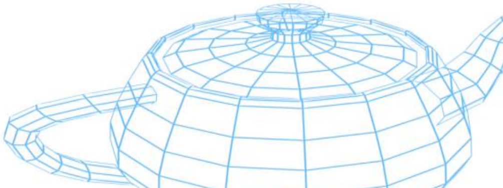
SUPSI

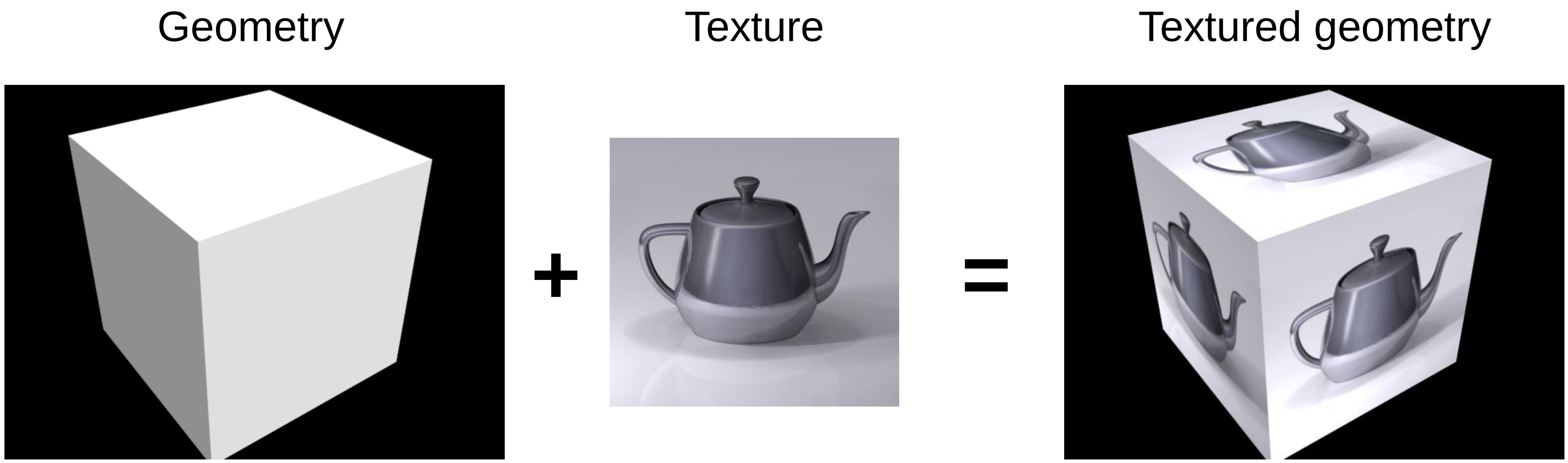
Computer Graphics

OpenGL (4): Texture mapping

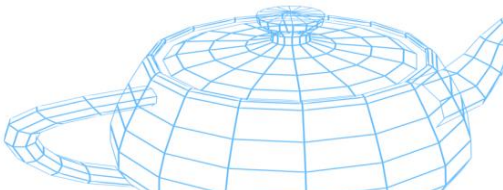
Achille Peternier, adjunct professor

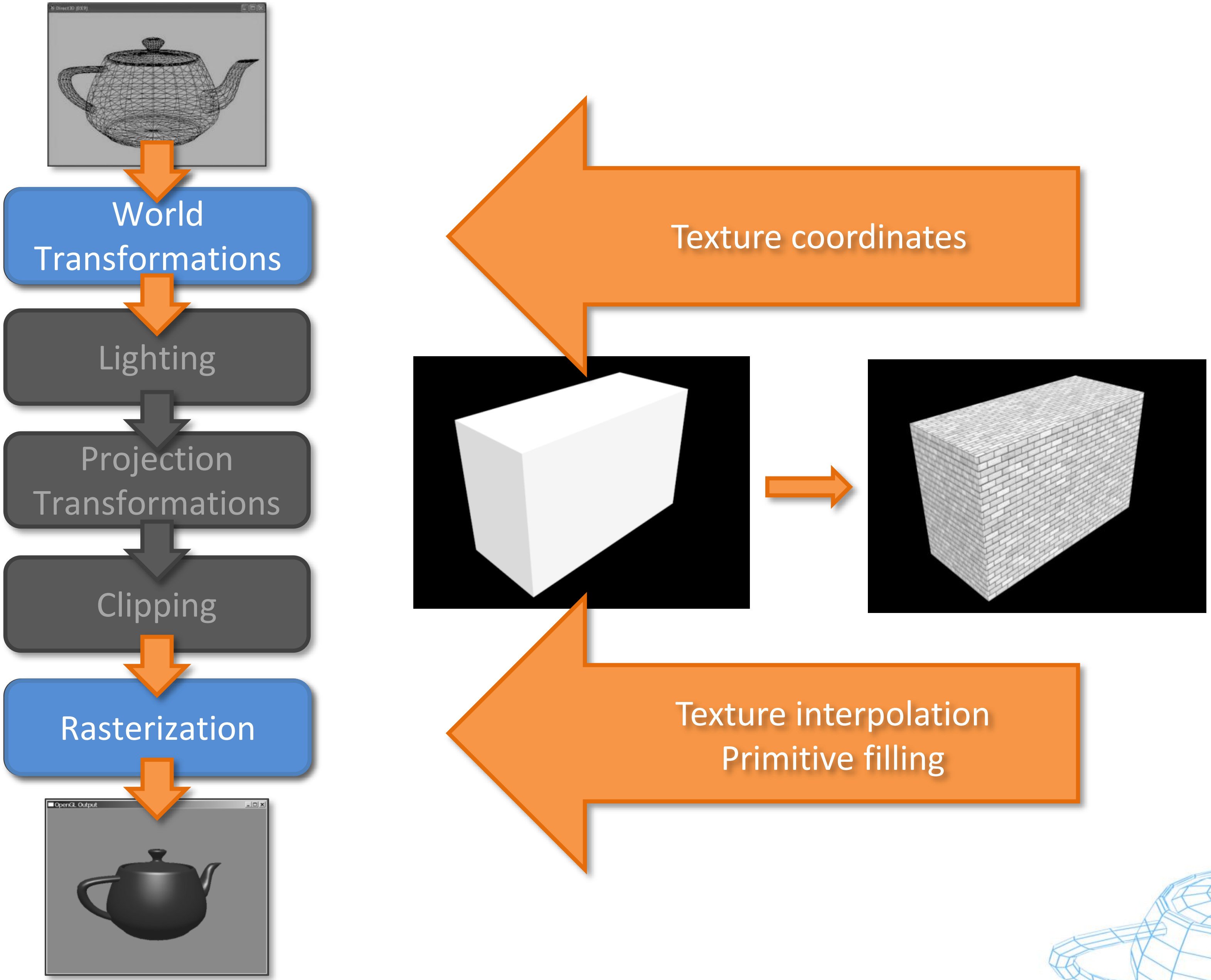


Texture mapping



Geometry + Texture = Textured geometry





World Transformations

Lighting

Projection Transformations

Clipping

Rasterization

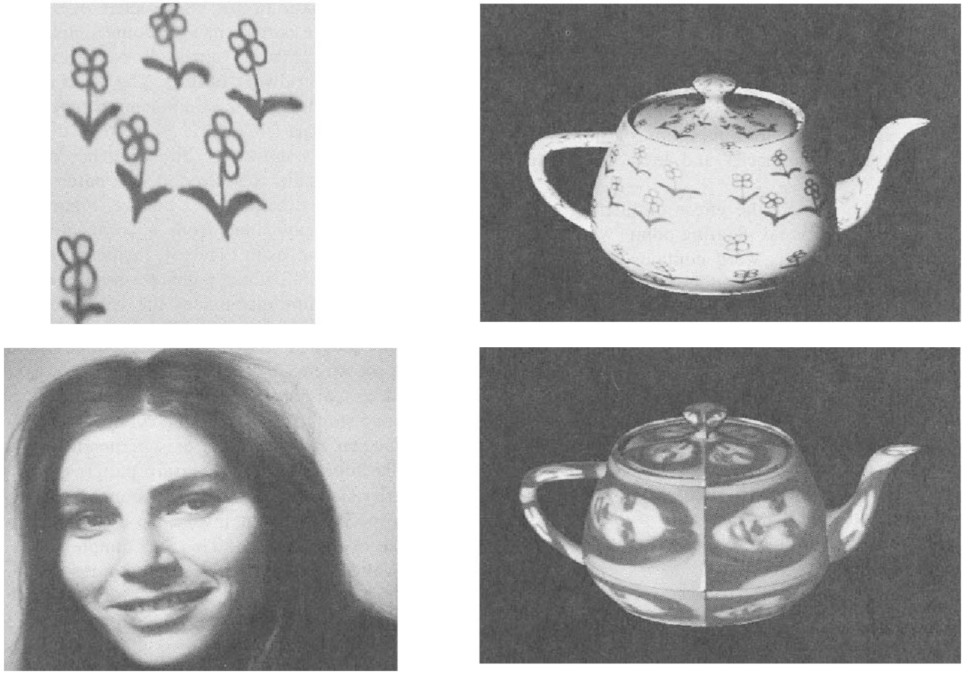
Texture coordinates

Texture interpolation
Primitive filling

Texture mapping

- Textures are images used for “painting” primitives during rasterization to provide additional detail without requiring additional geometry.

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

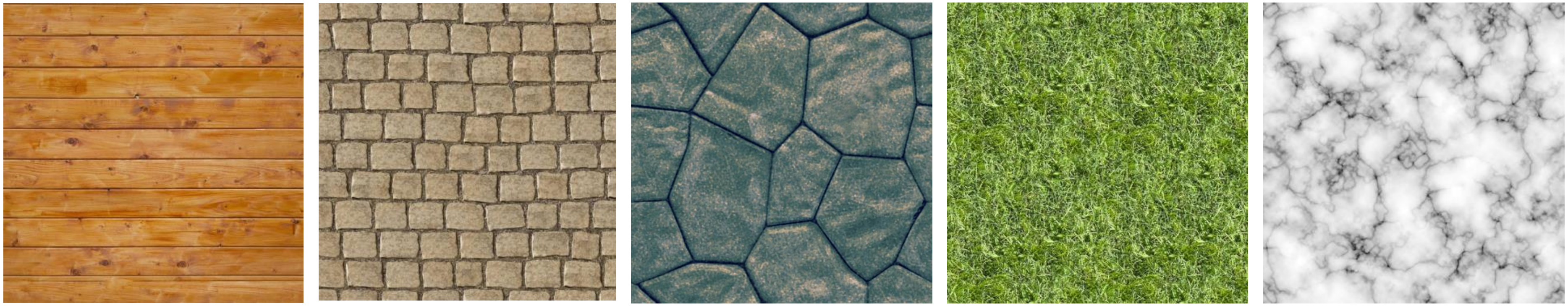


Edwin Catmull
1945

A bit of history…

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/

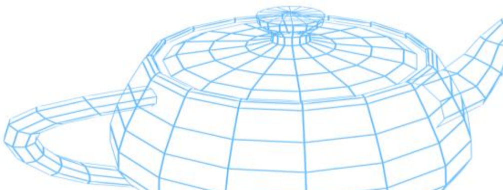


Texture mapping

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

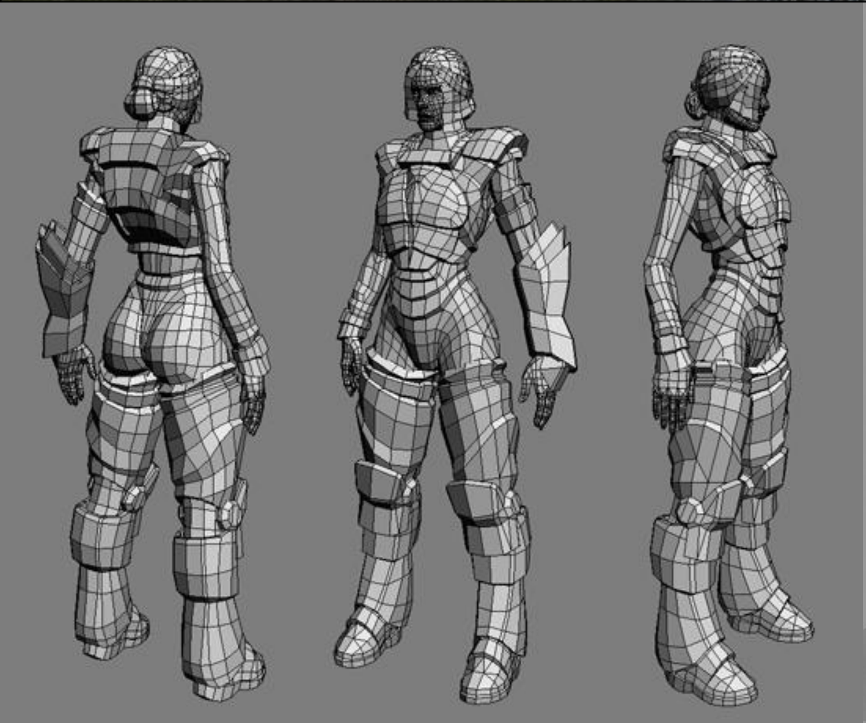
- During rasterization, each *texel*[*] color is multiplied by the color computed by the lighting model or directly specified by the programmer:
  - You can change this default setting via **`glTexEnv*();`**

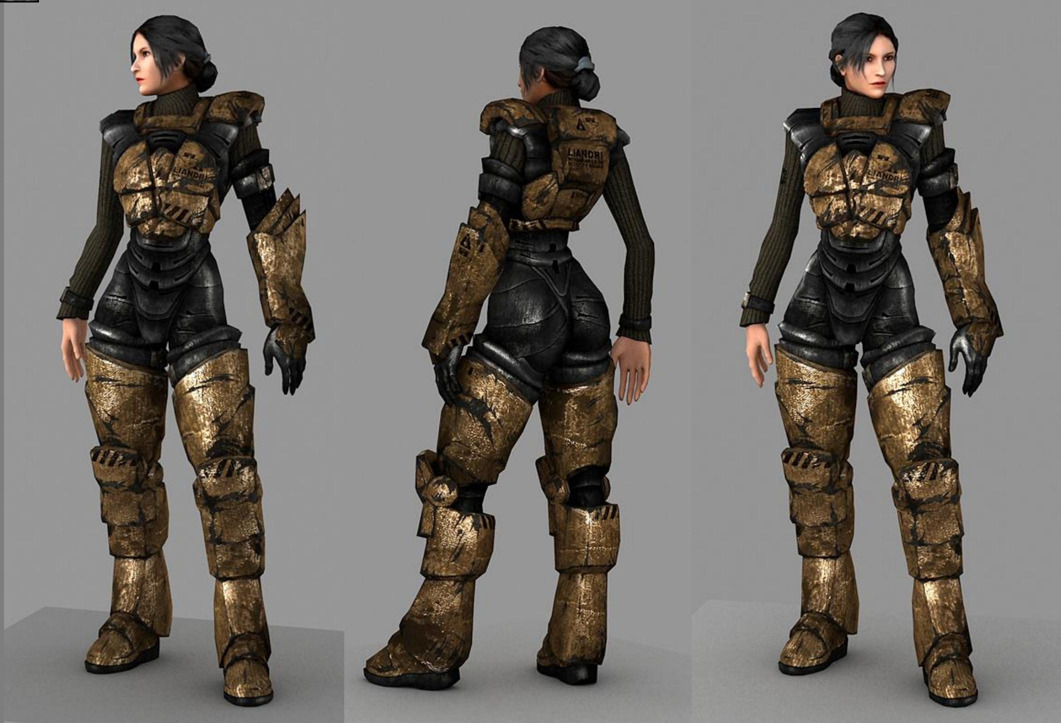
- Texture mapping is widely used in modern computer graphics for implementing a series of advanced techniques such as shadow mapping, deferred rendering, physically-based materials, real-time global illumination, etc.:
  - There’s a reason behind the tons of VRAM in today’s consumer graphics cards…



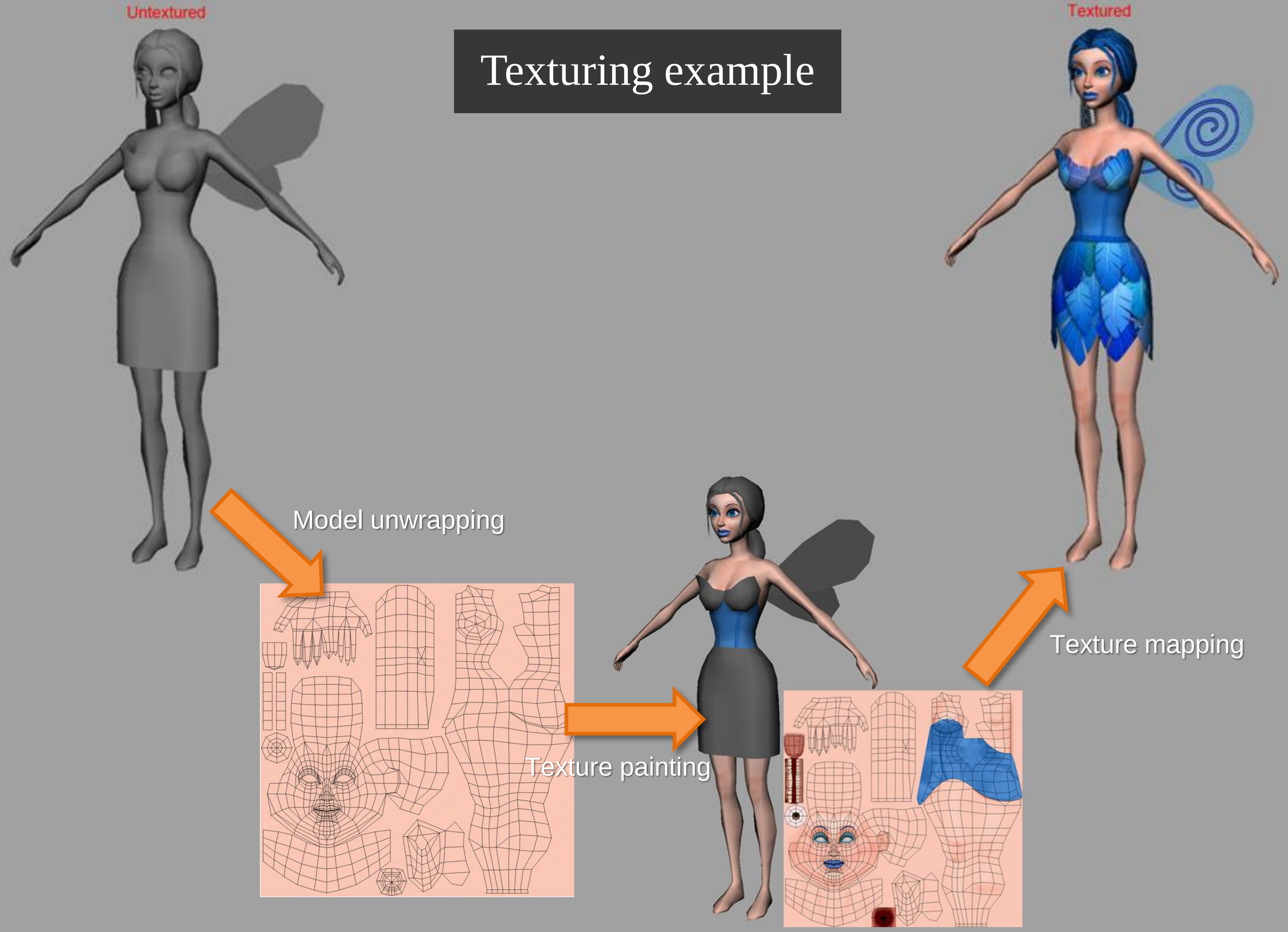
*) ***TEX**ture **EL**ement (texel)*

Textured models





Texturing example



Model unwrapping

Texture painting

Texture mapping

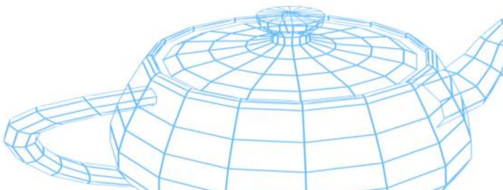
https://software.intel.com/en-us/articles/creating-textures-for-characters-in-autodesk-maya

Texture mapping

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

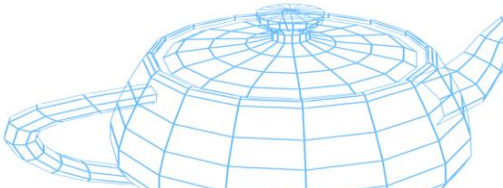
- Sizes are then normalized into the [0, 1] range:
  - …in the same way normalized device coordinates abstract from real screen sizes.

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



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 coordinates

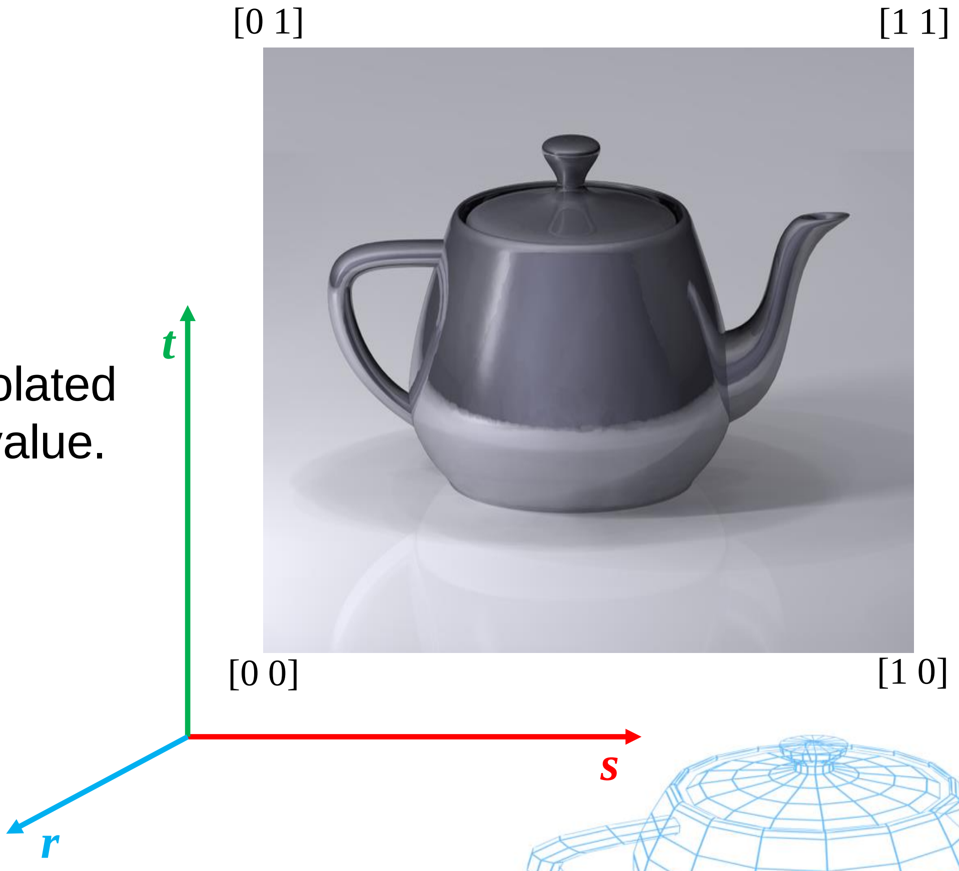
- Texture coordinates are expressed through 1D, 2D, and 3D coordinates defined as *s*, *t*, and *r* :

  s = u = x dimension

  t = v = y dimension

  r = w = z dimension

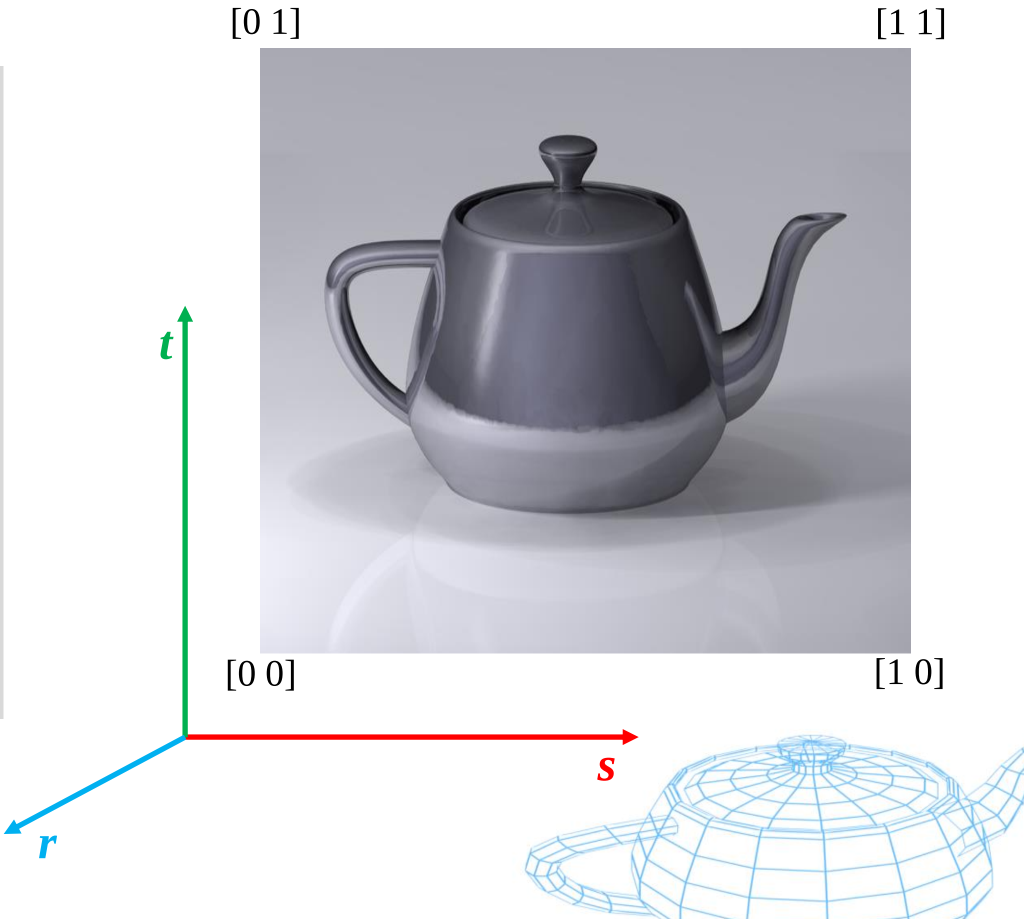
- Texture coordinates are also interpolated during rasterization, like any other value.



Texture coordinates

- 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, 0.0f);

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

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

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

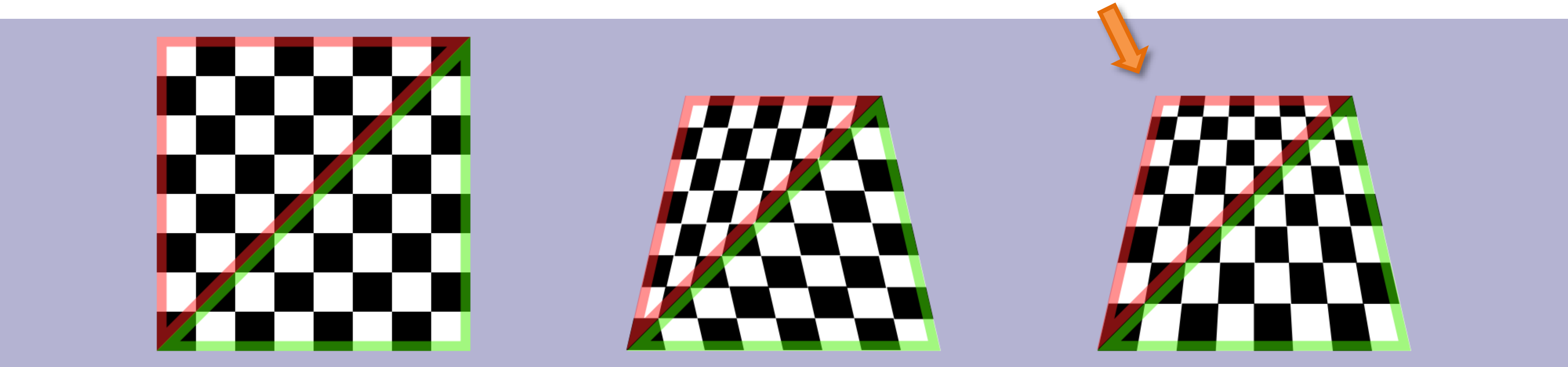


Texture coordinates

- Texture coordinates specified at each vertex are interpolated across the primitive.

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

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



Texture mapping

Texture creation and configuration

```
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);
```

Texture destruction

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

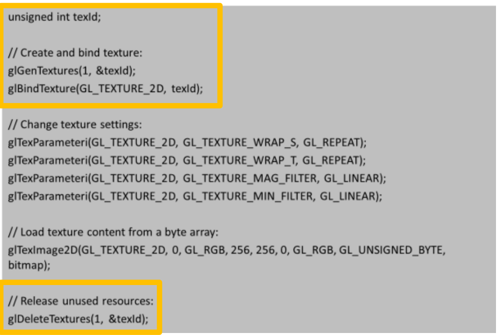
Texture utilization

```
// Each time you want to use a texture, simply:
glBindTexture(GL_TEXTURE_2D, texId);
glEnable(GL_TEXTURE_2D);
```

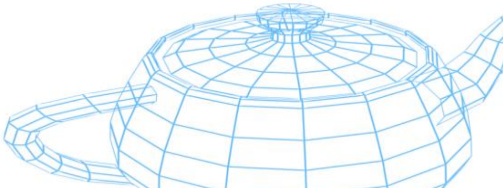
Texture creation and destruction

- Each texture object generated by OpenGL has a name (as an unsigned integer identifier) and 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 a 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);
```

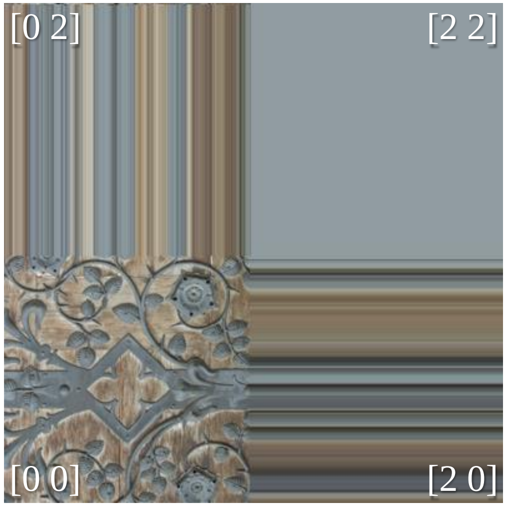


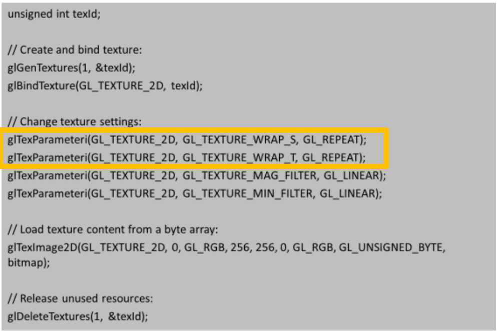
Texture wrapping

- When texture coordinates are not in the range [0, 1], you can tell OpenGL what to do. The most used options are:
  - Lower/higher values are clamped to 0 or 1.
  - Coordinates become circular to repeat 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*);
```





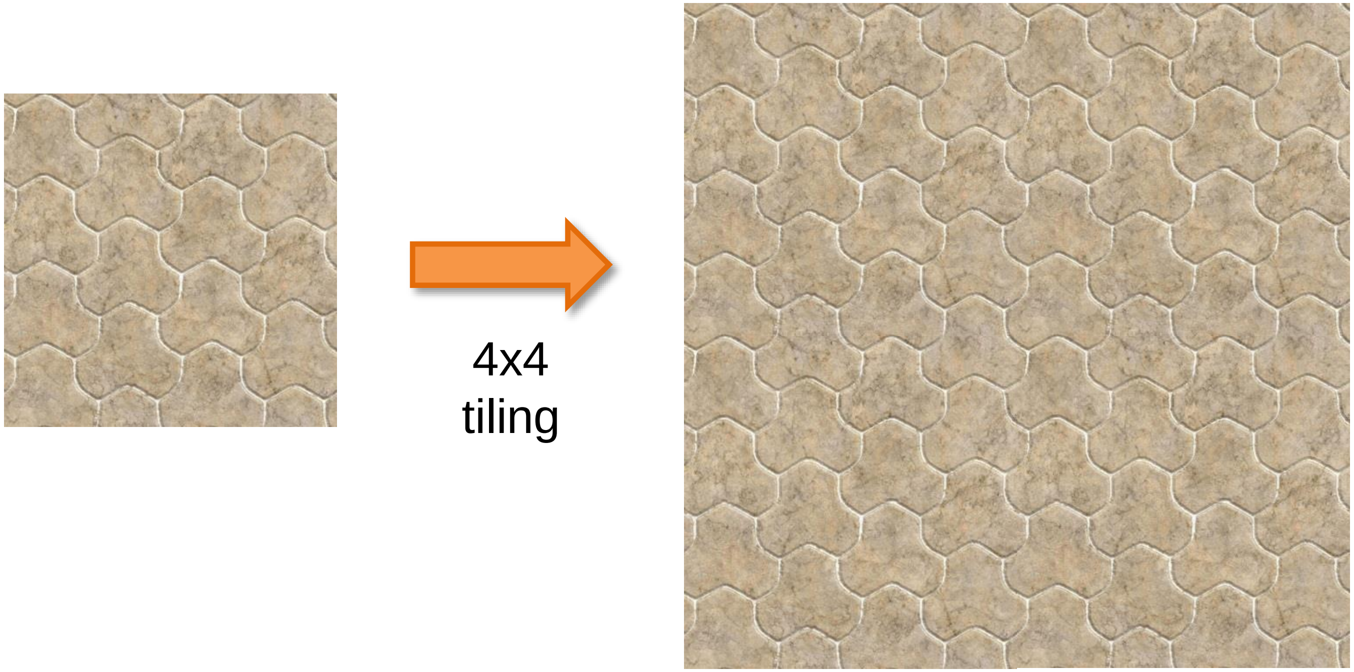
```
unsigned int texId;

// 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);
```

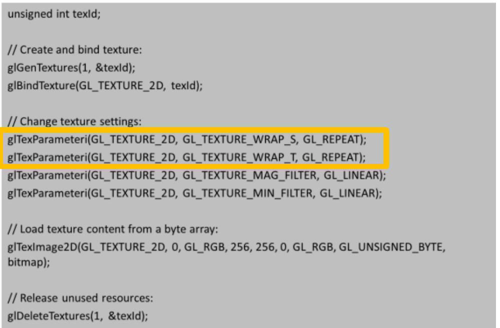
**) available since OpenGL 1.2*

Tileable textures

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

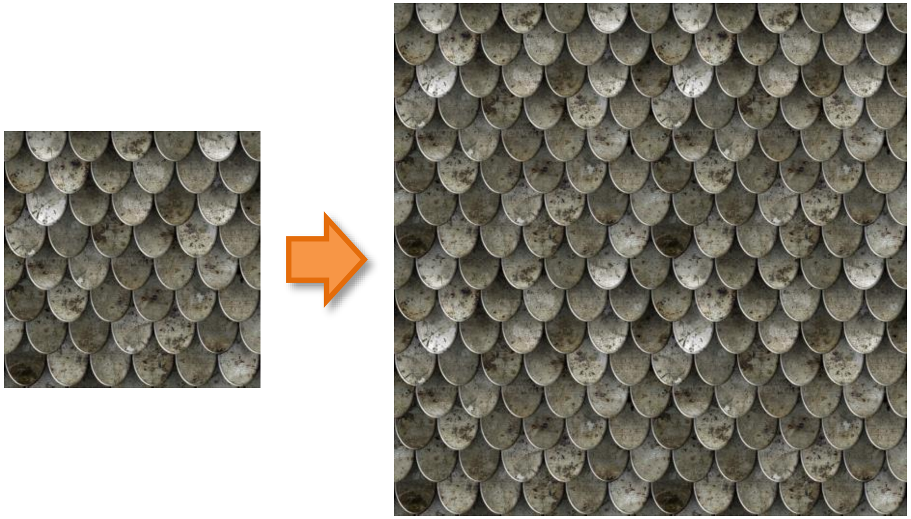
- Tileable textures are seamless images that can be put one next to the other without glitches:

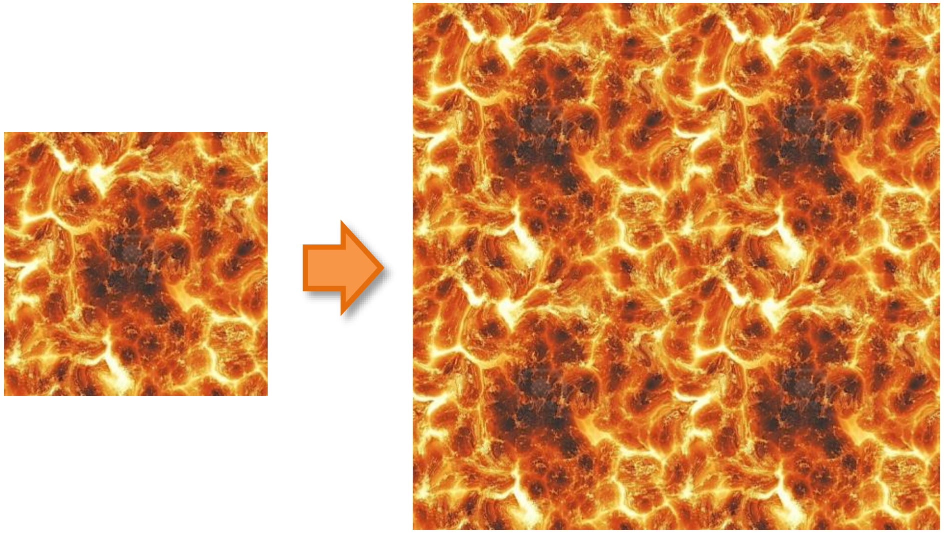




```
unsigned int texId;

// 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);
```

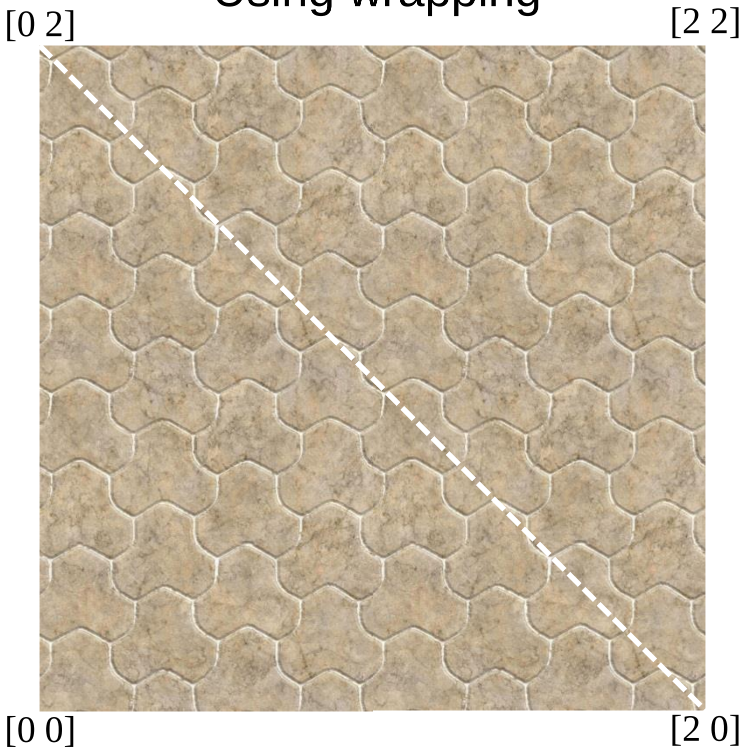
Tileable textures



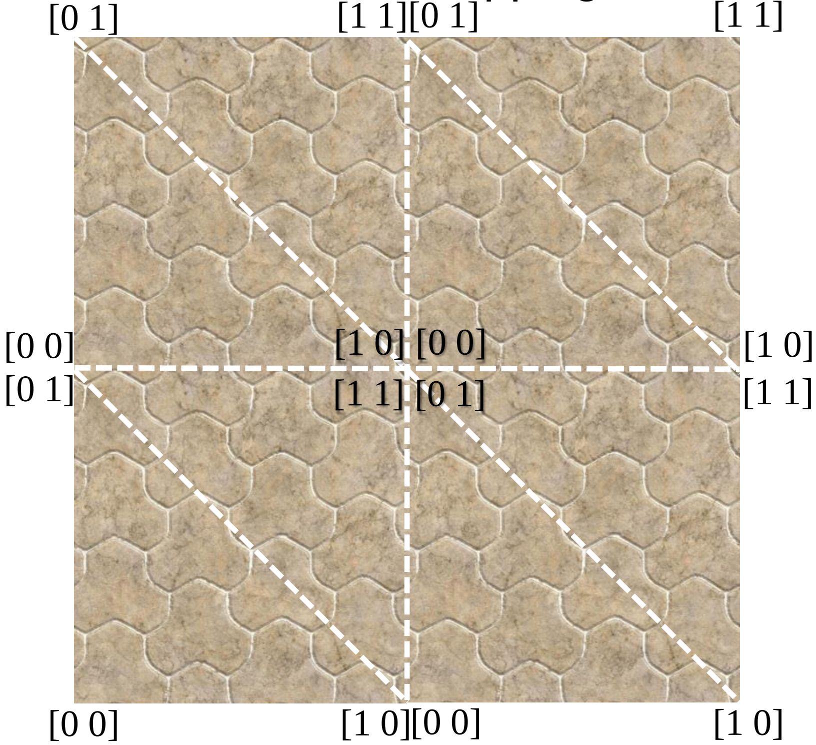


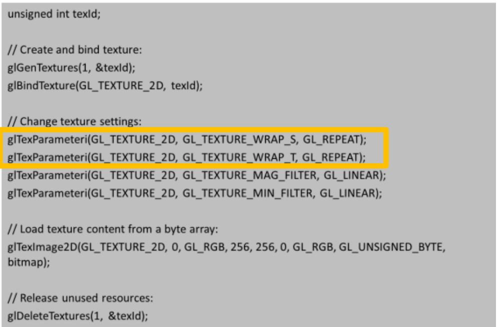
Tileable textures

Using wrapping

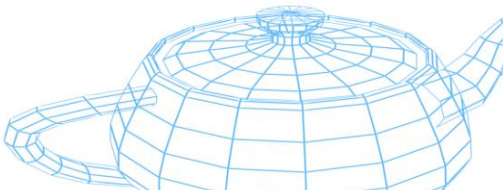


Without wrapping



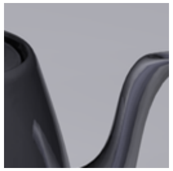


```
unsigned int texId;

// 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);
```

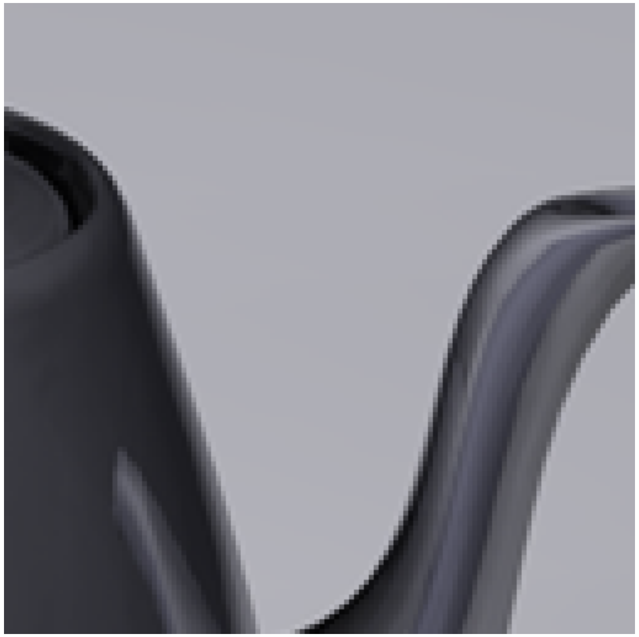


Texture filtering: linear

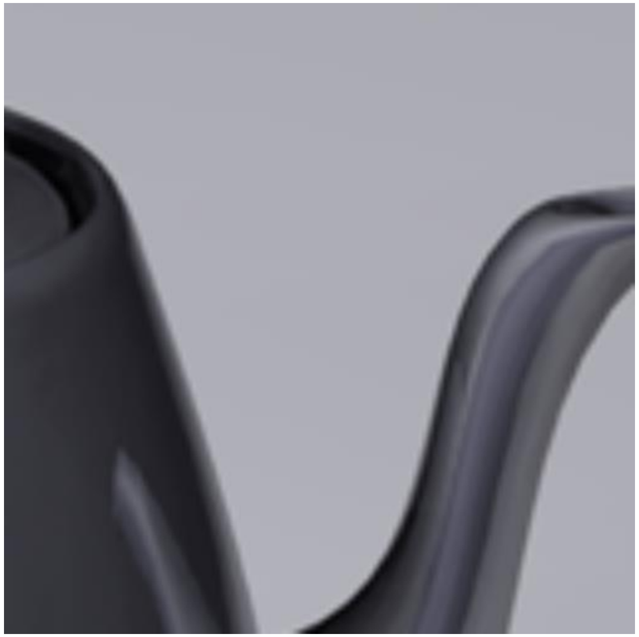
- Since textures are based on raster images, they have a finite resolution:
  - Zooming in (magnification) causes aliasing.



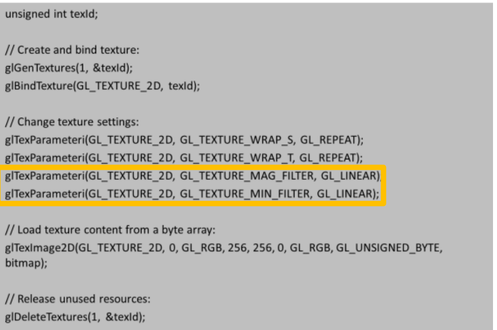
original image



no filtering (**GL_NEAREST**)



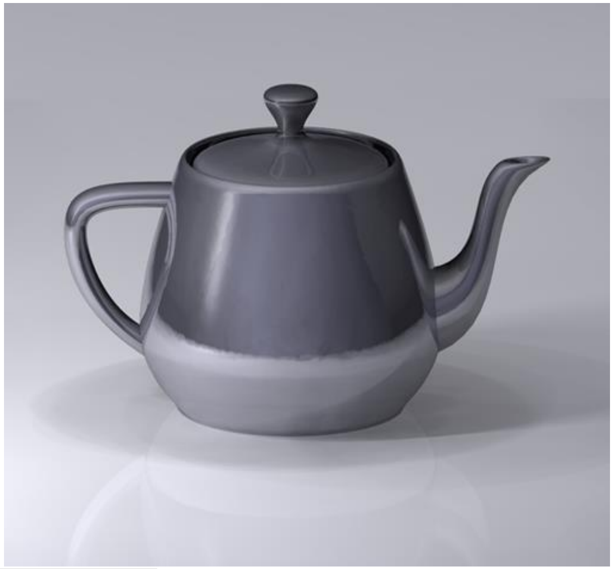
linear filtering (**GL_LINEAR**)



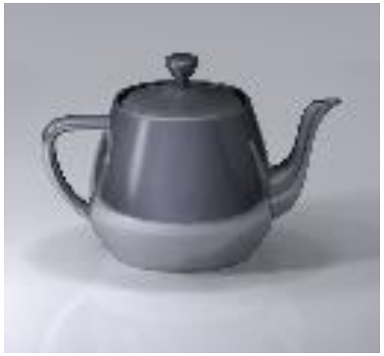
```
unsigned int texId;

// 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: linear

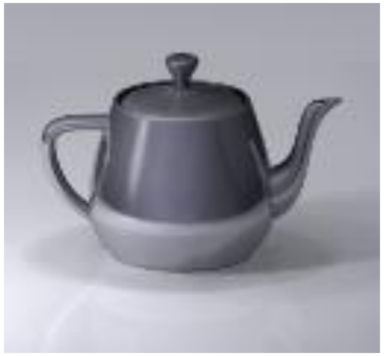
- Since textures are based on raster images, they have a finite resolution:
  - Zooming out (minimization) causes jittering.



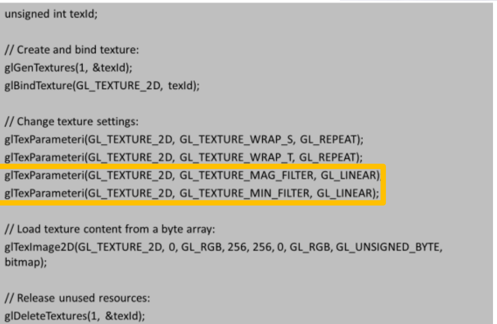
original image



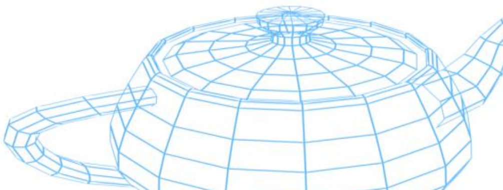
no filtering (**`GL_NEAREST`**)



linear filtering (**`GL_LINEAR`**)

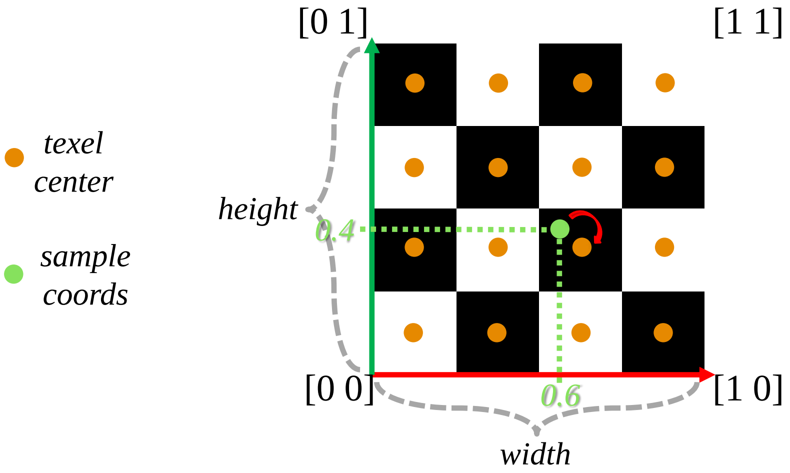


```
unsigned int texId;

// 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);
```



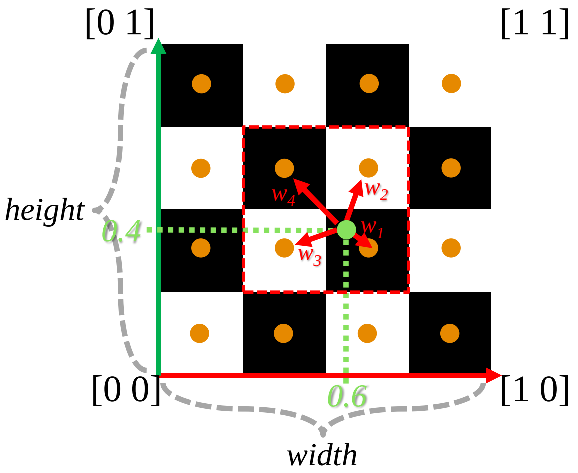
Texture filtering: linear

- Filtering of sample at *uv* = [0.6, 0.4] for an image of *width* x *height* of 4x4 pixels:



no filtering
(**`GL_NEAREST`**)

sample(u, v) = RGB(0, 0, 0)



linear filtering
(**`GL_LINEAR`**)

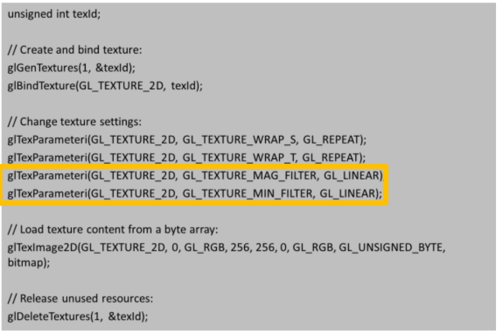
sample(u, v) = w_1 x RGB(0, 0, 0) +
w_2 x RGB(1, 1, 1) +
w_3 x RGB(1, 1, 1) +
w_4 x RGB(0, 0, 0)

Texture filtering: linear

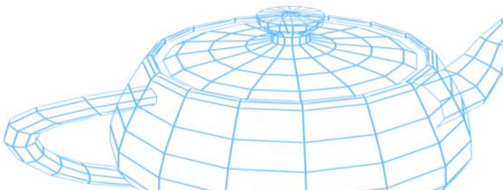
- Filtering requires additional computational power but is done by OpenGL, using the available hardware acceleration.

- Filtering is enabled through:

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

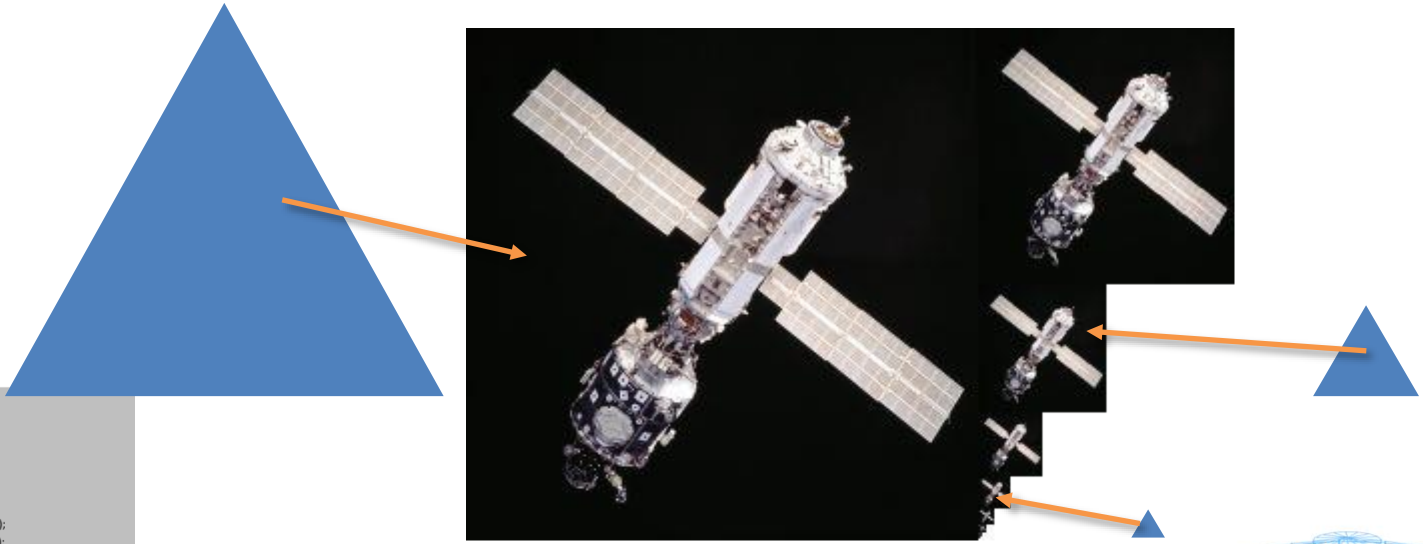


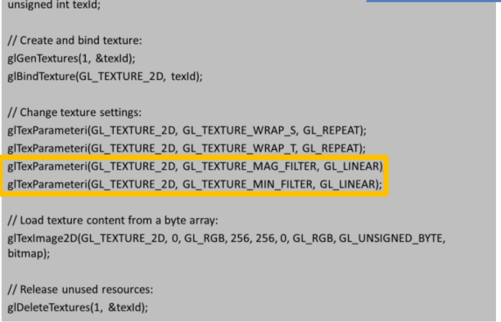
```
unsigned int texId;

// 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);
```



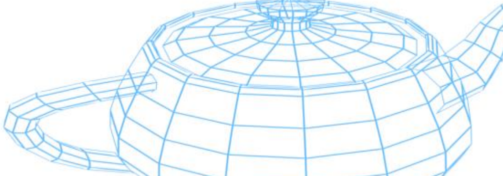
Texture filtering: mipmapping

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

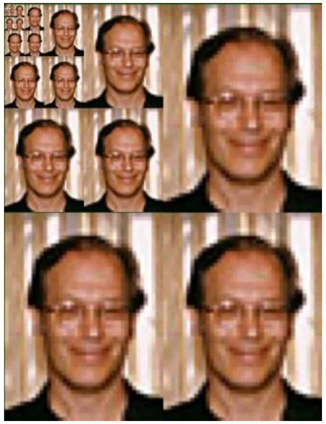




```
unsigned int texId;

// 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);
```

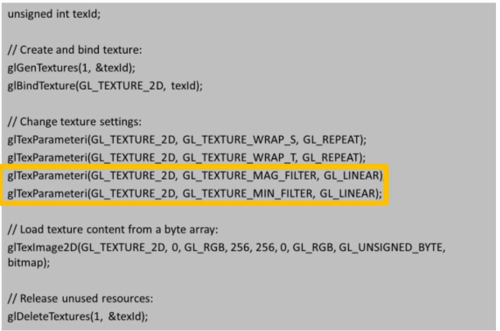


Texture filtering: mipmapping

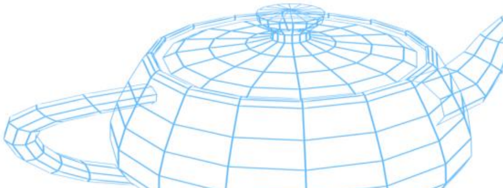


Lance Williams
1949

- 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

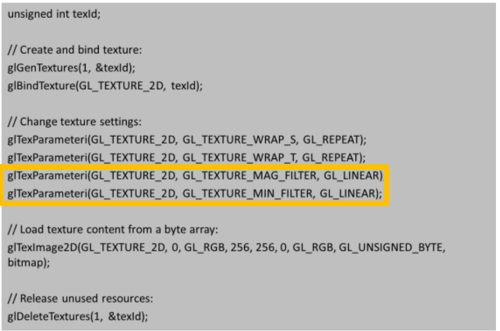
- OpenGL decides what mipmap LOD to use according to the size of the primitive during rasterization.

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

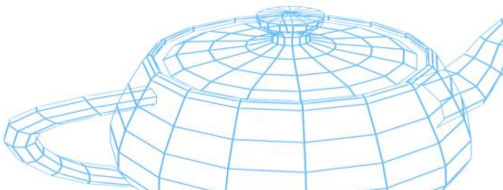
- If trilinear filtering is used, the mipmap subimage is computed as the interpolation between the nearest two LODs:
  - Trilinear filtering is activated using:

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

1 2 3

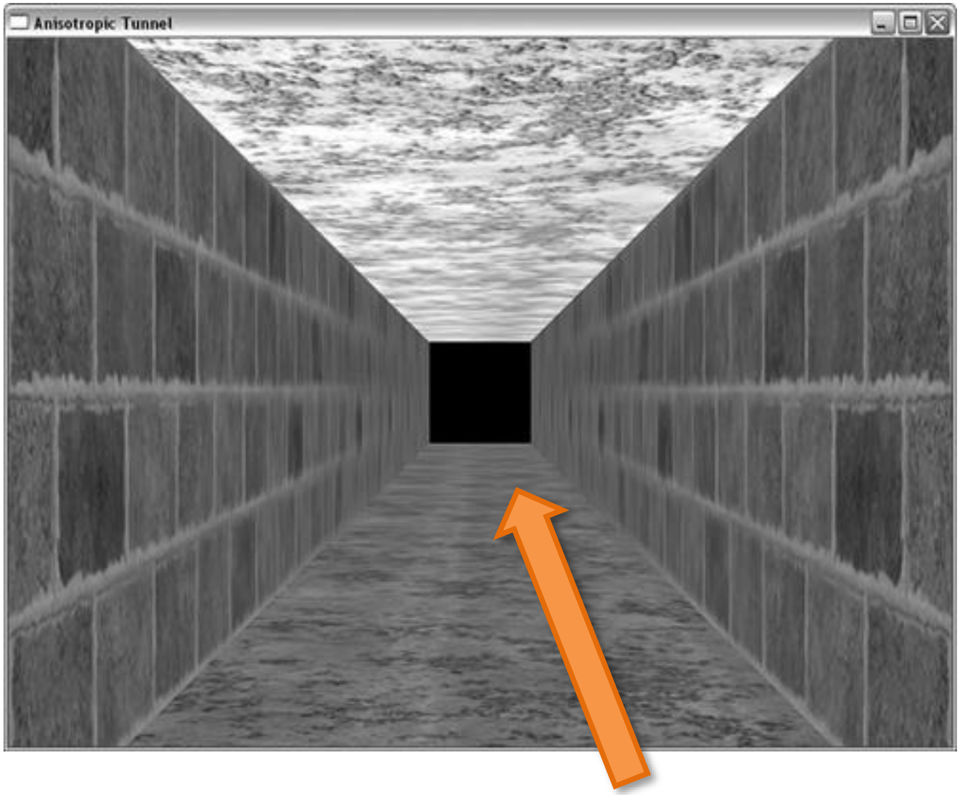


```
unsigned int texId;

// 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);
```

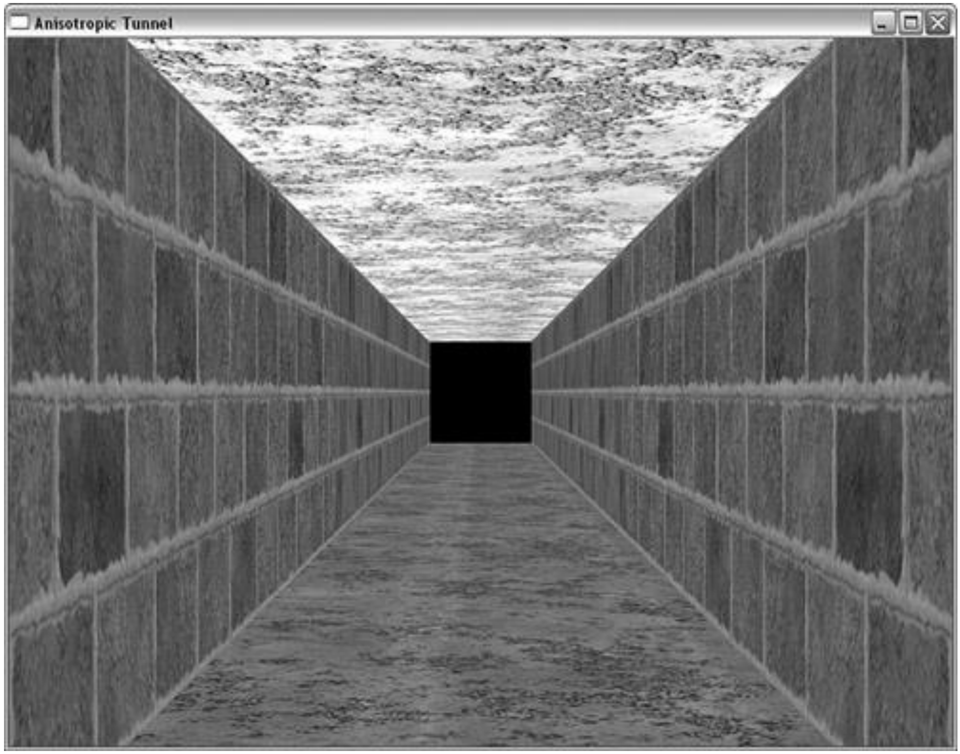


Texture filtering: anisotropic

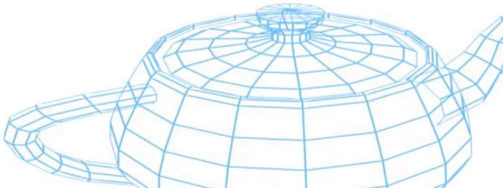
Trilinear filtering



Trilinear filtering + anisotropic filtering



Significant blur due to excessive filtering



Texture filtering: anisotropic

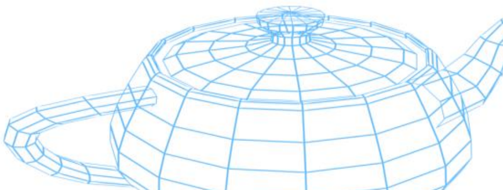
- Anisotropic filtering takes the view angle in account and uses more samples to increase signal frequency and reduce blur in textures that are oblique to the viewer.

- 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);`



Texture mapping

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

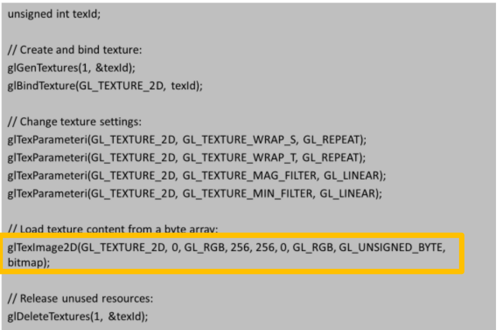
- For performance reasons, textures are stored on dedicated device memory:
  - Load once, reuse often:

mipmap level

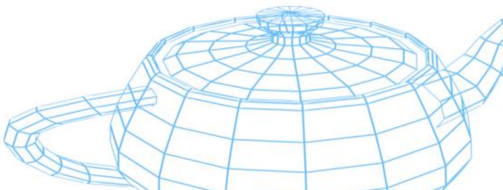
```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB,
             GL_UNSIGNED_BYTE, data);
```

  - 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);
```



Tutorial

Texture mapping