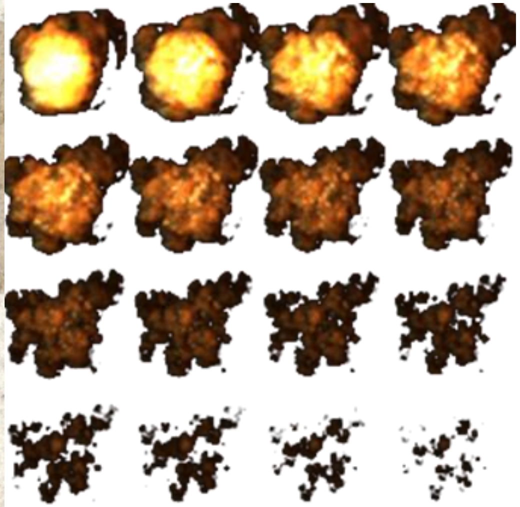
- During rasterization, each *texel* color is multiplied by the color computed by the lighting model or directly specified by the programmer:
  - You can change this default setting via **`glTexEnv*();`**

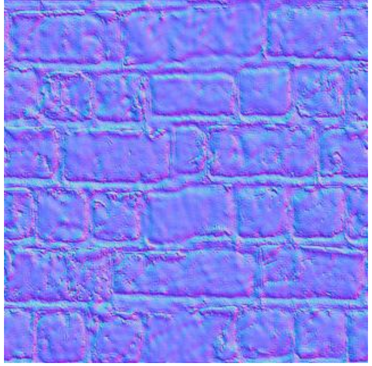
texel = TEXture ELement

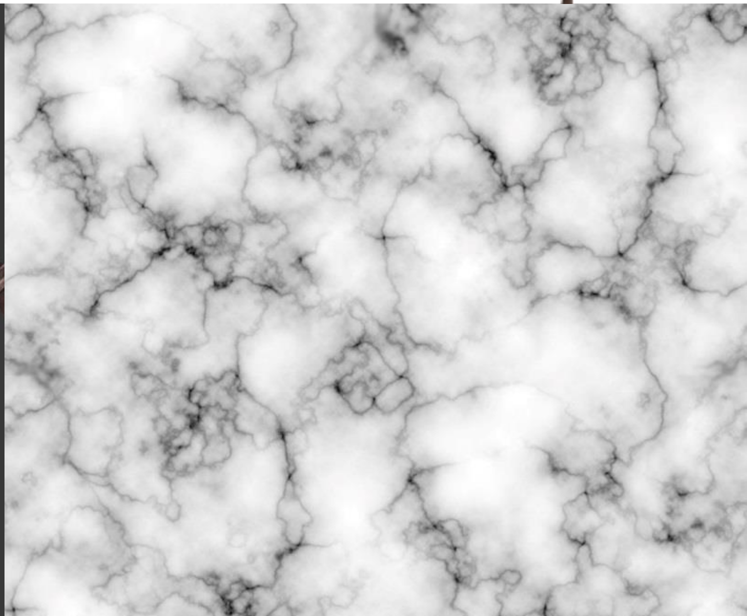


Texture examples



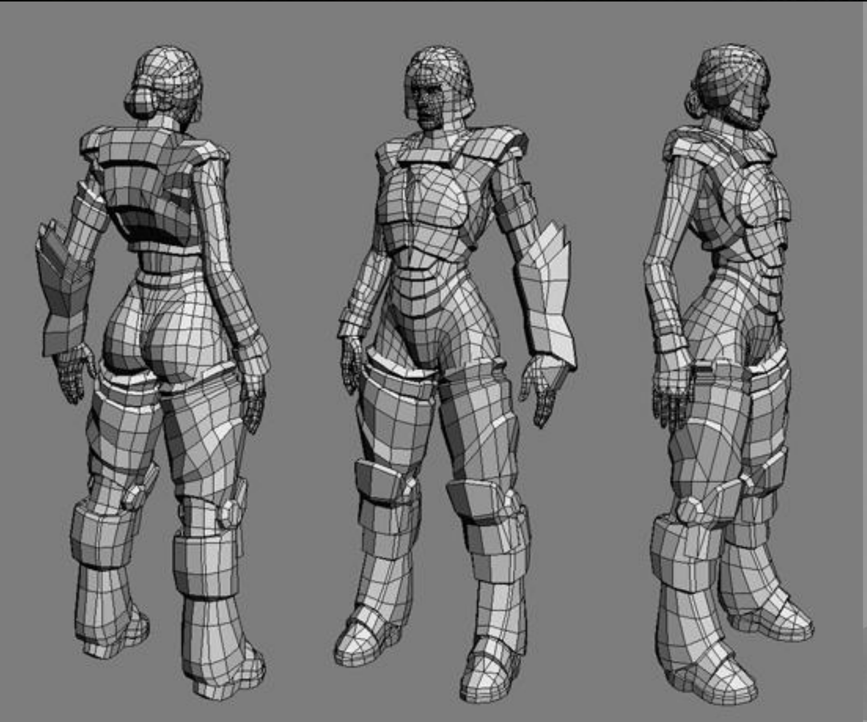


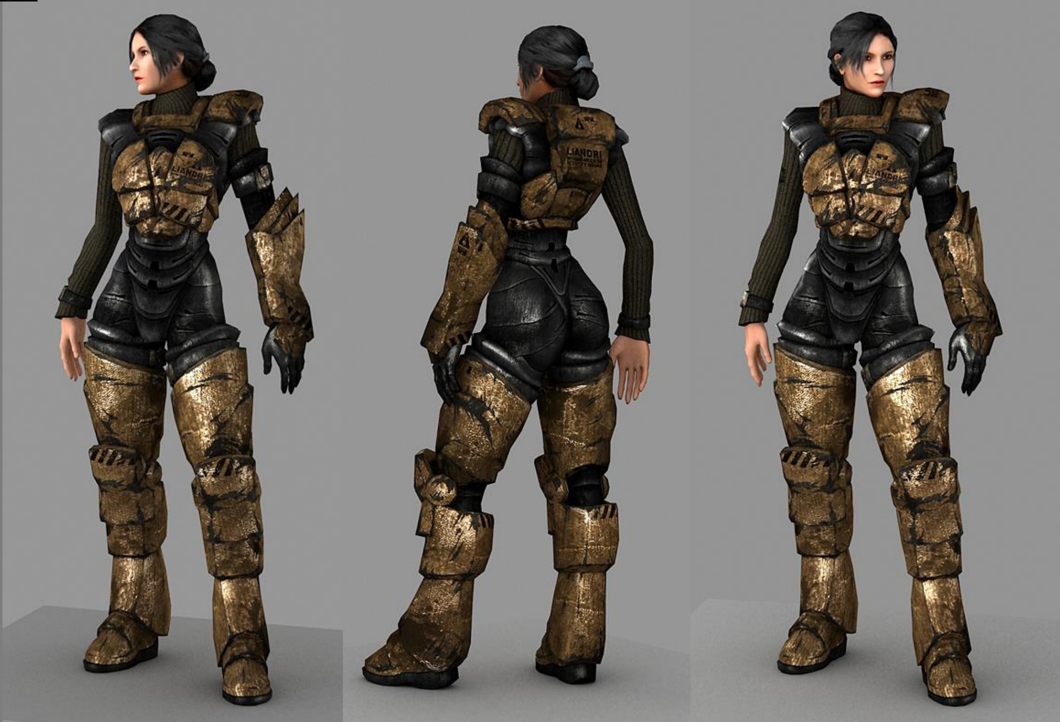




Texture examples

Textured models





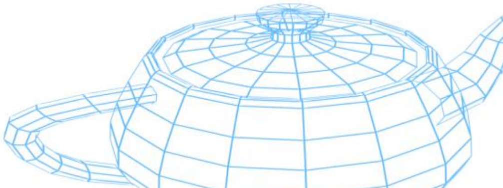
Per-vertex information

- Vertex position
  - x, y, z[, w] (usually as *float*)

- Vertex normal
  - x, y, z (usually as *float*)

- Vertex texture coordinates
  - s, t[, r] (usually as *float*)

- Vertex color (RGB or RGBA)
  - r, g, b[, a] (usually as *byte*)

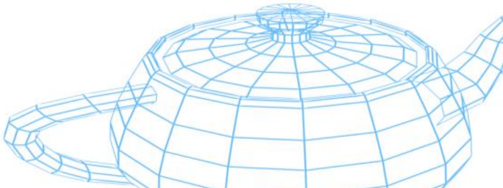


Texture bitmaps

- Texture sizes must be a power of two, e.g.: 256x512, 1024x256, 128x128, etc.

- Sizes are then normalized into the [0, 1] range:
  - …in the same way normalized device coordinates abstract from real screen sizes.

- Modern devices and recent versions of OpenGL are more relaxed about image sizes:
  - Check for the **`ARB_texture_rectangle`** extension.



Texture mapping

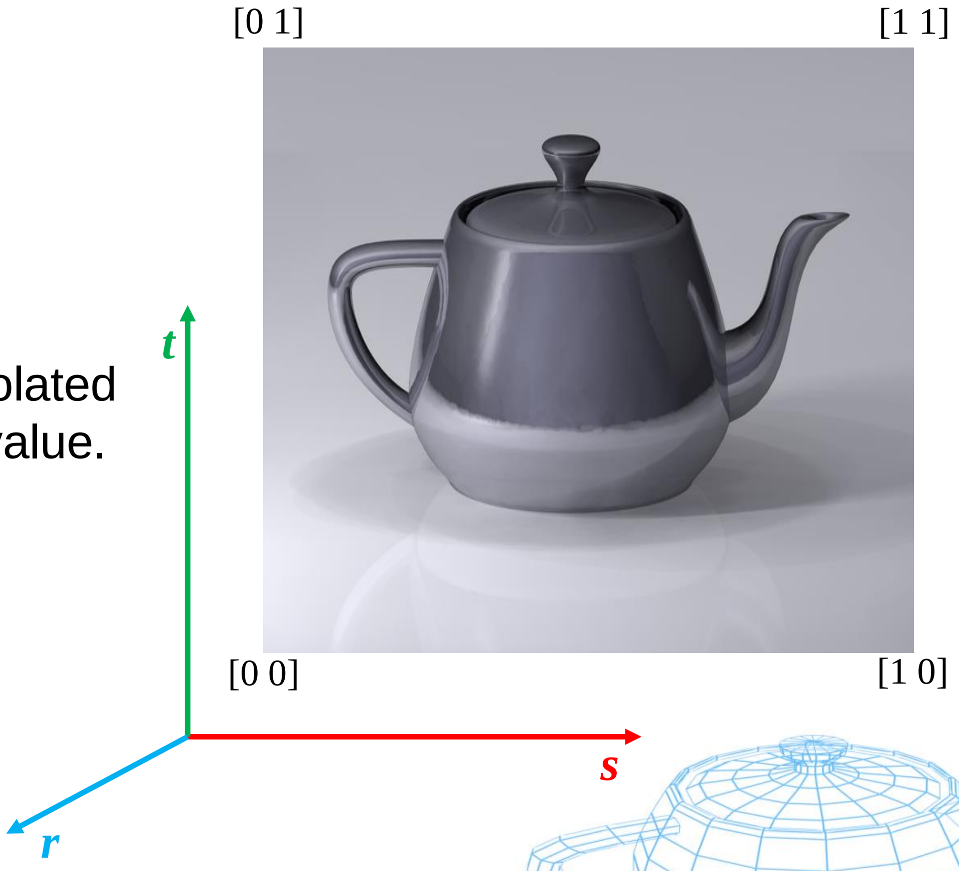
- Texture coordinates are expressed through 1, 2, and 3D coordinates defined as *s*, *t*, and *r* :

  s = u = x dimension

  t = v = y dimension

  r = w = z dimension

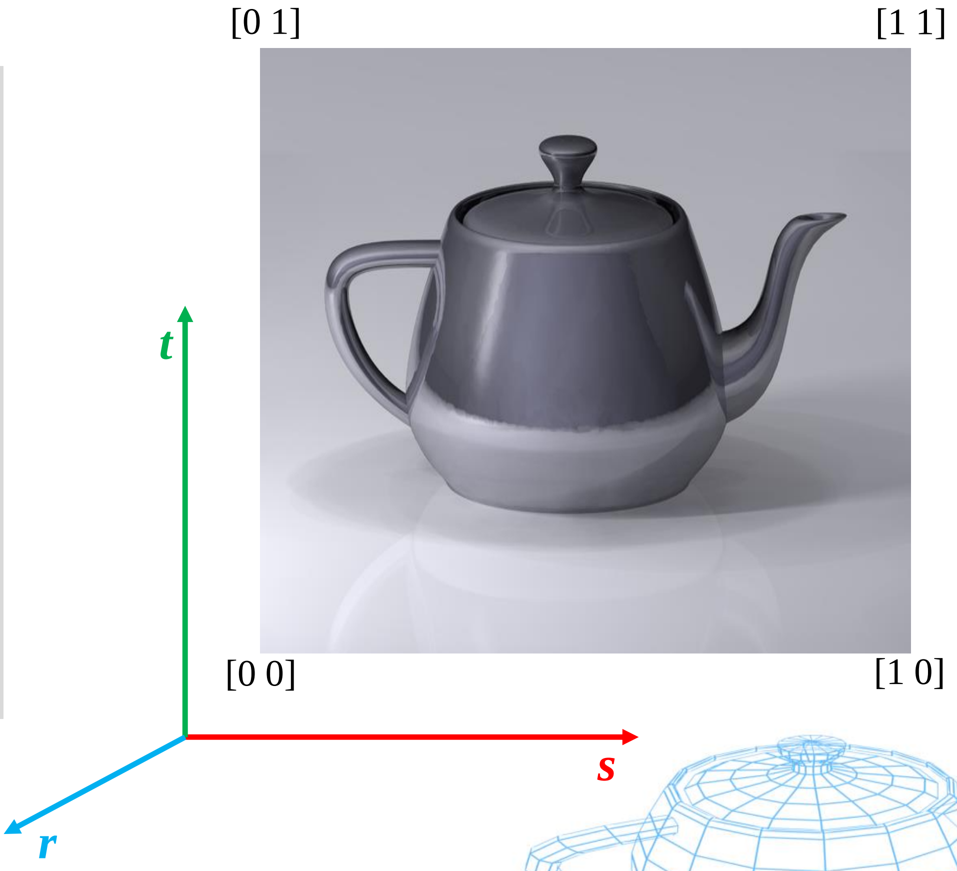
- Texture coordinates are also interpolated during rasterization, like any other value.



Texture mapping

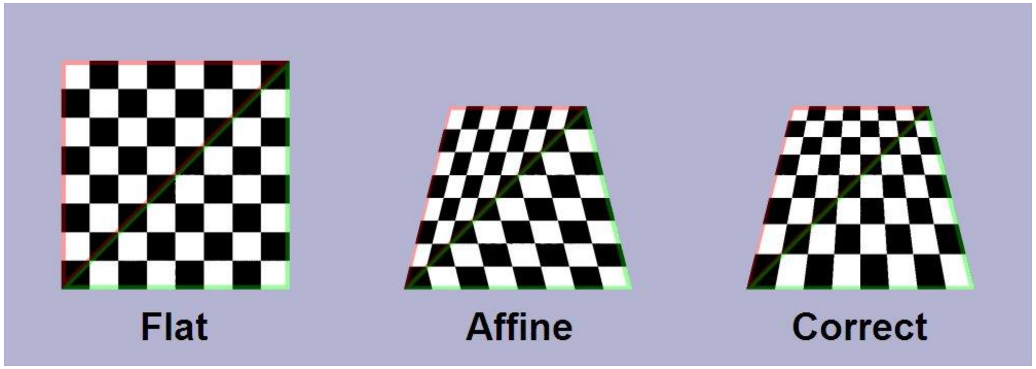
- Texture coordinates are specified per vertex through the **`glTexCoord*()`** instruction:

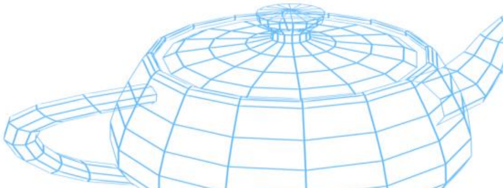
```
glBegin(GL_TRIANGLE_STRIP);
glNormal3f(0.0f, 0.0f, 1.0f);
   glTexCoord2f(0.0f, 0.0f);
   glVertex3f(size, -size, size);

   glTexCoord2f(1.0f, 0.0f);
   glVertex3f(-size, -size, size);

   glTexCoord2f(0.0f, 1.0f);
   glVertex3f(size, size, size);

   glTexCoord2f(1.0f, 1.0f);
   glVertex3f(-size, size, size);
glEnd();
```



Texture mapping

- Texture coordinates specified at each vertex are linearly interpolated across the primitive:
  - This approach is known as *affine texture mapping*.
  - Produces artifacts when the primitive is not perpendicular to the viewer.





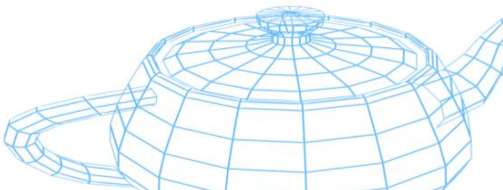
Texture mapping

- Affine coordinates at point u_p (where $0 \leq p \leq 1$) are computed as:

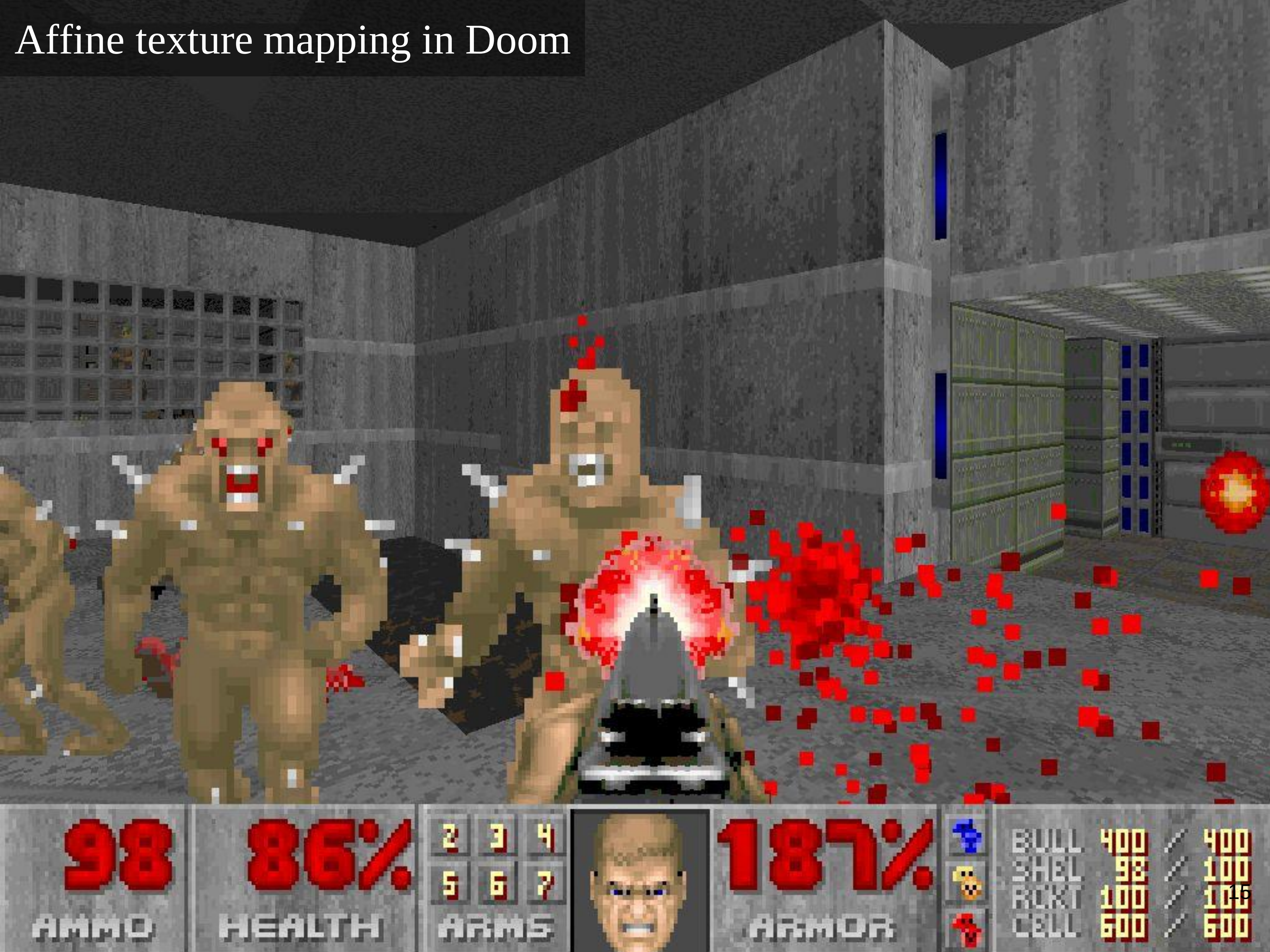
$$u_p = (1 - p)u_0 + pu_1$$

- Disabled by default, can be activated through:

```
glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_FASTEST);
```



Affine texture mapping in Doom



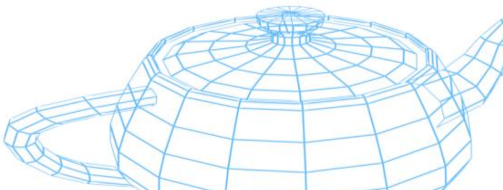
Texture mapping

- Perspective-correct texture mapping considers the 3D position of the fragment in the space:

$$u_p = \frac{(1-p)\dfrac{u_0}{z_0} + p\dfrac{u_1}{z_1}}{(1-p)\dfrac{1}{z_0} + p\dfrac{1}{z_1}}$$

- Slower than the affine technique but produces better results:
  - All modern devices support perspective-correct texture mapping in hardware.
  - Default setting in OpenGL:

```
glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
```



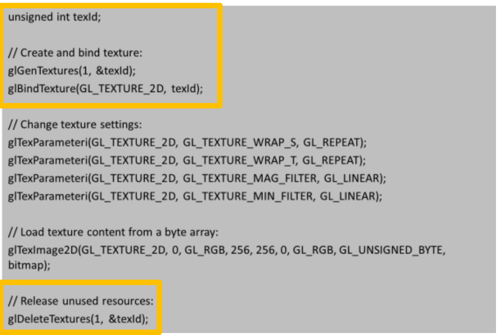
Texture mapping

```
unsigned int texId;

// Create and bind texture:
glGenTextures(1, &texId);
glBindTexture(GL_TEXTURE_2D, texId);

// Change texture settings:
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_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

// Load texture content from a byte array:
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 256, 256, 0, GL_RGB,
             GL_UNSIGNED_BYTE, bitmap);

// Release unused resources:
glDeleteTextures(1, &texId);
```

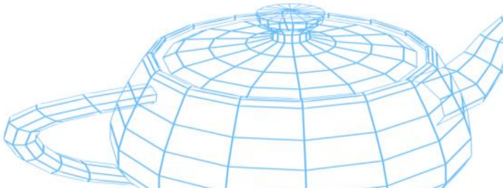
Texture mapping

- Each texture object generated by OpenGL stores a series of specific settings:
  - With a single call you can generate one or more texture objects:
    - **`glGenTextures(nrOfTextures, ptrToTexArray);`**
  - Delete them when no longer required:
    - **`glDeleteTextures(nrOfTextures, ptrToTexArray);`**

- Texture mapping and settings are applied to the current texture:
  - Use **`glBindTexture(texId)`** to set one texture as current.



```
unsigned int texId;

// Create and bind texture:
glGenTextures(1, &texId);
glBindTexture(GL_TEXTURE_2D, texId);

// Change texture settings:
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_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

// Load texture content from a byte array:
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 256, 256, 0, GL_RGB, GL_UNSIGNED_BYTE,
bitmap);

// Release unused resources:
glDeleteTextures(1, &texId);
```

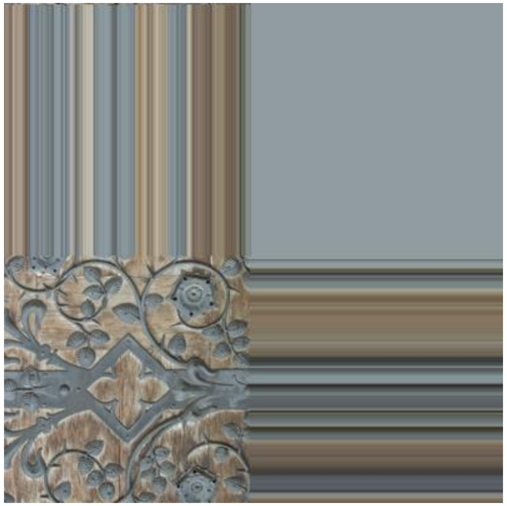


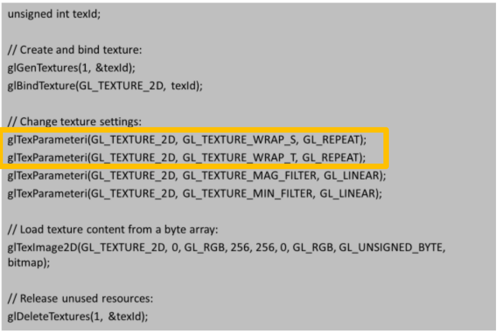
Texture mapping

- When texture coordinates are not in the range [0, 1], you can instruct how OpenGL will react. The most used options are:
  - Lower/higher values are clamped to 0 or 1.
  - Coordinates become circular in order to tile the texture multiple times.

- Parameters are set per texture and per dimension:

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



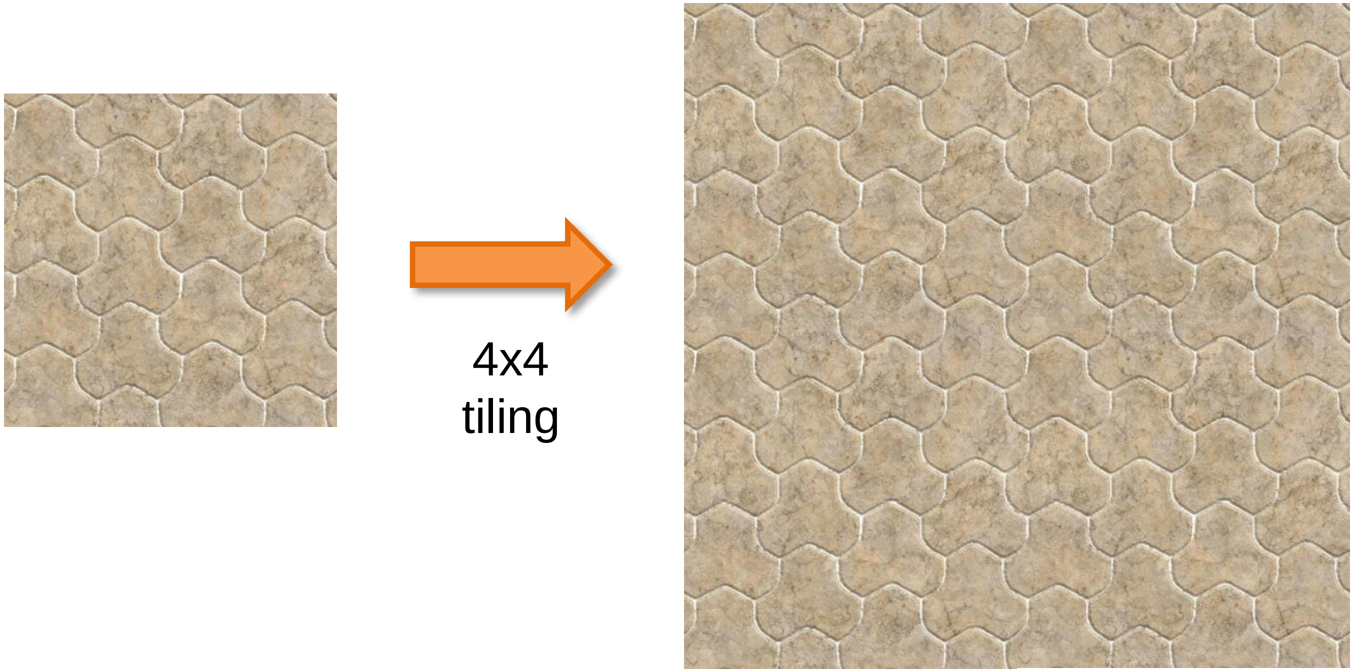


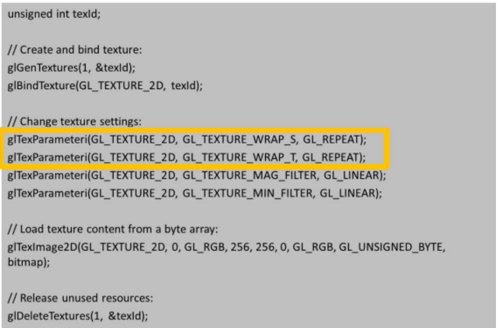
```
unsigned int texId;

// Create and bind texture:
glGenTextures(1, &texId);
glBindTexture(GL_TEXTURE_2D, texId);

// Change texture settings:
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_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

// Load texture content from a byte array:
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 256, 256, 0, GL_RGB, GL_UNSIGNED_BYTE,
bitmap);

// Release unused resources:
glDeleteTextures(1, &texId);
```

Tileable textures

- When wrapping is set to “repeat”, texture coordinates not within the [0, 1] range are used to repeat the same image.

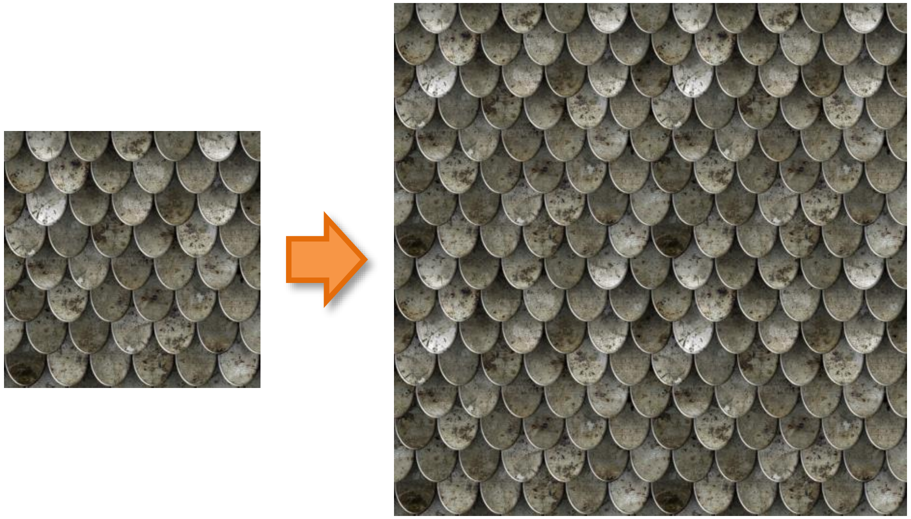
- Tileable textures are seamless images that can be put one next to the other without glitches:

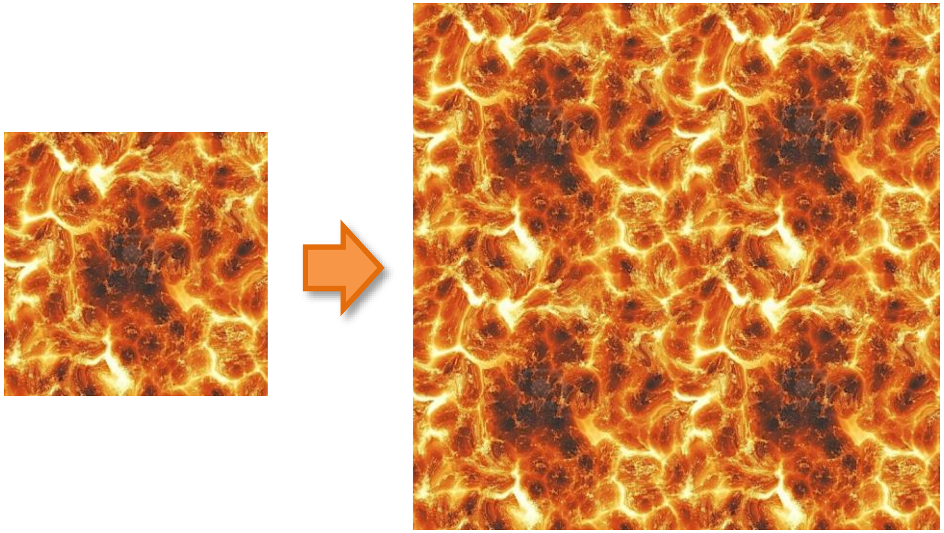




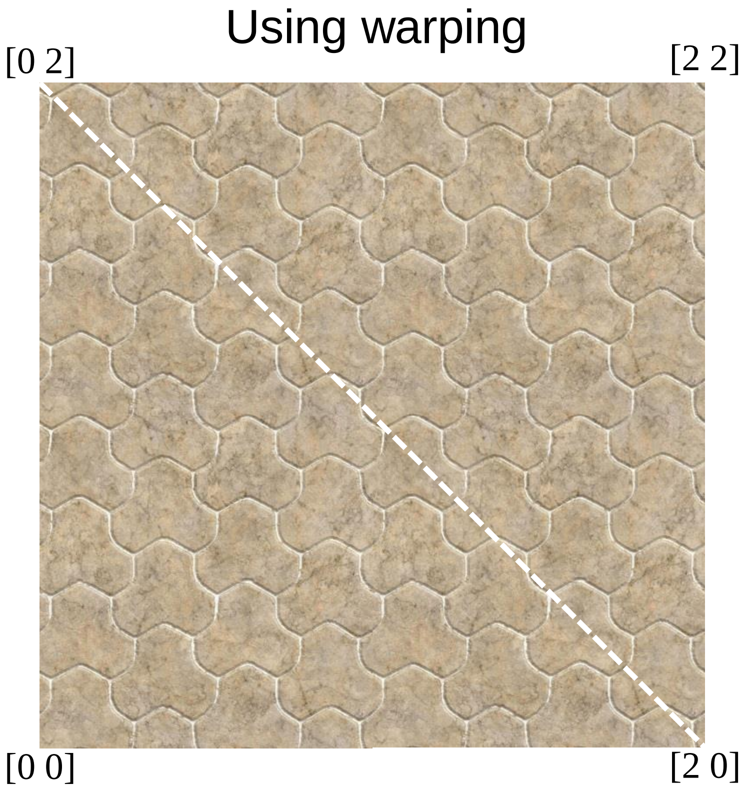
```
unsigned int texId;

// Create and bind texture:
glGenTextures(1, &texId);
glBindTexture(GL_TEXTURE_2D, texId);

// Change texture settings:
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_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

// Load texture content from a byte array:
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 256, 256, 0, GL_RGB, GL_UNSIGNED_BYTE,
bitmap);

// Release unused resources:
glDeleteTextures(1, &texId);
```

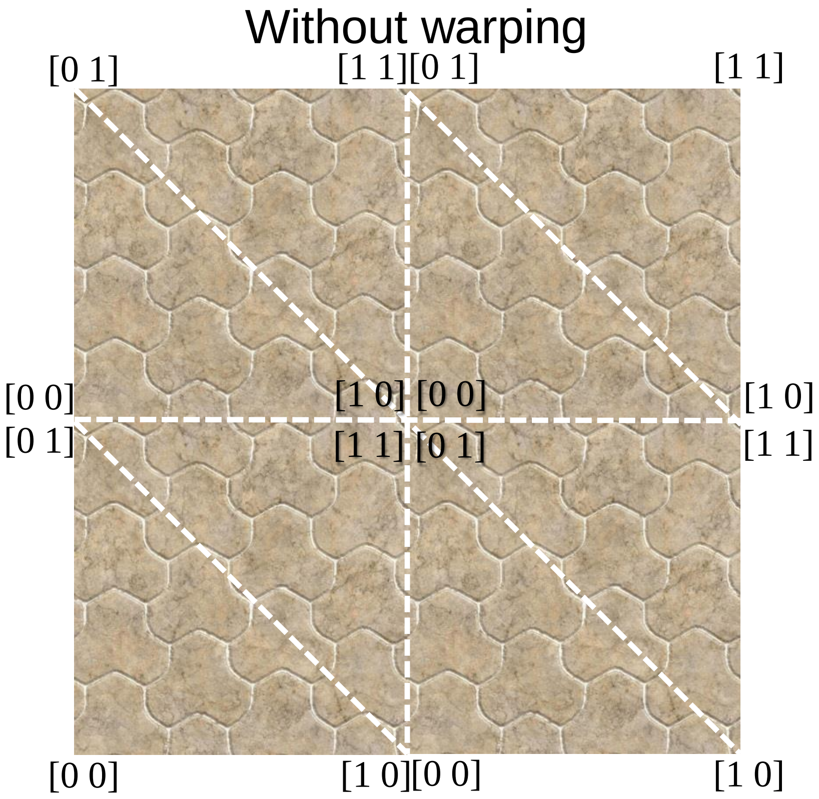
Tileable textures

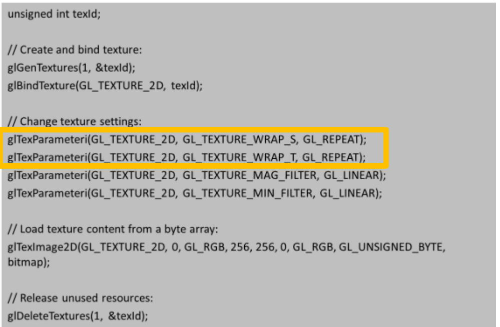




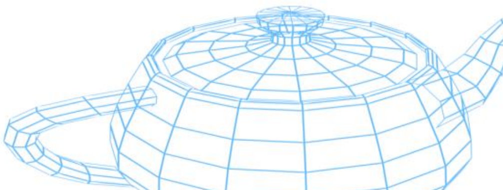
Tileable textures





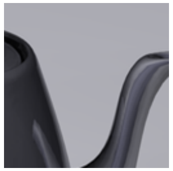


```
unsigned int texId;

// Create and bind texture:
glGenTextures(1, &texId);
glBindTexture(GL_TEXTURE_2D, texId);

// Change texture settings:
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_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

// Load texture content from a byte array:
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 256, 256, 0, GL_RGB, GL_UNSIGNED_BYTE,
bitmap);

// Release unused resources:
glDeleteTextures(1, &texId);
```

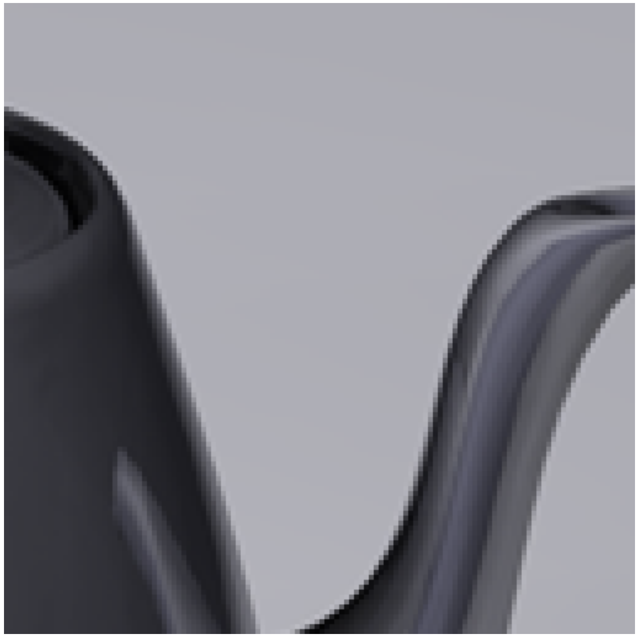


Texture filtering

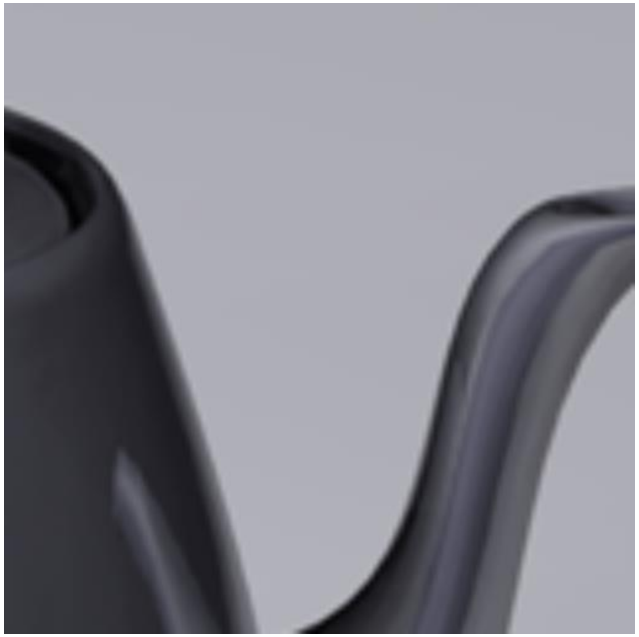
- Since textures are based on raster images, they have a finite resolution:
  - Zooming in (magnification) causes aliasing.



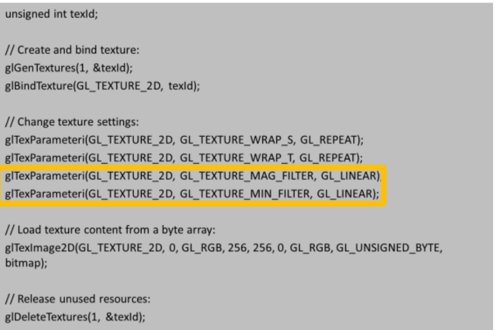
original image



no filtering (**GL_NEAREST**)



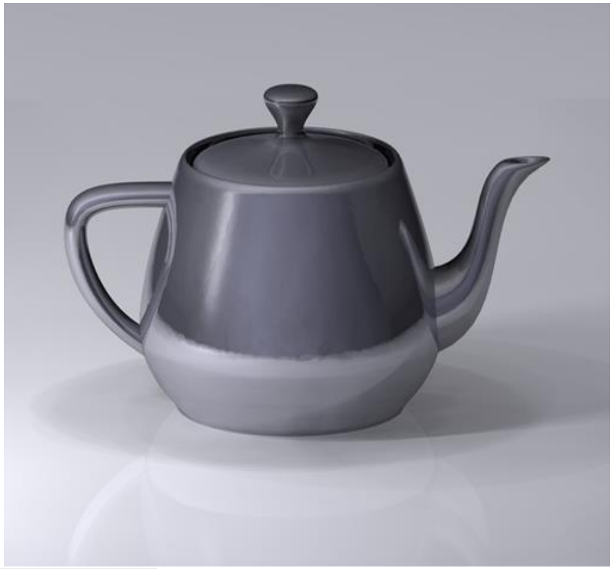
linear filtering (**GL_LINEAR**)



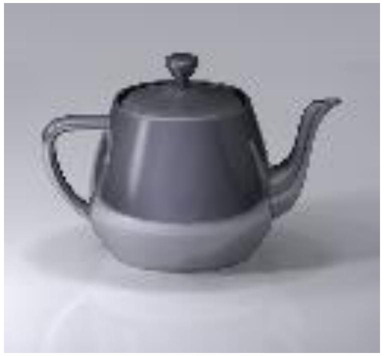
```
unsigned int texId;

// Create and bind texture:
glGenTextures(1, &texId);
glBindTexture(GL_TEXTURE_2D, texId);

// Change texture settings:
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_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

// Load texture content from a byte array:
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 256, 256, 0, GL_RGB, GL_UNSIGNED_BYTE,
bitmap);

// Release unused resources:
glDeleteTextures(1, &texId);
```

Texture filtering

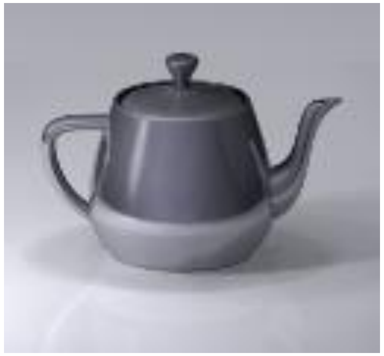
- Since textures are based on raster images, they have a finite resolution:
  - Zooming out (minimization) causes jittering.



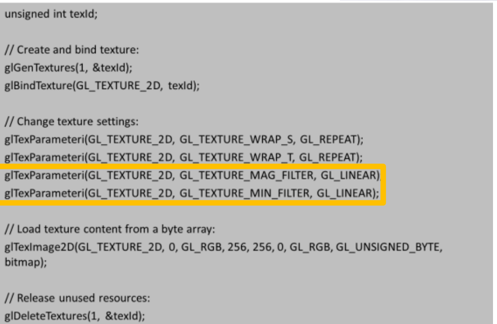
original image



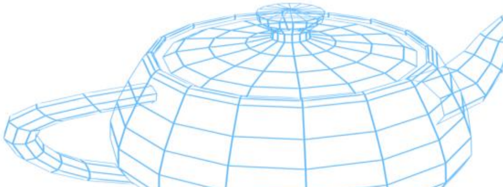
no filtering (**GL_NEAREST**)



linear filtering (**GL_LINEAR**)



```
unsigned int texId;

// Create and bind texture:
glGenTextures(1, &texId);
glBindTexture(GL_TEXTURE_2D, texId);

// Change texture settings:
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_LINEAR)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

// Load texture content from a byte array:
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 256, 256, 0, GL_RGB, GL_UNSIGNED_BYTE,
bitmap);

// Release unused resources:
glDeleteTextures(1, &texId);
```

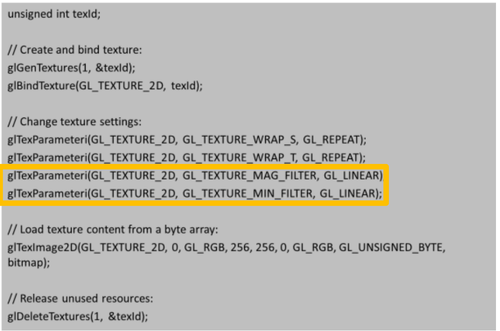


Texture filtering

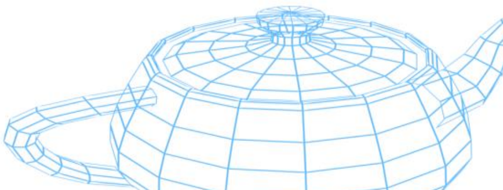
- Filtering requires additional computational power but is done by OpenGL, using the available hardware acceleration.

- Filtering is enabled through:

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
                GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                GL_LINEAR);
```

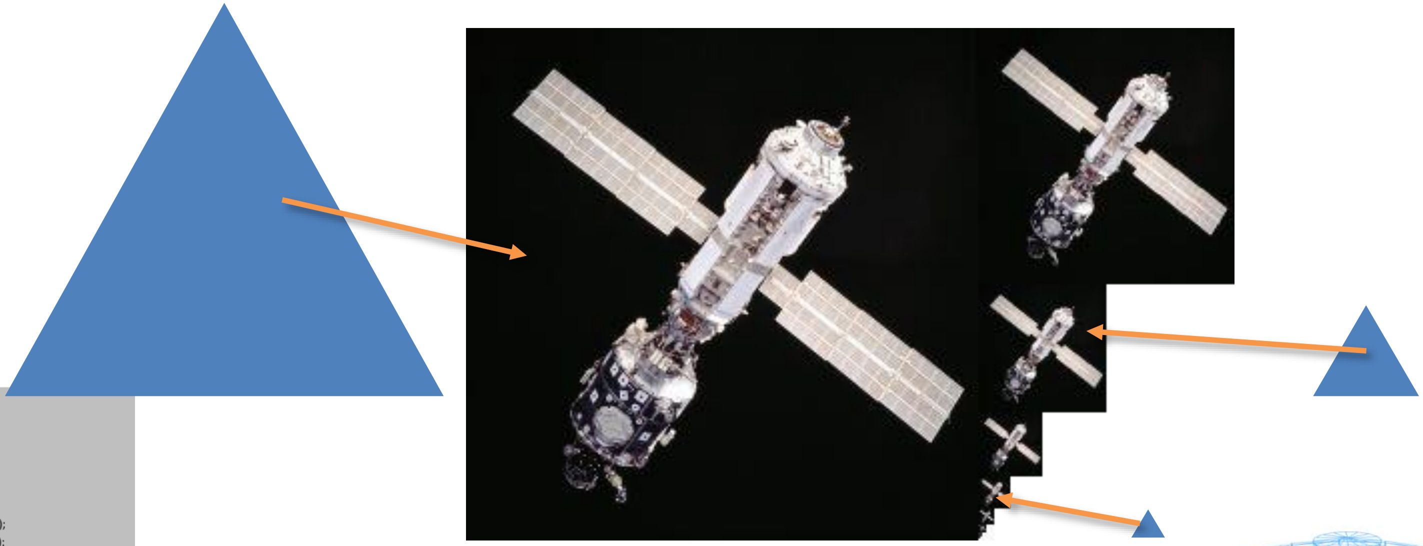


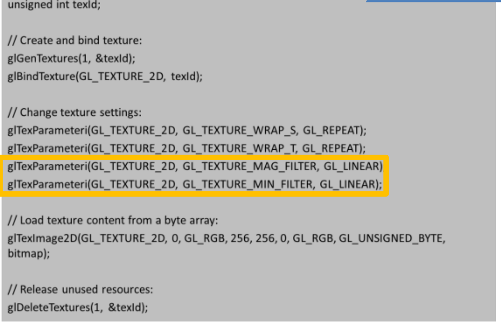
```
unsigned int texId;

// Create and bind texture:
glGenTextures(1, &texId);
glBindTexture(GL_TEXTURE_2D, texId);

// Change texture settings:
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_LINEAR)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

// Load texture content from a byte array:
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 256, 256, 0, GL_RGB, GL_UNSIGNED_BYTE,
bitmap);

// Release unused resources:
glDeleteTextures(1, &texId);
```



Texture filtering (mipmapping)

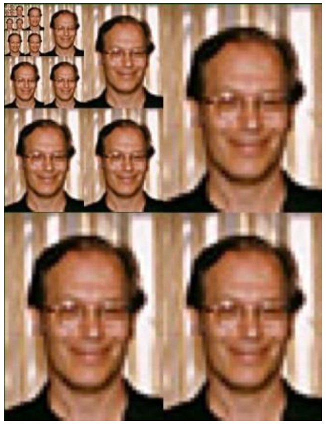
- “Multum in parvo” (much in little).
- One same texture is pre-processed and filtered at different smaller sizes to get better levels of details (LODs) and filtering.
- The optimal LOD is used according to the screen dimension of the primitive, leading to visually better results and faster rendering.





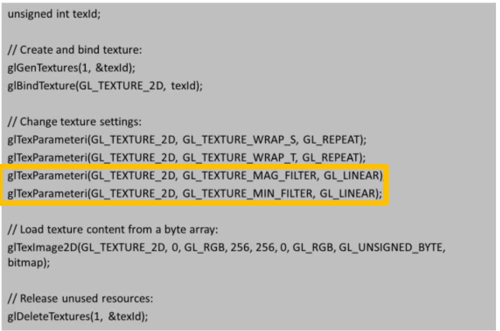
```
unsigned int texId;

// Create and bind texture:
glGenTextures(1, &texId);
glBindTexture(GL_TEXTURE_2D, texId);

// Change texture settings:
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_LINEAR)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

// Load texture content from a byte array:
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 256, 256, 0, GL_RGB, GL_UNSIGNED_BYTE,
bitmap);

// Release unused resources:
glDeleteTextures(1, &texId);
```

Texture filtering (mipmapping)

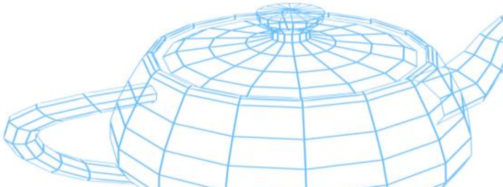


Lance Williams
1949

- Introduced by Lance Williams in 1983.
- Mipmaps require 1/3 additional VRAM to store all the LODs.
- Mipmaps are computed off-line, using the best filtering algorithms available and/or designer skills.
- Mipmaps can be procedurally generated:
  - **`gluBuild2DMipmaps();`** // Part of GLU, deprecated, computed on the CPU
  - **`glGenerateMipmap();`** // OpenGL 3.0+ only (or as extension before), hardware-accelerated
- You can also implement your own mipmap generator.



```
unsigned int texId;

// Create and bind texture:
glGenTextures(1, &texId);
glBindTexture(GL_TEXTURE_2D, texId);

// Change texture settings:
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_LINEAR)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

// Load texture content from a byte array:
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 256, 256, 0, GL_RGB, GL_UNSIGNED_BYTE,
bitmap);

// Release unused resources:
glDeleteTextures(1, &texId);
```



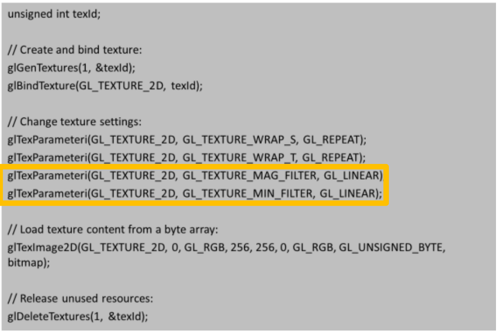
Texture filtering (mipmapping)

- OpenGL decides what mipmap LOD to use according to the size of the primitive during rasterization.

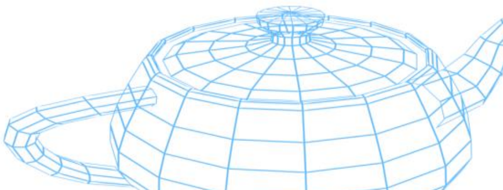
- If linear filtering is used, the proper mipmap subimage is further filtered.

- If trilinear filtering is used, the mipmap subimage is computed as the interpolation between the nearest two LODs:
  - Trilinear filtering is activated using:

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                               GL_LINEAR_MIPMAP_LINEAR);
```



```
unsigned int texId;

// Create and bind texture:
glGenTextures(1, &texId);
glBindTexture(GL_TEXTURE_2D, texId);

// Change texture settings:
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_LINEAR)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

// Load texture content from a byte array:
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 256, 256, 0, GL_RGB, GL_UNSIGNED_BYTE,
bitmap);

// Release unused resources:
glDeleteTextures(1, &texId);
```



Texture mapping

- Texture mapping is activated by invoking **`glEnable(GL_TEXTURE_2D);`**
  - 1D and 3D texture mapping work in a similar way.
  - The texture currently set via **`glBindTexture()`** is used during rasterization.

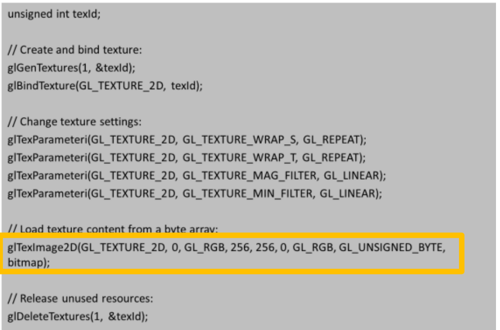
- For performance reasons, textures are stored on dedicated device memory:
  - Load once, reuse often:

mipmap level

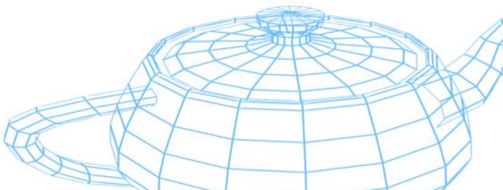
```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB,
                 GL_UNSIGNED_BYTE, data);
```

  - To update a previously loaded texture (or a sub-region):

```
glTexSubImage2D(GL_TEXTURE_2D, 0, xOffset, yOffset, width, height,
                    GL_RGB, GL_UNSIGNED_BYTE, data);
```

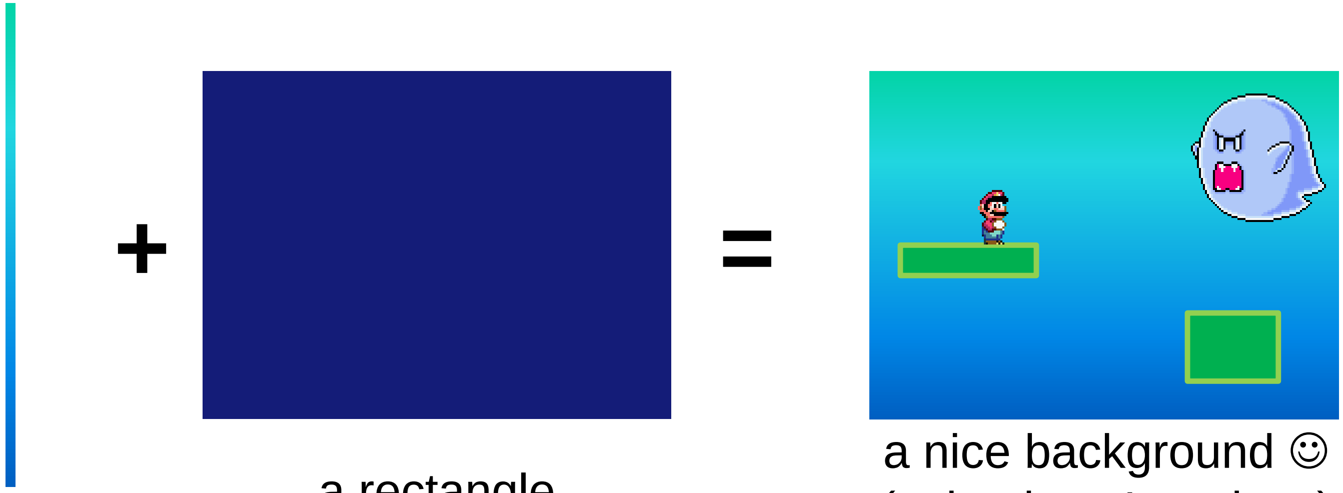


```
unsigned int texId;

// Create and bind texture:
glGenTextures(1, &texId);
glBindTexture(GL_TEXTURE_2D, texId);

// Change texture settings:
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_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

// Load texture content from a byte array:
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 256, 256, 0, GL_RGB, GL_UNSIGNED_BYTE,
bitmap);

// Release unused resources:
glDeleteTextures(1, &texId);
```



Texture mapping

- 1D texture mapping:
  - Used to accelerate conventional rendering or to use textures as 1D data arrays for shaders.



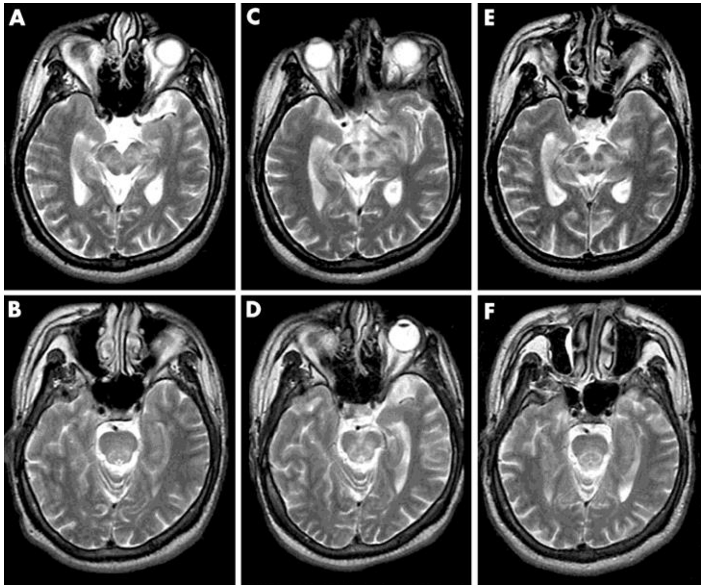
1D texture

a rectangle

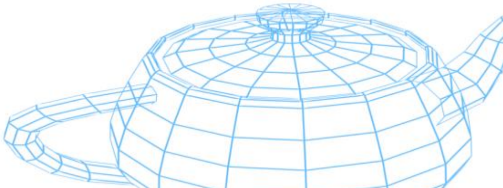
a nice background ☺
(using just 4 vertices)

Texture mapping

- 3D texture mapping (a. k. a. “volumetric texture mapping” or “*voxel* space”):
  - Medical imagery (body scan, CAT, etc.).
  - Terrain engines (to use different textures according to the height).

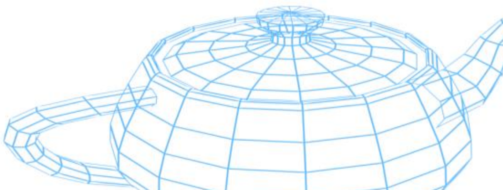


voxel = VOlumetric piXEL or volumetric picture element



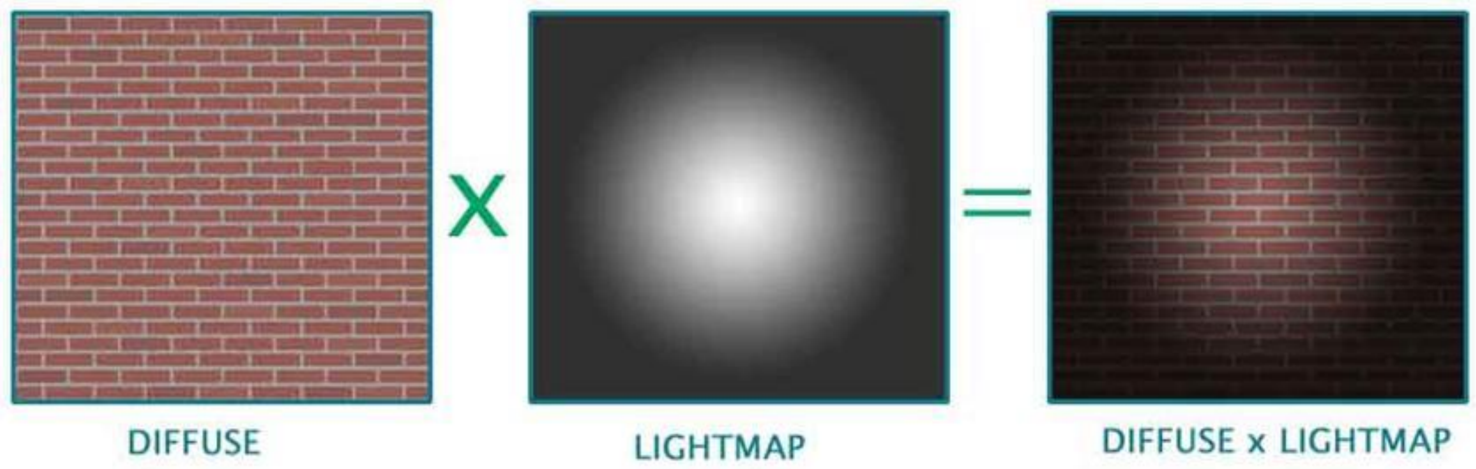
Multitexturing

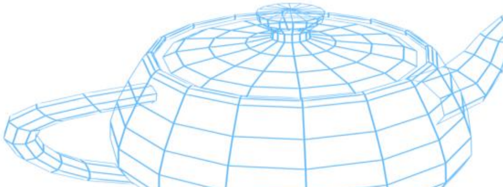
- More than one single texture can be used at the same time during the rendering of primitives.
- For each texture, a different set of texturing coordinates can be specified:
  - Using multiple coordinate levels.
- The way multiple textures interact is specified by the programmer:
  - Typically, using fragment shaders or register combiners on older versions of OpenGL.
  - Many advanced techniques rely on multitexturing, like deferred rendering, depth peeling, normal mapping, etc.



Multitexturing

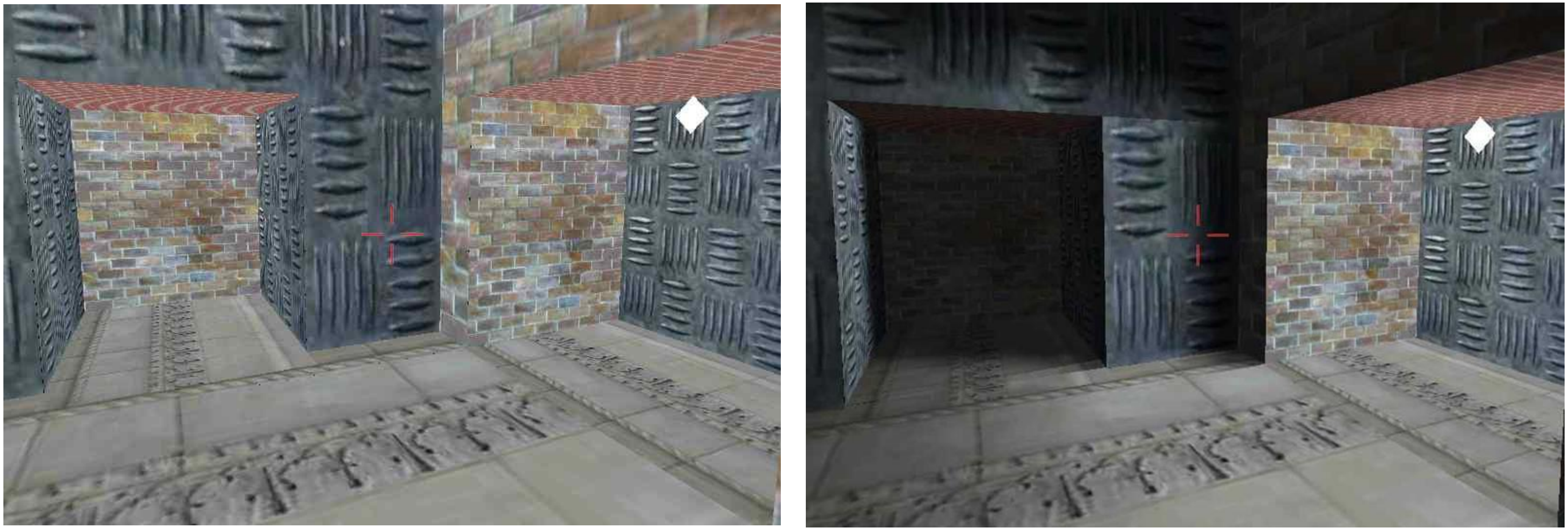
- Lightmapping is a technique based on multitexturing to provide **static** lighting to the scene.
- Each object in the scene uses a second texture where (pre-computed) illumination information is stored, e.g.:

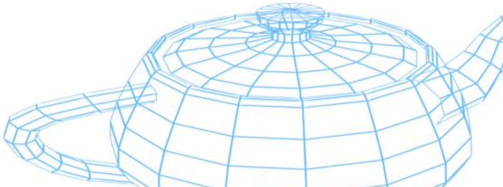




Multitexturing

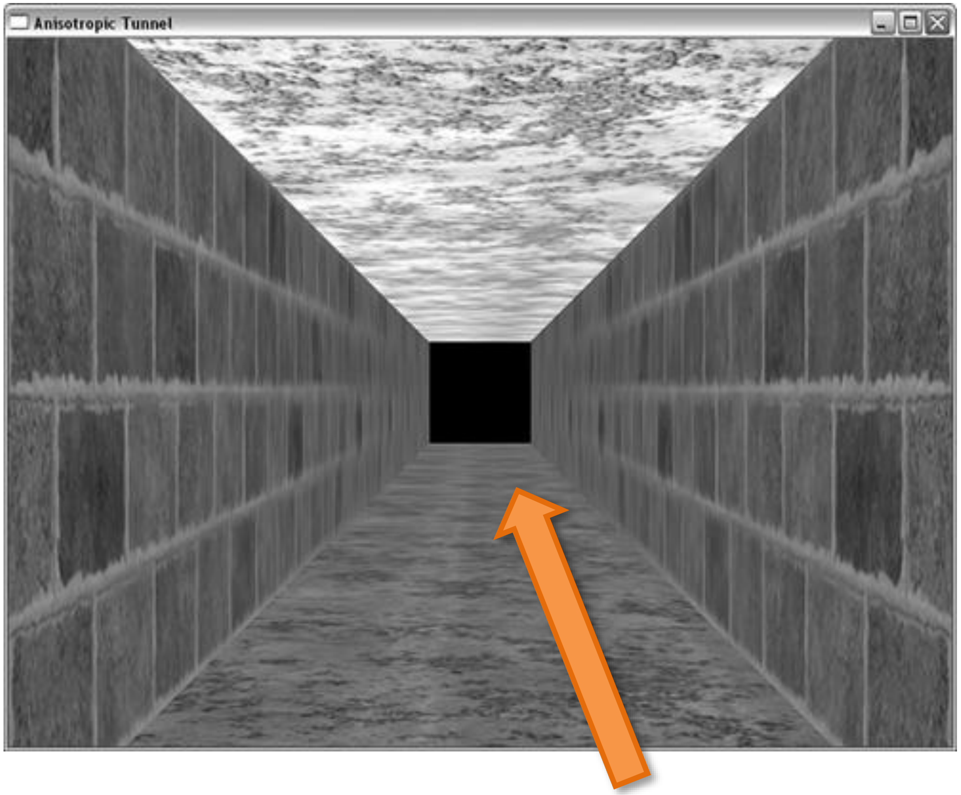
- The lightmaps are computed off-line, by using a ray-tracer or other techniques (which might include radiosity, shadows, global illumination, etc.).





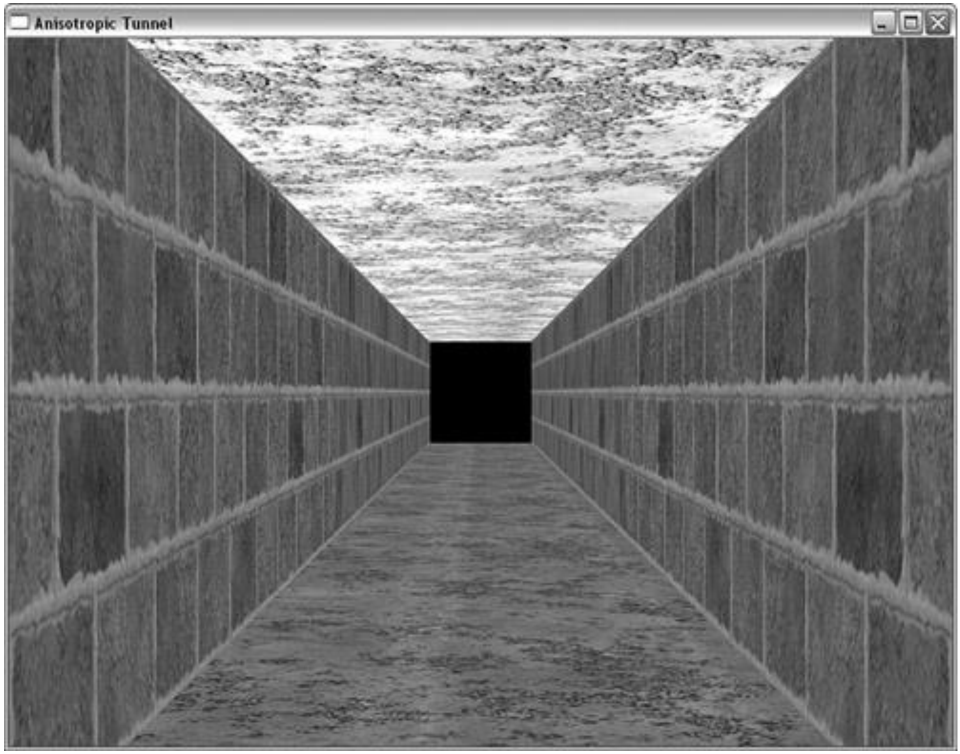
Anisotropic filtering

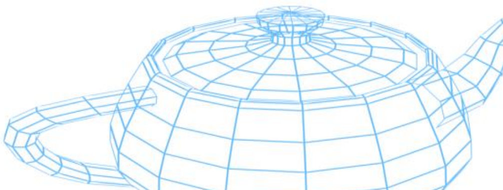
Trilinear filtering



Significant blur due to excessive filtering

Trilinear filtering + anisotropic filtering



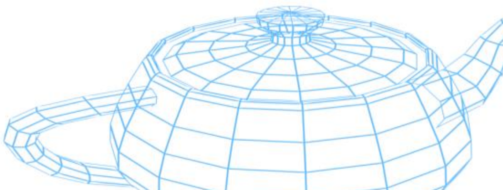


Anisotropic filtering

- Anisotropic filtering takes the view angle in account and uses more samples to increase signal frequency and reduce blur in textures that are oblique to the viewer.

- Available through the extension **`GL_EXT_texture_filter_anisotropic`**.

- New per-texture-object setting activated through:
  `glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAX_ANISOTROPY_EXT, value);`
  - Where 1 ≤ *value* ≤ *maxAnisotropy.*
  - *maxAnisotropy* is usually 8 or 16 and it is determined through:
    `glGetFloatv(GL_MAX_TEXTURE_MAX_ANISOTROPY_EXT, &maxAnisotropy);`



Read pixels

- **`glReadPixels()`** copies the content (or a portion) of the OpenGL framebuffer into a user-specified memory buffer.

-
```
glReadPixels(xOffset, yOffset, width, height, GL_RGB,
                    GL_UNSIGNED_BYTE, bufferPtr);
```
  - When double-buffering is used, **`glReadPixels()`** reads data from the back buffer by default.

- The bitmap retrieved from the framebuffer can be used as bitmap for a texture.

- **`glReadPixels()`** can be used to get a screenshot of the image rendered by OpenGL, e.g., to save it to file.