h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Visual Computing

Grafische Objekte und deren Programmierung

Hochschule Darmstadt

Björn Frömmer

Prof. Dr. Benjamin Meyer

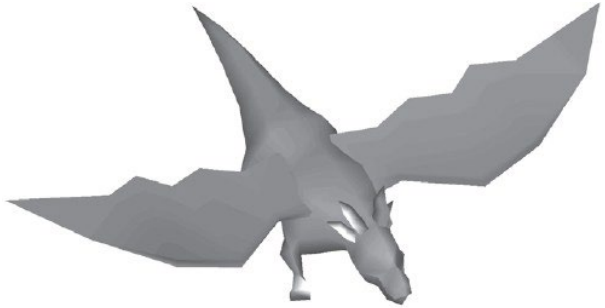
KAPITEL 9b

Texturierung

Why textures?

- Isn’t geometry and materials all you need?

Model



Textured Model

www.realtimerendering.com

Texture

Why textures?

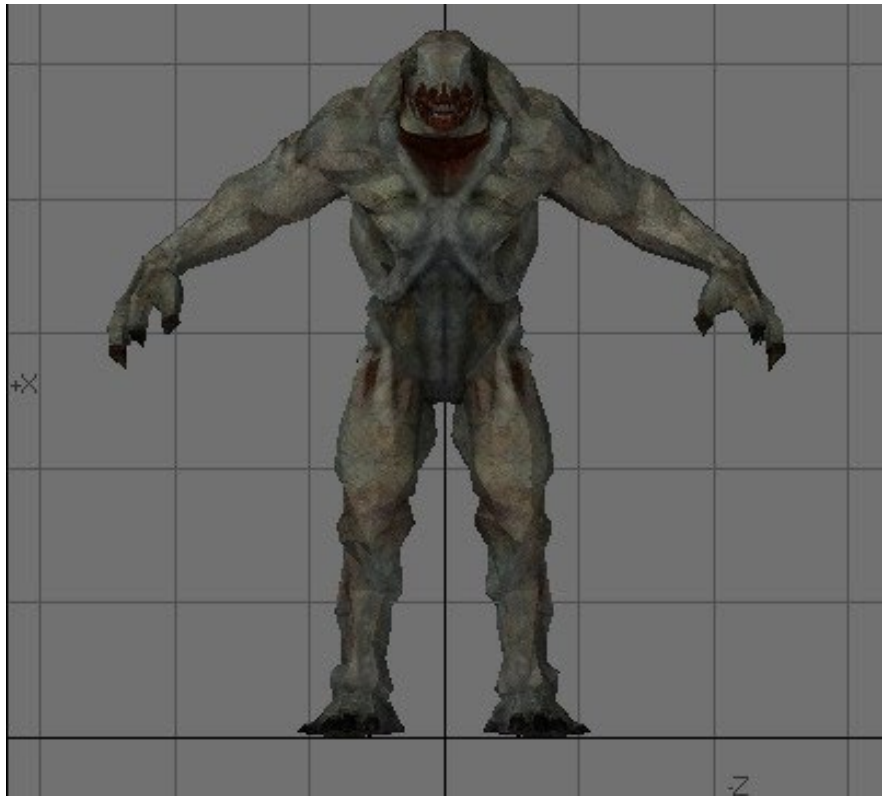
- Isn’t geometry and materials all you need?

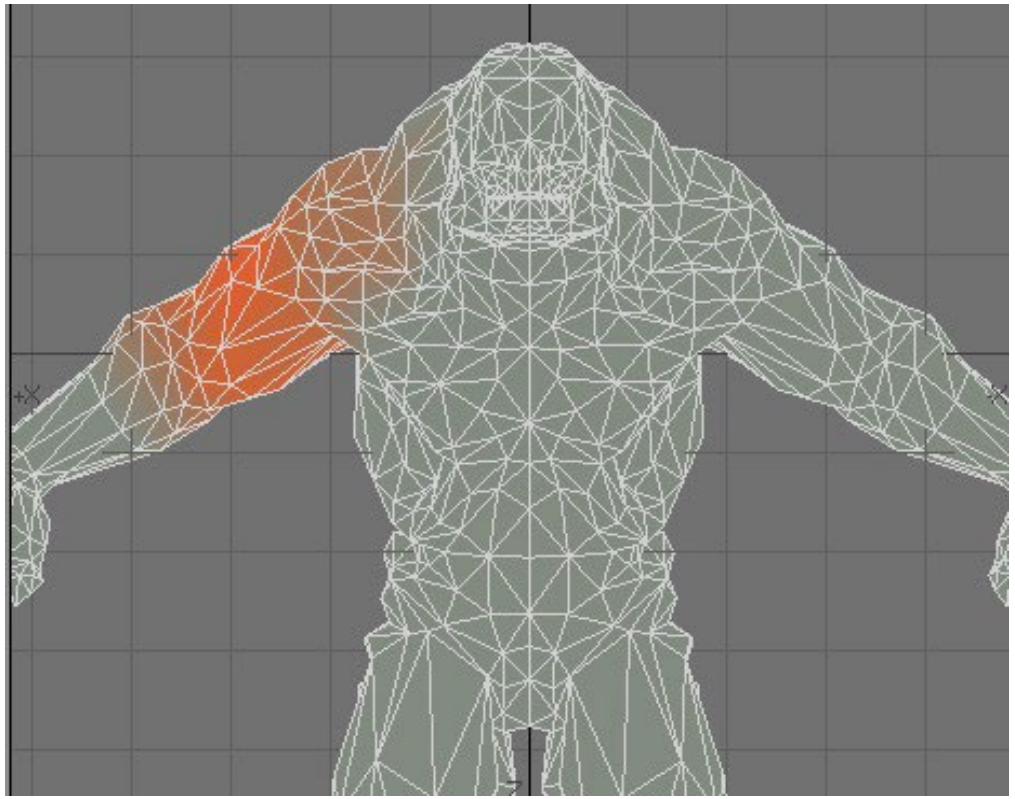
But:

- Rendering detailed geometry gets expensive
- It‘s a horrible lot of work to model and animate
- Better workflow: Separate material from geometry
- Allows to precompute / simulate lighting

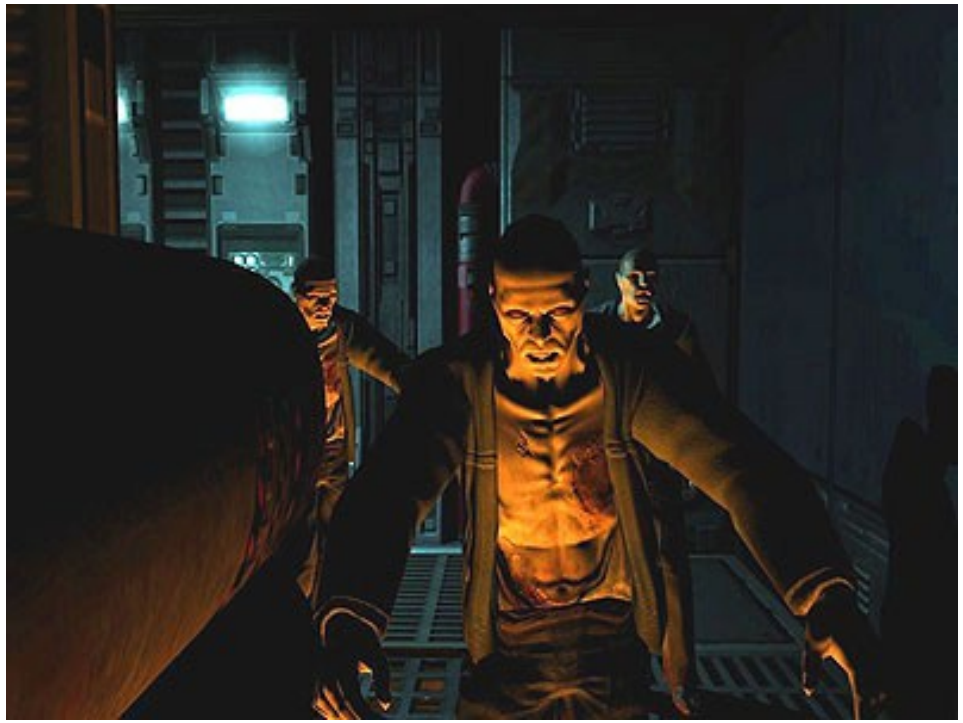
Textures

Textures





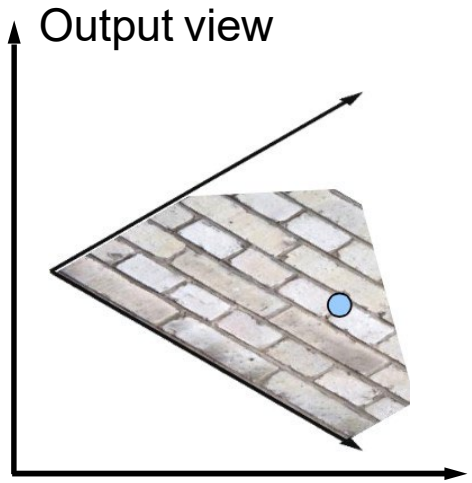
Textures

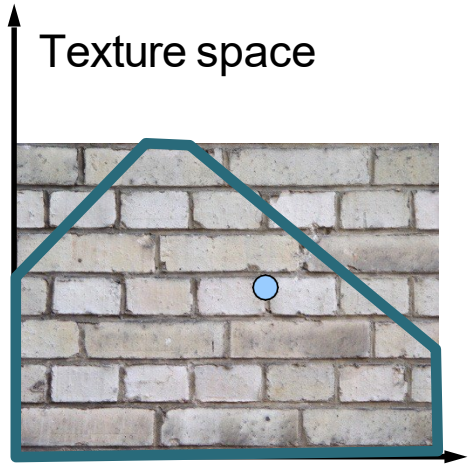


Texturing

- Rasterizer (Fragment Shader) maps texture onto polygon

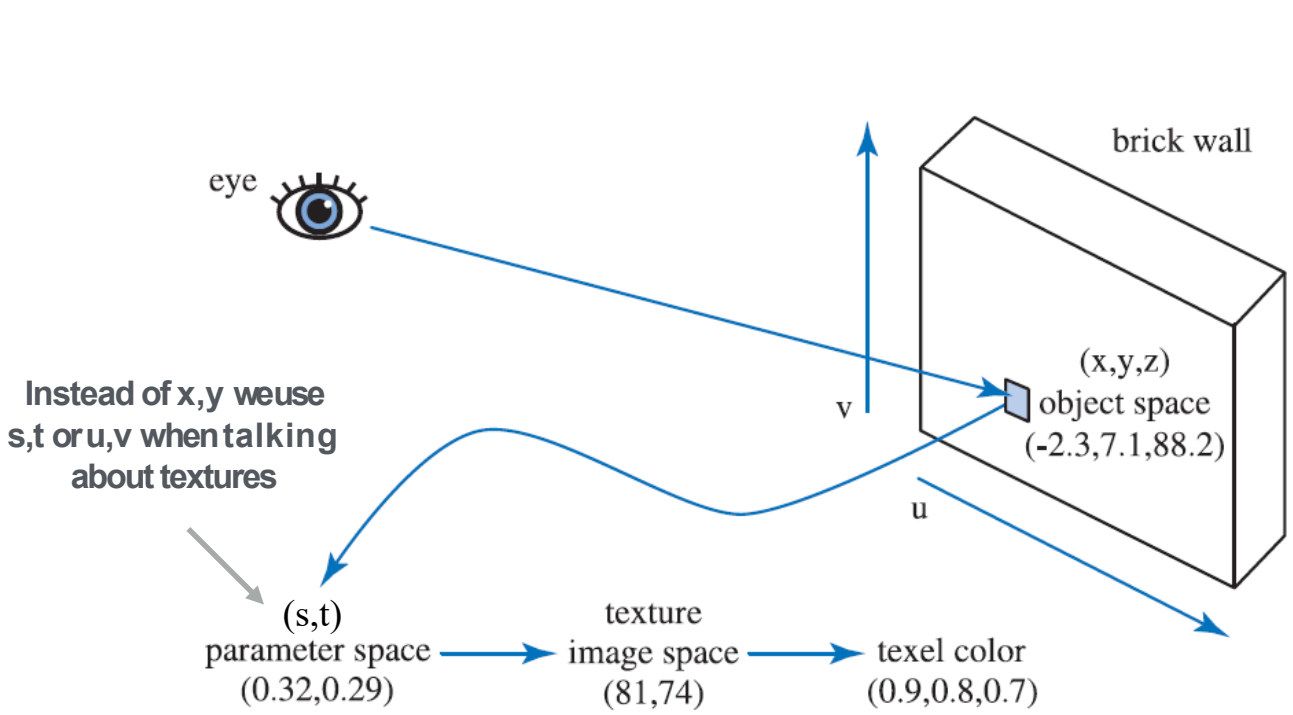
- **Texture coordinates** for every vertex needed





Texture Coordinates

- During rasterization the texture coordinate (in the range **[0,1]x[0,1]**) for every pixel is derived (just another vertex attribute)
- This is used to find the appropriate texture element (texel) in the texture image



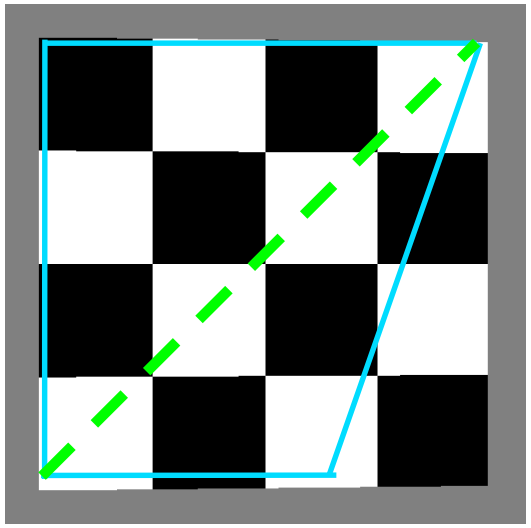
Texture Atlas

©UMBC

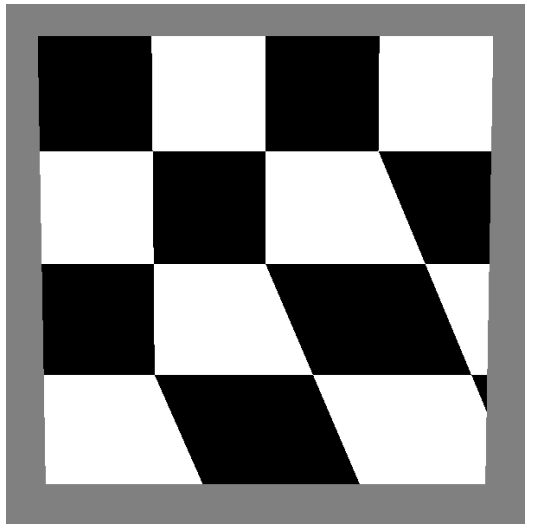
- Often one mapping per object for efficiency reasons

How to choose Texture Coordinates

- By hand / automatic projections
- Modeling tools with interactive placement
- Pay attention to deformation (OpenGL triangulates)



Texture section



Textured Polygon

Texture Definition

- Similar to VBOs
- Generate ID
  - `glGenTextures(GLsizei n,  GLuint * textureID);`
  - Generate `n` IDs and save them in `textureID`
- Activate texture
  - `glBindTexture(GLenum target, GLuint texture)`
  - target: Type of texture, e.g. `GL_TEXTURE_2D`
- Delete Texture
  - `glDeleteTextures( GLsizei n, const GLuint * textureID);`
  - Delete `n` Textures with the IDs `textureID`

Texture Definition

- Define Texture and upload data
  - `glTexImage{1,2,3}D(GLenum target, GLint level, GLint internalformat, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid * data)`
- `target`: `GL_TEXTURE_1D`, `GL_TEXTURE_2D`, `GL_TEXTURE_3D`, …
- `level`: 0 or the appropriate mipmap level (later)
- `internalFormat`: number of components and representation
  - E.g. `GL_ALPHA`, `GL_RGB`, `GL_RGBA`, `GL_RG16`

Texture Definition

- Define Texture and upload data
  - `glTexImage{1,2,3}D(GLenum target, GLint level, GLint internalformat, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid * data)`
- `width`: width of the texture in texels / pixels
- `height`: height of the texture in texels
  - only for 3D textures: `depth`: depth of the texture in texels
  - Good idea to use 2^m for width, height and depth (see Mipmapping)
- `border`: applies a border of size 0 or 1 (deprecated, must be 0 since OpenGL 3.1)

Texture Definition

- Define Texture and upload data
  - `glTexImage{1,2,3}D(GLenum target, GLint level, GLint internalformat, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid * data)`
- `format`: format of the data array, values per pixel, e.g. `GL_RGB, GL_BGR, GL_RGBA`, …
- `type`: data type used in the data array, e.g. `GL_UNSIGNED_BYTE, GL_FLOAT`, …
- `data`: texture data in main memory (no image loader)
- Example:
  `glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 512, 512, 0, GL_RGBA, GL_FLOAT, mydata);`

Texture Coordinates

- Similar to vertex buffer objects
  - Just another VBO

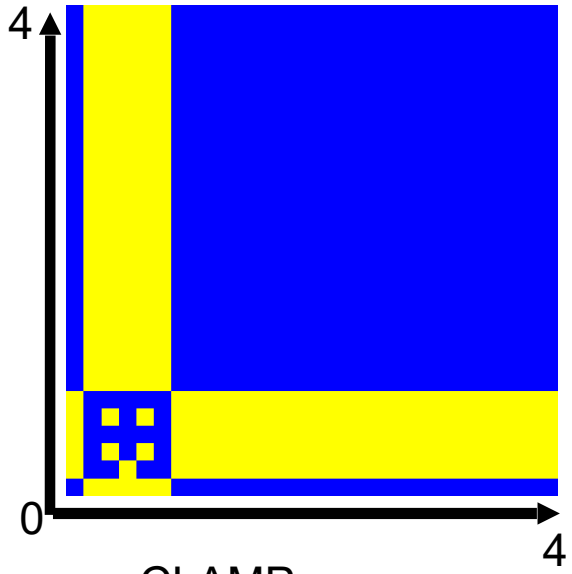
- Define the texture coordinate data
- `glVertexAttribPointer(index, size, type, normalized, stride, start);`

- Activate texture coordinates
- `glEnableVertexAttribArray(index);`

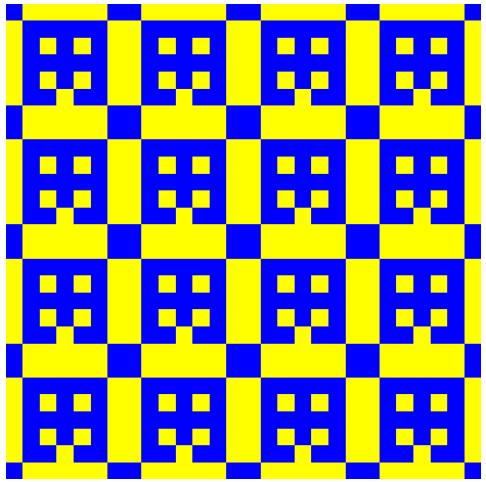
- index is also used in the shader later, e.g.
- `layout(location = index) vec2 texCoords;`

Texture Wrapping

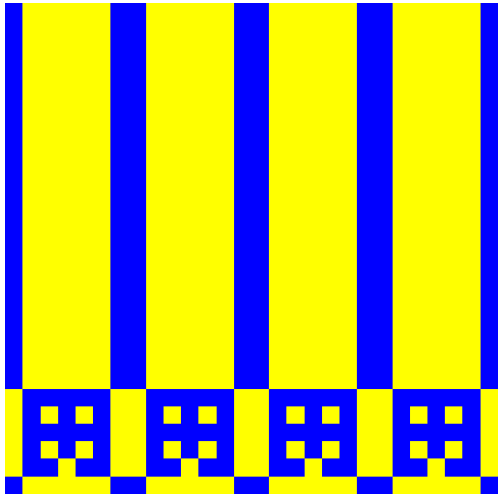
- What happens if texture coordinates larger than [0,1]x[0,1] are used?



CLAMP



REPEAT



REPEAT for s-direction

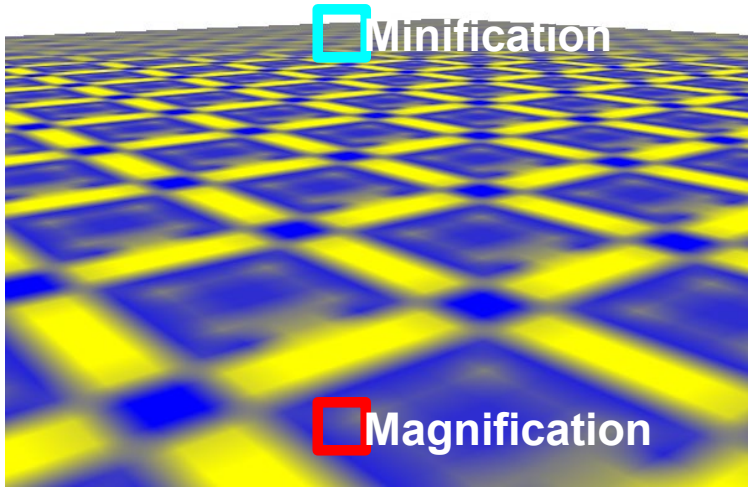
CLAMP for t-direction

Texture Wrapping

- What happens if texture coordinates larger than [0,1]x[0,1] are used?
- `glTexParameteri(GL_TEXTURE{123}D, name, value);`
- `name: GL_TEXTURE_WRAP_{STR}`
  - For the respective texture coordinate (s, t, r)
- `value = GL_CLAMP_TO_EDGE`: Values smaller than 0 or larger 1 are clamped
- `value = GL_REPEAT`: Values after decimal point are used

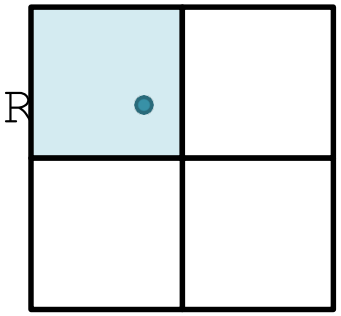
Texture Filtering

- **Texture Magnification**
  - One texel is mapped to more than one pixel
- **Texture Minification**
  - Many texels are mapped to the same pixel

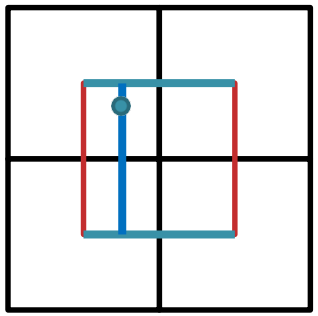


Filter Mode

- `glTexParameteri(GL_TEXTURE{123}D, type, mode);`
- Filter Types
  - `GL_TEXTURE_MIN_FILTER or GL_TEXTURE_MAG_FILTER`
- Filter Modes
  - GL_NEAREST
    - Nearest texel
  - GL_LINEAR
    - Linear average
- Mipmapping



Nearest Texel

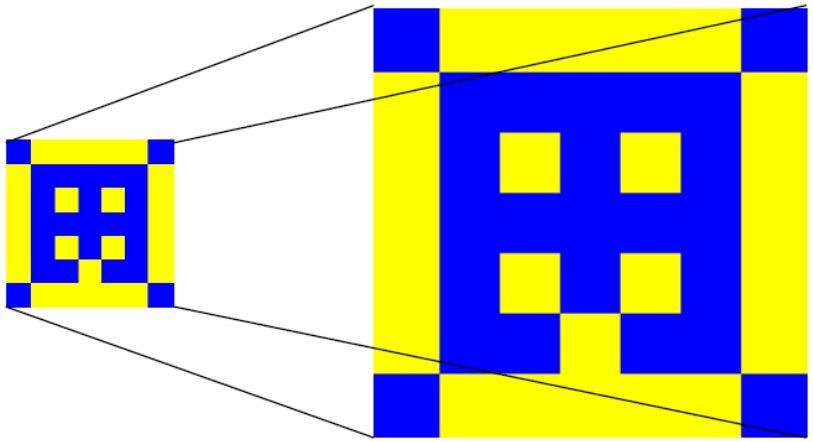


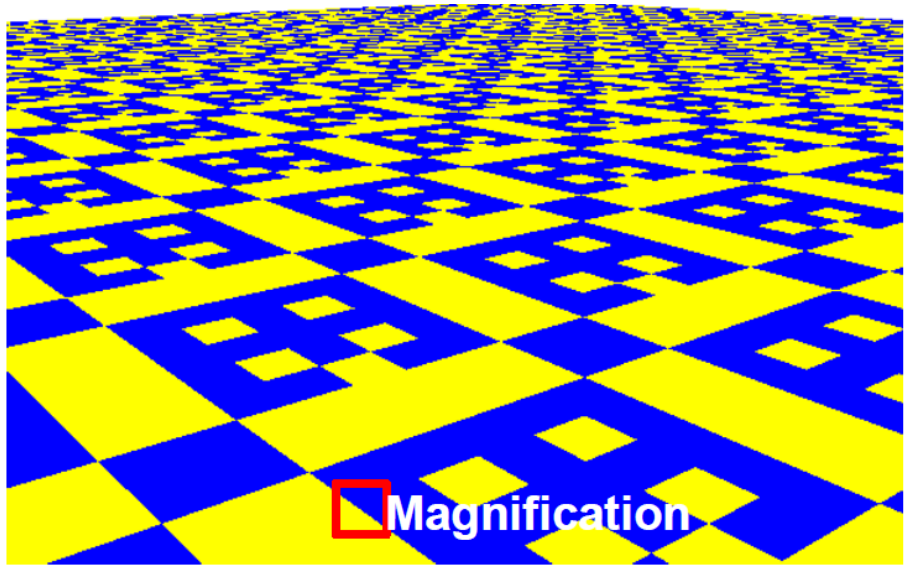
Linear average of the closest four texels

Texture Filtering

- Texture Magnification
  - `glTexParameteri(GL_TEXTURE{123}D, GL_TEXTURE_MAG_FILTER, value);`

- `value` = `GL_NEAREST`: Point filter, texels are replicated

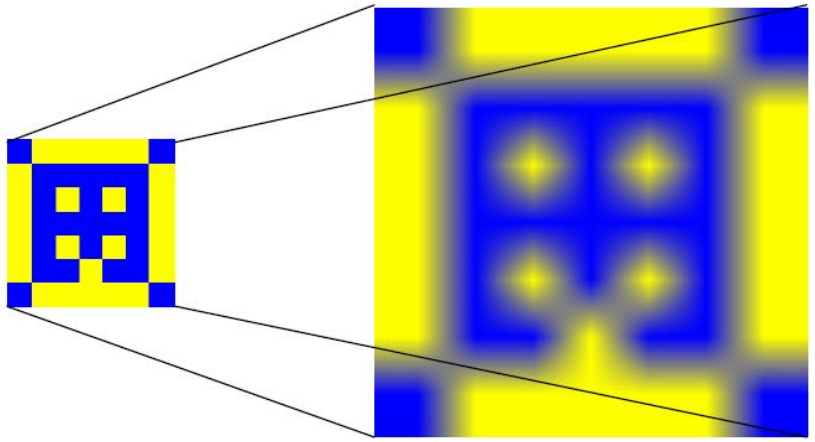


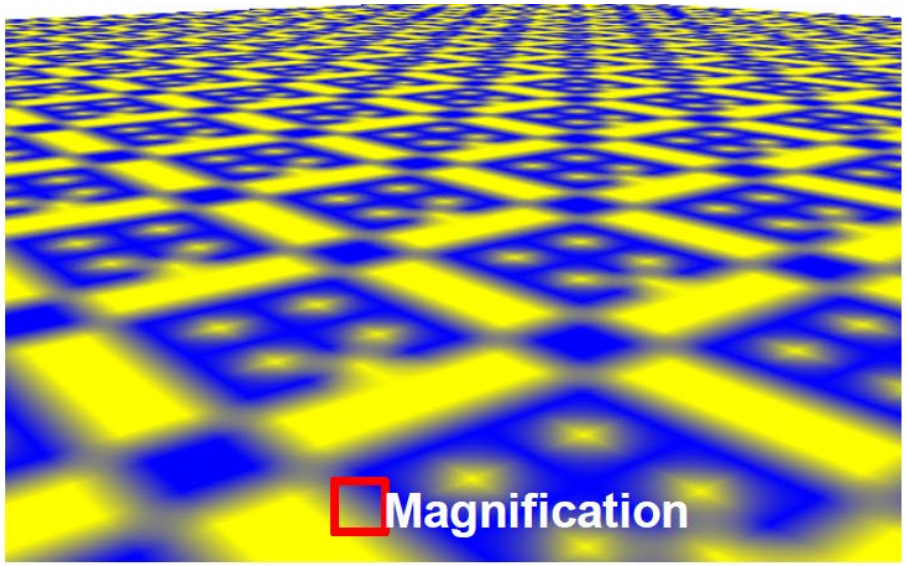


Texture Filtering

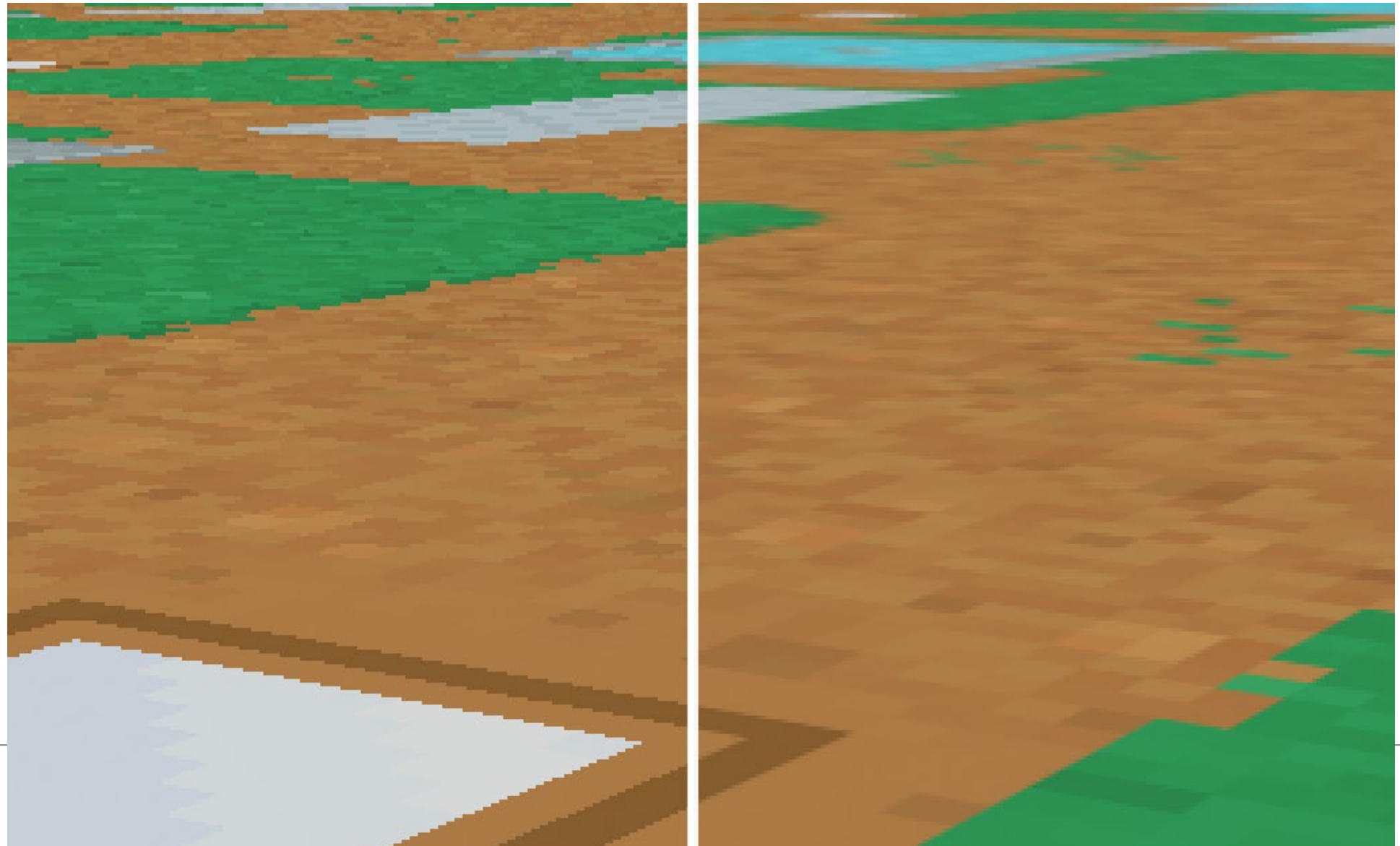
- Texture Magnification
  - `glTexParameteri(GL_TEXTURE{123}D, GL_TEXTURE_MAG_FILTER, value);`

- `value` = `GL_NEAREST`: Point filter, texels are replicated
- `value` = `GL_LINEAR`: Bilinear interpolation



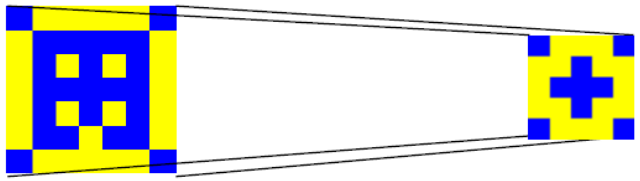


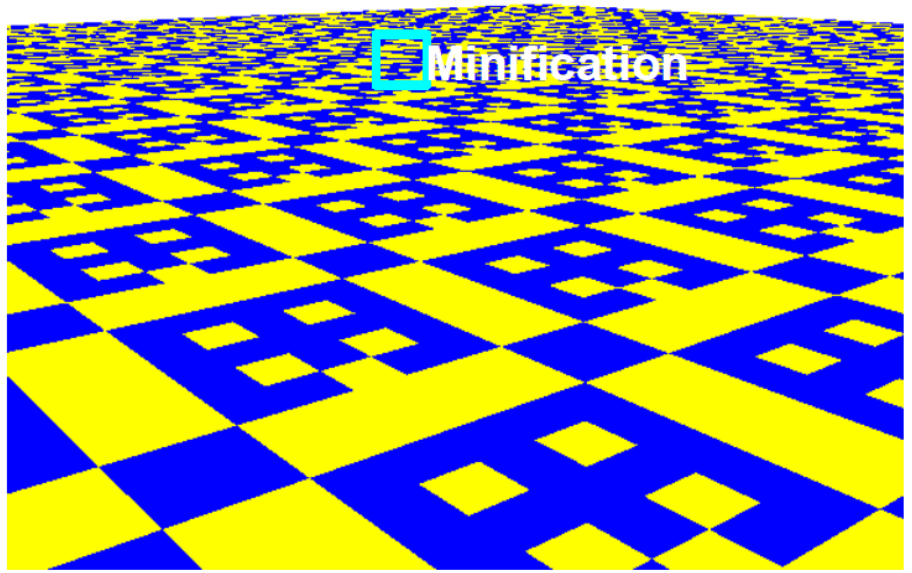
Example



Texture Filtering

- Texture Minification
  - `glTexParameteri(GL_TEXTURE{123}D, GL_TEXTURE_MIN_FILTER, value);`
- `value` = `GL_NEAREST`: Exact texture Coordinate is used

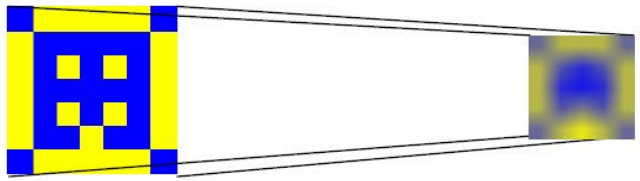


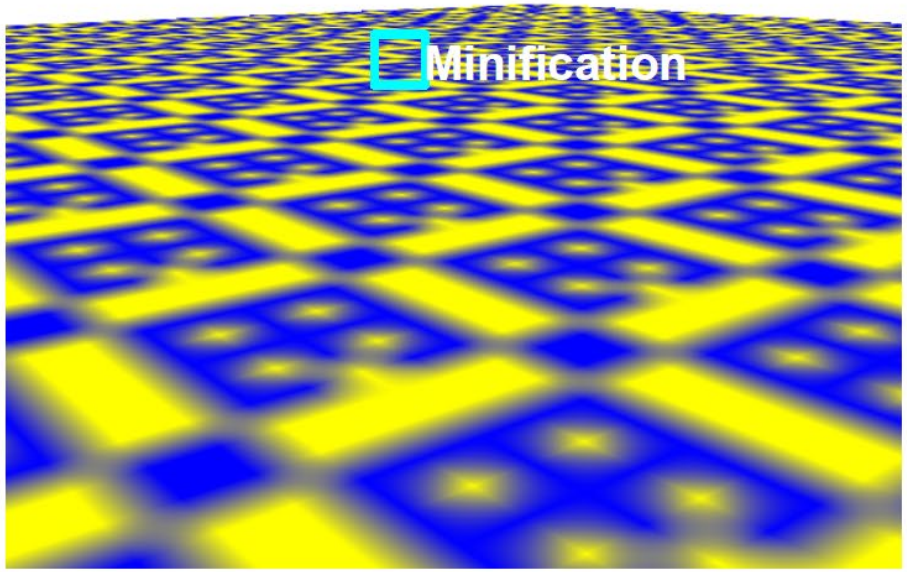


Texture Filtering

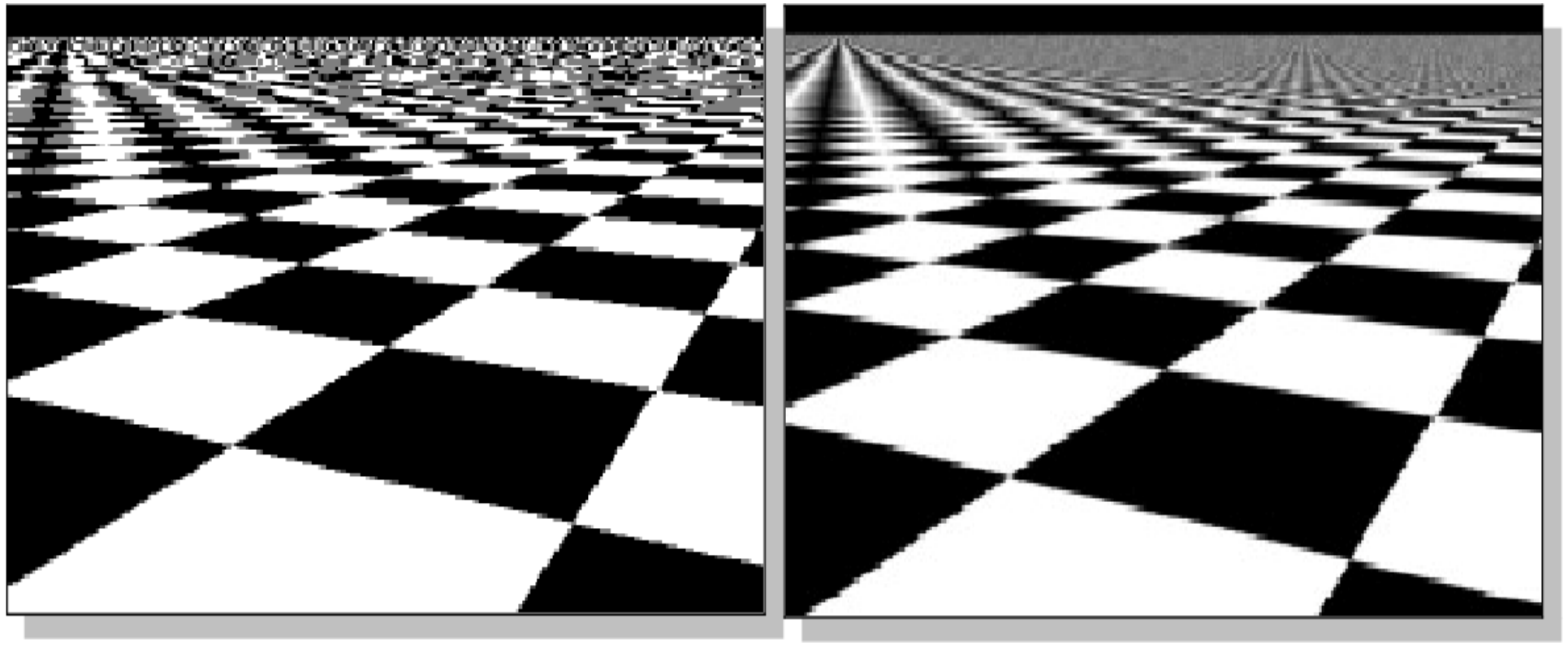
- Texture Minification
  - `glTexParameteri(GL_TEXTURE{123}D, GL_TEXTURE_MIN_FILTER, value);`

- `value = GL_NEAREST`: Exact texture Coordinate is used
- `value = GL_LINEAR`: Bilinear interpolation



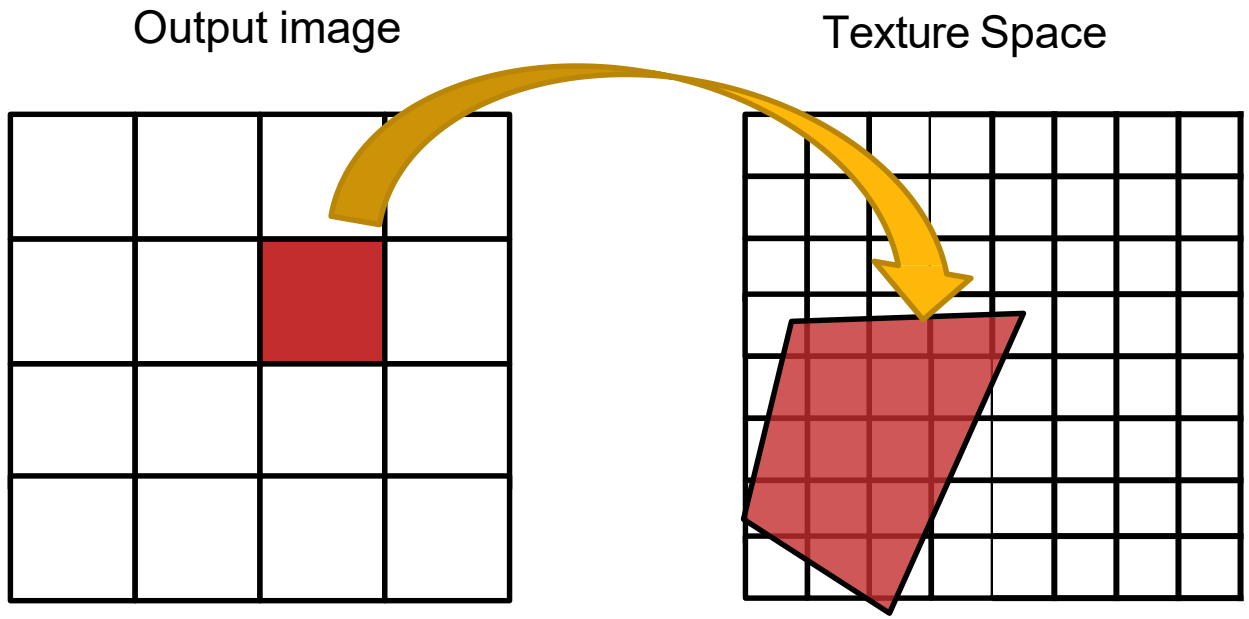


Example



Texture Filtering

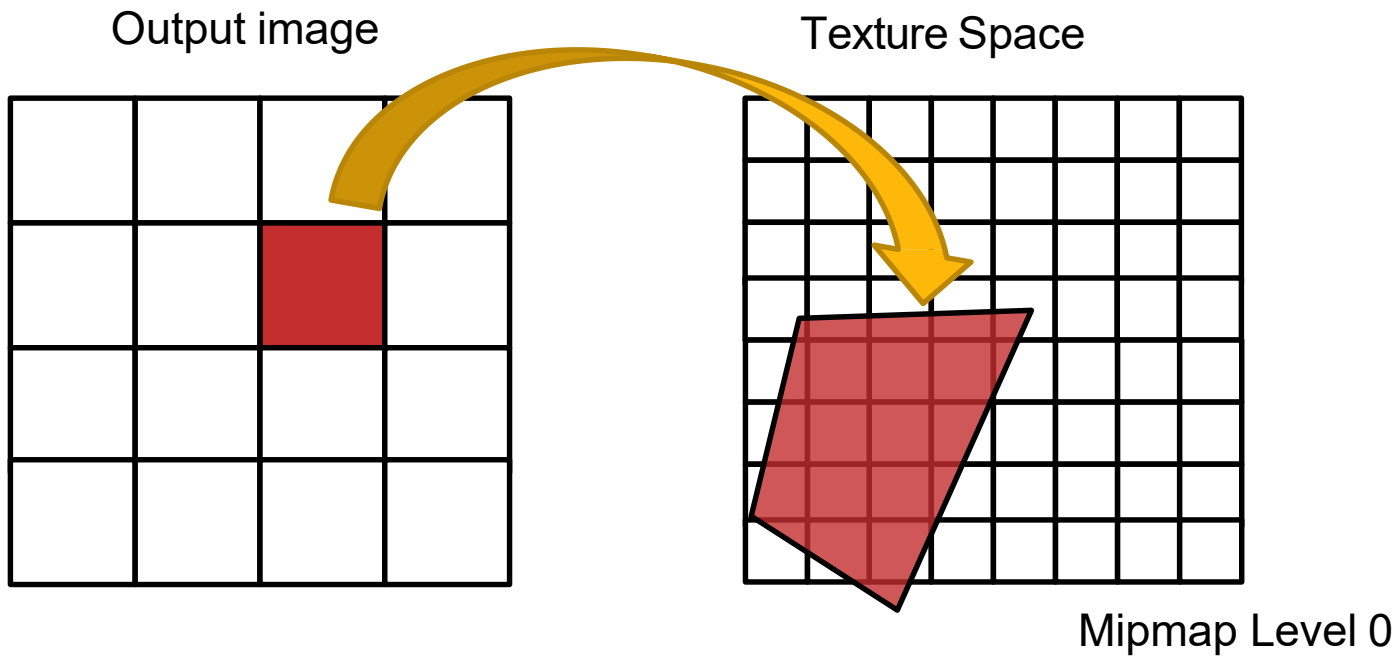
- Texture is a sampled {123}D function
- How to calculate texture color in output fragment?
- Needed is the normalized integral of the covered area!



- However, looking up all covered texels is computationally too expensive!

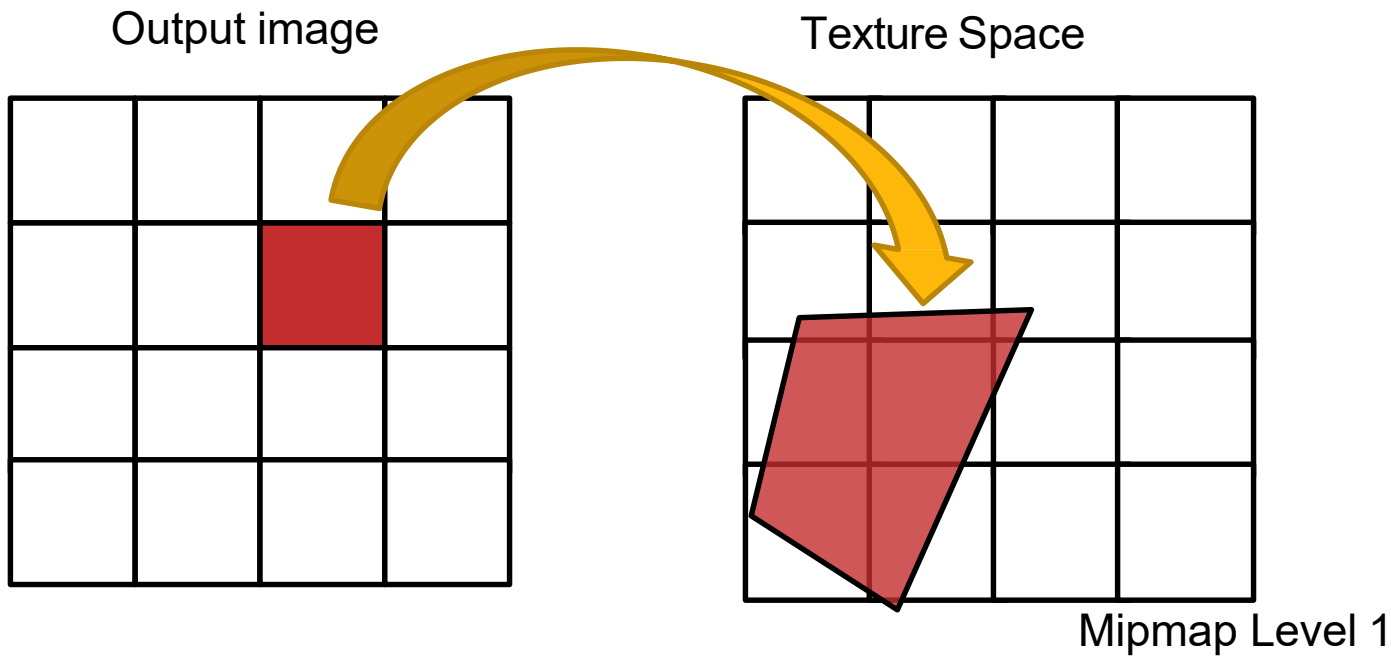
Mipmaps

- Approximate integral by averaged texels at different resolution level



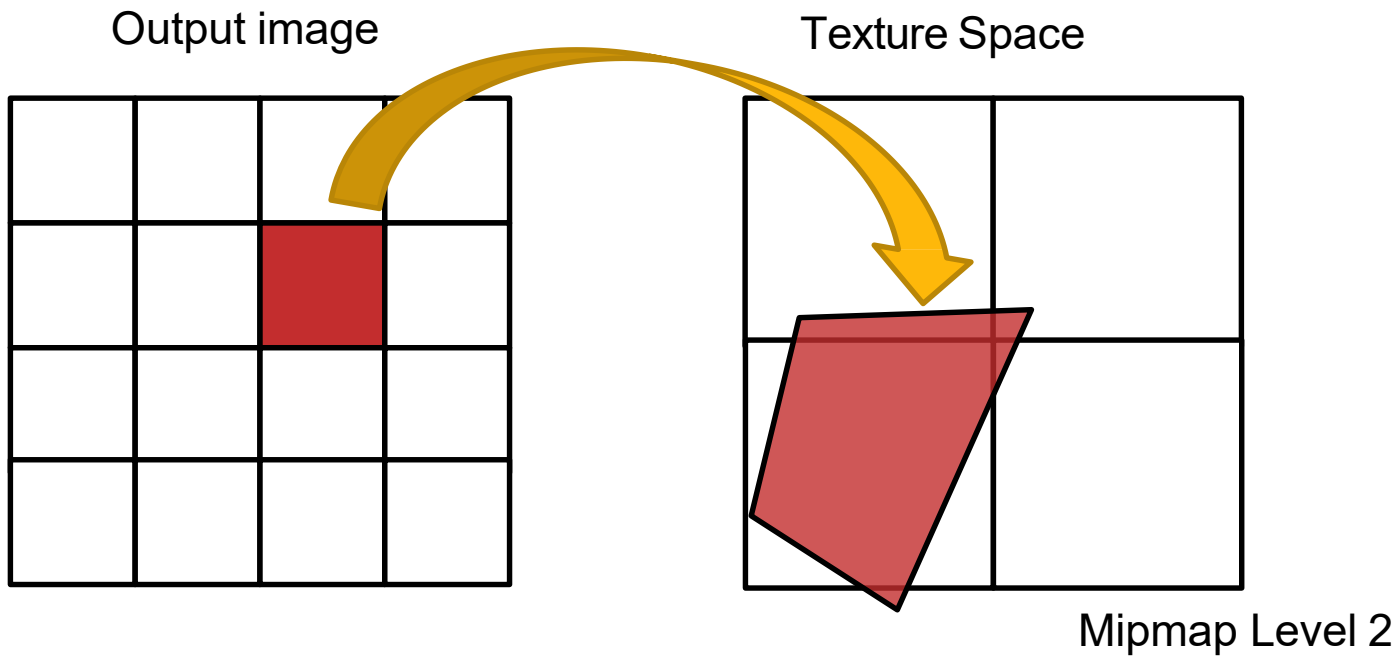
Mipmaps

- Approximate integral by averaged texels at different resolution level



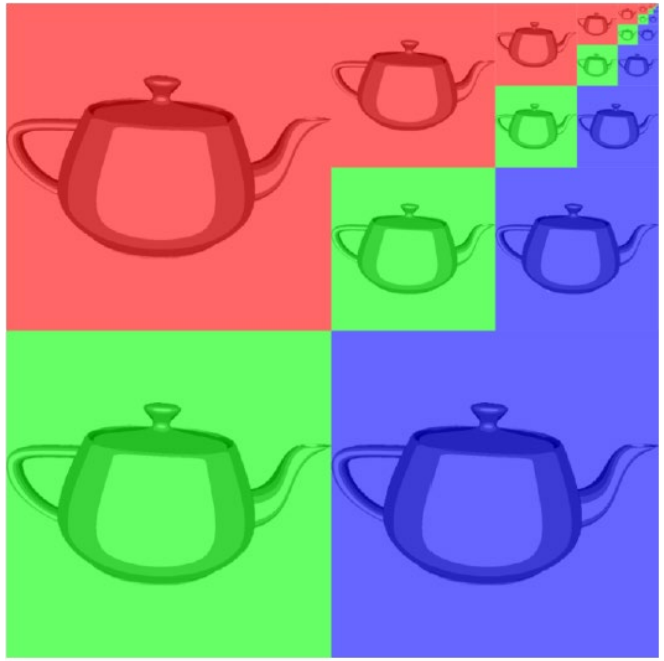
Mipmaps

- Approximate integral by averaged texels at different resolution level



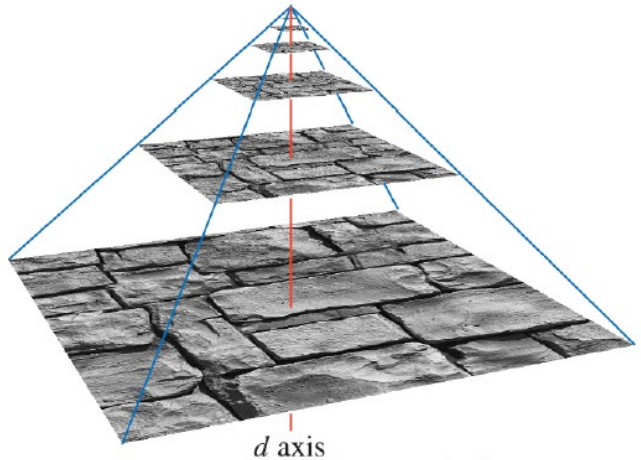
Mipmaps

- Texture precalculated in different resolutions
- 2^n image represented as 2^n+1 image internally
  - `glGenerateMipmap(GL_TEXTURE_{123}D);`
  - Generates a mipmap for the currently active texture
  - Or use level parameter of glTexImage{123}D(…)
- Mipmaps are always created from 2n x 2n textures
  - (scaled internally if necessary)



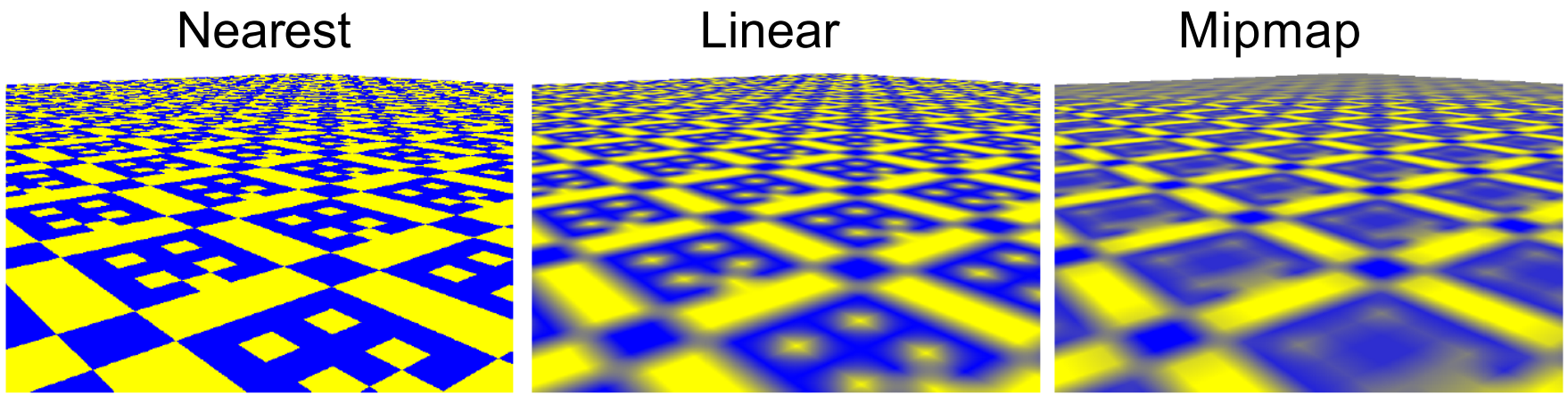
Mipmaps

- Texture Magnification
  - `GL_NEAREST`: Pointfilter
  - `GL_LINEAR`: Bilinear interpolation
- Texture Minification
  - `GL_{NEAREST,LINEAR}_MIPMAP_{NEAREST,LINEAR}`
- 1st part: Interpolation inside one level of mipmap
- 2nd part: Interpolation between two levels (trilinear interpolation)



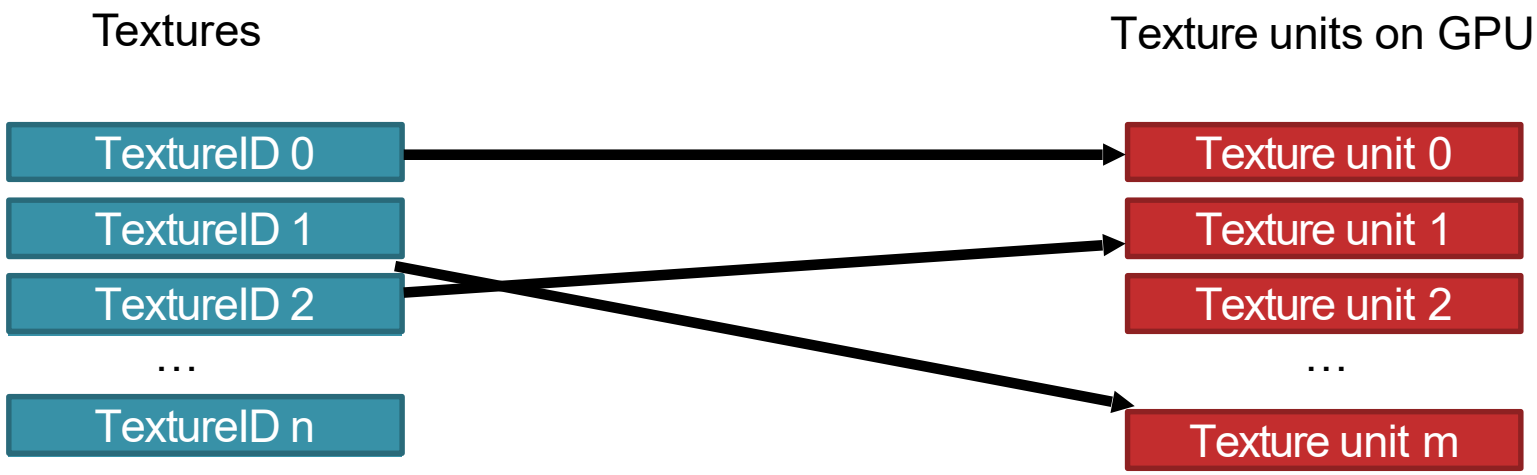
Mipmaps

- Benefit most visible when moving



Multitexturing

- Textures need to be bound to texture (mapping) units (TMU) on the GPU
- TMUs have their own texture engine to map textures onto 3D geometry
- In general max. num. textures >> available texture units (m~32)



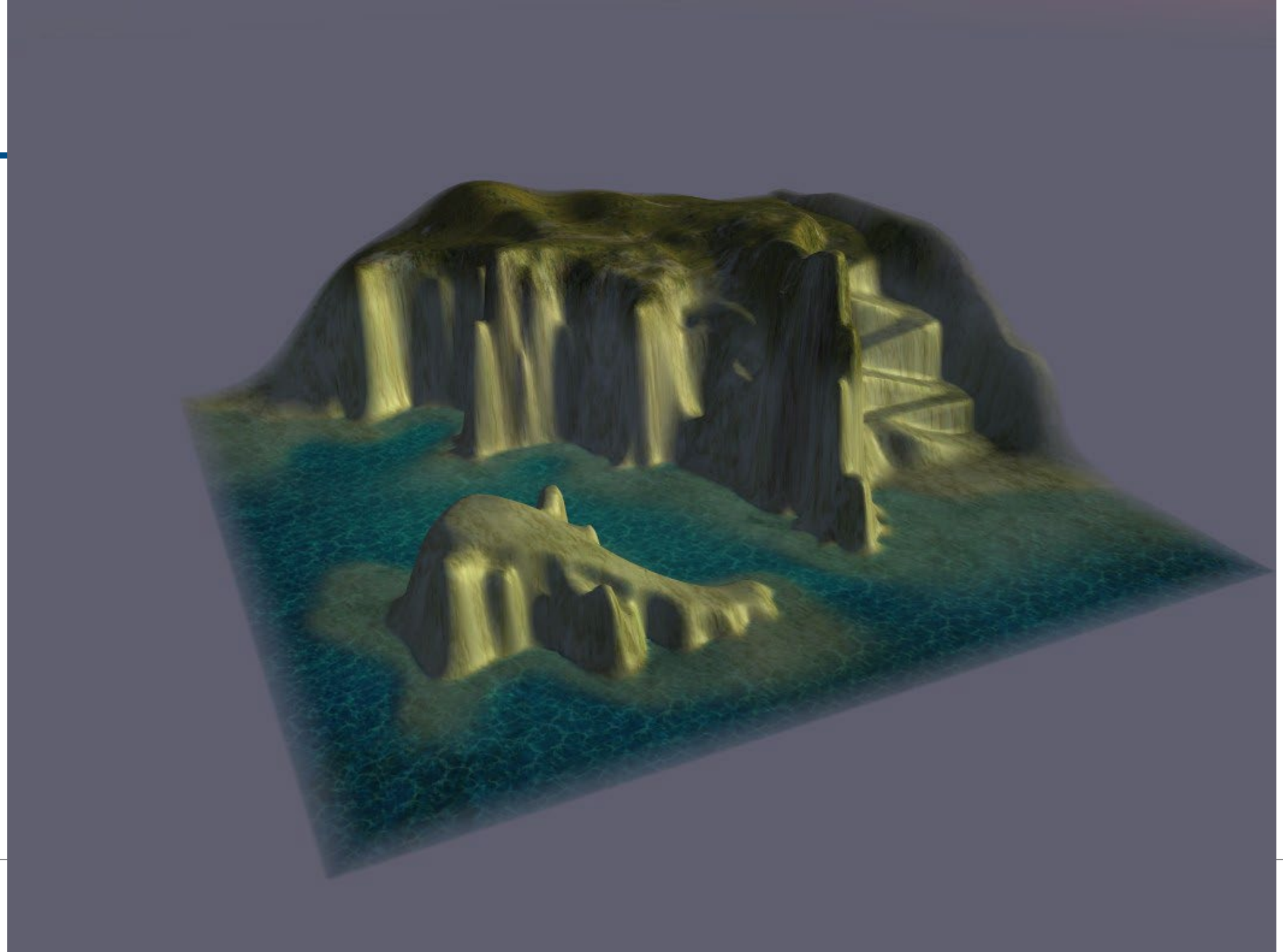
Multitexturing

- Textures need to be bound to texture units

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, myFirstTexture);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, mySecondTexture);
```

Or:

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, myFirstTexture);
glActiveTexture(GL_TEXTURE0 + 1);
glBindTexture(GL_TEXTURE_2D, mySecondTexture);
```



Textures and Shaders

- Texturing affects both vertex and fragment shader

Traditional use:

- **Vertex Shader**
  - (Compute) texture coordinates
  - pass values to fragment shader

- **Fragment Shader**
  - Query interpolated texture coordinates
  - Query color value of texture at that position
  - Use queried value, e.g. for shading

Texturing in the Vertex Shader

- Query texture coordinate
  - Given by vertex attribute, e.g.

```
layout(location = 2) in vec2 texcoords;
```

- Write texture coordinates to out variable

```
out vec2 tc;
      // do something with texcoords //
tc = texcoords;
```

Texturing in the Fragment Shader

- Textures are called samplers in the fragment shader, e.g.

```
uniform sampler1D tex0;
uniform sampler2D tex1;
```

- Access functions

```
texture(texID, texCoord)
```

- `texID`: name of the texture (i.e. of the sampler), e.g. `tex0`, …
- `texCoord`: vec{1,2,3} texture coordinate

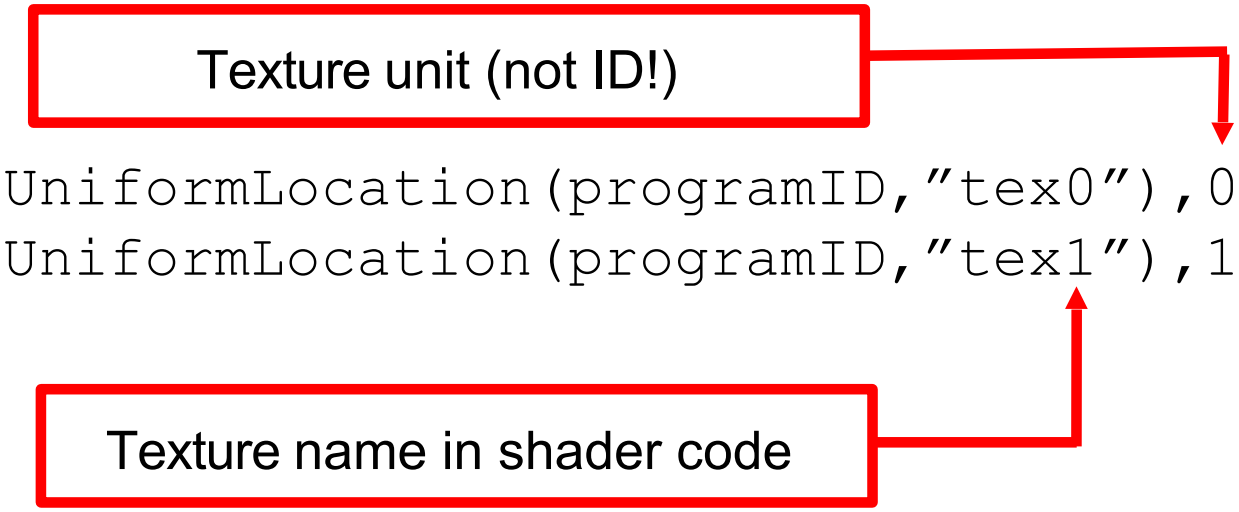
Example

Fragment Program

```
out vec4 color;

in vec2 tc0;
in vec2 tc1;

uniform sampler2D tex0;
uniform sampler2D tex1;

void main(){
      color = texture(tex0,tc0) +  texture(tex1,tc1);
}
```

Setting up textures for shader usage

- Load them as uniforms
  - `void glUniform1i(GLuint location, GLuint textureUnit);`



Texture Checklist

Create

- Create ID (`glGenTextures()`)
- Bind texture (`glBindTexture()`)
- Define texture and send data (`glTexImage2D()` with optional `glGenerateMipmap(GL_TEXTURE_2D)`)
- Set parameters(Wrapping and Filter) (`glTexParameteri()`)

Texture Checklist

Usage

- Activate texture unit (`glActiveTexture(GL_TEXTUREi)`)
- Bind texture to active unit (`glBindTexture()`)
- Assign texture coordinates in vertex shader
- (Input: `in variable`, Output: `out variable`)
- Use textures in fragment shader (`texture(texID,texCoord)`)