HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
–
Fakultät IV
Wirtschaft und
Informatik

Introduction to Computer Graphics and Animation
Lecture 4 of 4

Prof. Dr. Dennis Allerkamp
December 6, 2024

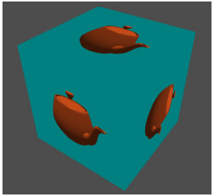
Summary

December 5, 2024

The following topics will be covered today:

- Texture mapping
- Texture sampling and filtering
- Multi-texturing
- Render to texture
- Environment mapping

After this day, participants will be able to render more interesting scenes as shown in the following examples.


Texture Mapping

Texture Mapping

- Fine structure *without* texture mapping:
  - Huge *increase in tessellation* needed
  - Number of polygons must be increased so that there is one vertex *per pixel*
  - Per vertex/pixel one color value
  - At *different resolutions* also adaptation of tessellation necessary
  - Disadvantages:
    - Not very efficient
    - Modeling effort very high
    - A lot of vertices that are sent to the pipeline
- Fine structure *with* texture mapping:
  - Texture color values only after the rasterization stage
  - Can be easily *combined with illumination*
  - Hardware accelerated on almost all graphics cards

Textures

- Photo that is glued onto a polygon mesh
- Attach 2D graphics to 3D surfaces
- “Breakthrough of photorealistic computer graphics”
- First approaches by Catmull (1974), Blinn & Newell (1976): James Blinn and Martin Newell. Texture and reflection in computer generated images. Comm. Of the ACM 19(10): 542-547, October 1976
- cf. wallpapers on walls, gift wrap, stickers on surprise egg models

Fig. 3. Hand sketched texture pattern: left-hand side shows texture pattern; right-hand side shows textured object.

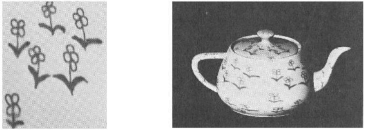
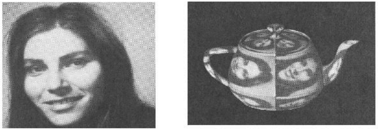


Fig. 4. Photographic texture pattern: left-hand side shows texture pattern; right-hand side shows textured object.



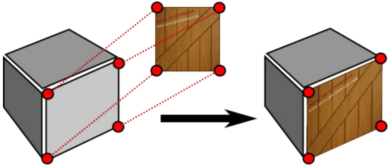
Blinn & Newell (1976)

Textures: Terms

- Adhesive process → *Texture Mapping*
- Adjustment to geometries → *Texture Coordinates*
- Scaling and Calculation of Texels → *Filtering*
- Filtering of textures in perspective → *Mip Mapping*
- Behavior at texture edges → *Texture Wrapping*

Texture Mapping in OpenGL

- A: Load image data / Create texture object
- B: Determine texture coordinates, draw geometry
- C: Adjust Shader



http://www.real3dtutorials.com/tut00005.php

Texture Mapping in OpenGL: Image Data / Texture Object

- *Load image data from an external image file*, often via a helper library (e.g. stb_image.h)

```
int width, height;
unsigned char *image = stbi_load("texture.jpg", &width, &height, &nrChannels, 0);
```

- *Generate texture object ID*, here 1 texture, and bind as 2D texture

```
GLuint texture;
glGenTextures(1, &texture);
glBindTexture(GL_TEXTURE_2D, texture);
```

- *Fill texture object with image data*

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

- *Delete image data* and *release binding*

```
stbi_image_free(image);
glBindTexture(GL_TEXTURE_2D, 0);
```

Texture Mapping in OpenGL: Texture → Geometry

- Create array with vertices and *texture coordinates*

```
GLfloat vertices[] = { //Pos., Colors, TextureCoor.
 0.5f, 0.5f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f, …
-0.5f, 0.5f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f, 1.0f };
```

- *VertexAttribPointer* for the texture coordinates

```
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8*sizeof(GLfloat), (GLvoid *) (6*sizeof(GLfloat)));
glEnableVertexAttribArray(2);
```

- Bind texture *before drawing*

```
GLuint loc = glGetUniformLocation(program, "ourTexture");
glUniform1i(loc, 13);
glActiveTexture(GL_TEXTURE13);
glBindTexture(GL_TEXTURE_2D, texture);
glBindVertexArray(VAO);
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
glBindVertexArray(0);
```

Texture Mapping in OpenGL: Adjust Shader

Vertex shader:

```
#version 330 core
layout (location=0) in vec3 position;
layout (location=1) in vec3 color;
layout (location=2) in vec2 texCoord;
out vec3 ourColor;
out vec2 tex;
void main()
{
  gl_Position = vec4(position, 1.0f);
  ourColor = color;
  tex = texCoord;
}
```

Pixel shader:

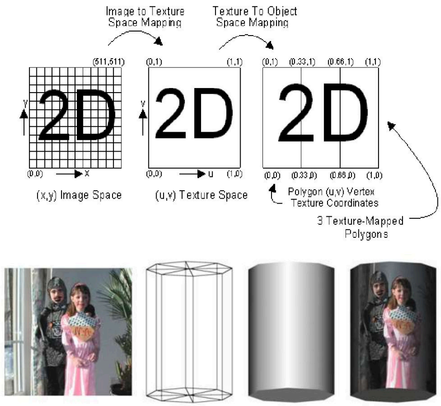
```
#version 330 core
in vec3 ourColor;
in vec2 tex;
out vec4 color;
uniform sampler2D ourTexture;
void main()
{
  color = texture(ourTexture, tex);
}
```

- *ourTexture* data type *sampler2D*
  - receives color value of the 2D texture at (s, t)

Texture Sampling and Filtering

Texture Mapping: Notation

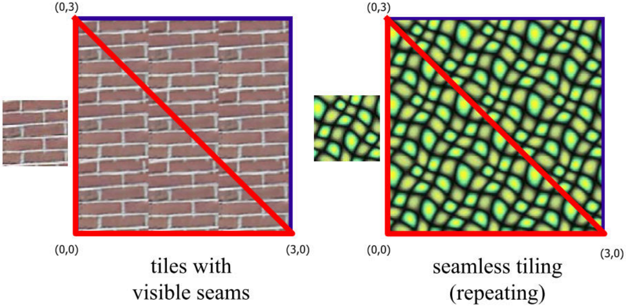
- Texture Mapping
  - *Mapping texture coordinates to surface coordinates*
  - (s, t) or (u, v) $\rightarrow$ (x, y, z)
  - 3 coordinate systems: 2D image, 2D texture, 3D object
- Image Space
  - 2D image with *Pixels (Picture Elements)* and pixel dimensions (e.g. 512 x 512 pixels)
- Texture Space
  - 2D image is stored in a "*Texture Map*"
  - The pixels of the Texture Map are called *Texel (Texture Element)*
  - Each 2D Texture Map has its own, normalized 2D coordinate system with *Texture Coordinates* (u, v / s, t)



Nischwitz 2011 S.267

Texture Mapping with Tiling

- How is a small texture repeatedly distributed on a polygon?
  → UV coordinates >1 and <0 lead to edge repetition, mirroring, tiling (configurable) → Example: s, t = (0, 0);(3, 0);(0, 3)
- *Seamless Tiling* = Textures are designed in such a way that no pattern transitions are visible



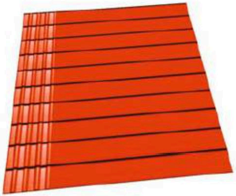
MIT Open Courseware

Texture Wrapping Parameters

Original-Textur

```
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_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
```



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

Texture Scrolling

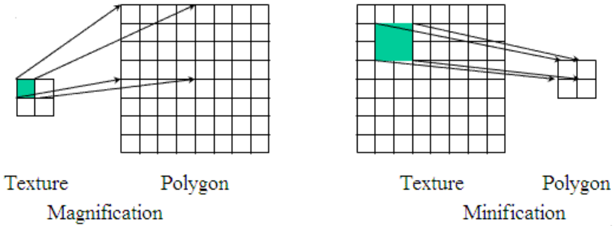
- *Animate* a texture over the *texture coordinates*
  - No change to the geometry!
  - E.g. road, fire, waterfall, conveyor belt, Skydome rotation
- Transfer of a *time-dependent variable* in the *Fragment Shader*

```
in vec2 TexCoord;
uniform sampler2D Texture0;
uniform float Time;
out vec4 out_FragColor;
void main() {
  out_FragColor = texture( Texture0, vec2(TexCoord.x + Time, TexCoord.y) );
}
```

- Further animations possible via transformation matrices
  - Zoom, Rotation, Shear, Morph, etc.

Texture Filter

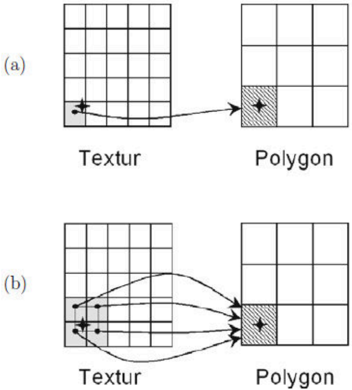
- So far: Texture is represented as a rectangular array of color values
  - Mapping is done on surfaces of objects using mapping on vertices
- Problems:
  - Distortion due to mapping from 2D to 3D coordinates
  - Perspective distortion due to projection/viewport transformations
  - A texel will therefore almost never be a pixel of the screen
- Depending on the transformations, the texture has to be enlarged (*Magnification*) or reduced (*Minification*)
- Question: Which texel color values should be used for certain pixels Since, it leads to the so-called "*sampling problem*"

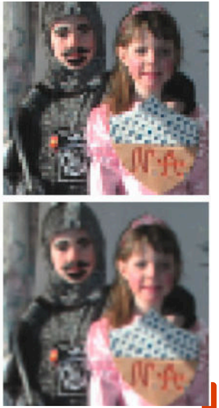


http://jcsites.juniata.edu/faculty/rhodes/graphics/texturemap.htm

Texture Filter

- `glTexParameteri()` determines which filter method should be used
- GL_NEAREST (a)
  - Color values of the texel that is closest to the pixel center
    - Very fast
    - Disadvantage: Aliasing effects
- GL_LINEAR (b)
  - Linear interpolation from the color values of a 2x2 texel array that is closest to the pixel center
  - Slower, but leads to smooth images

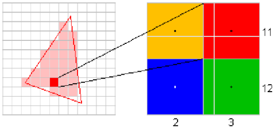


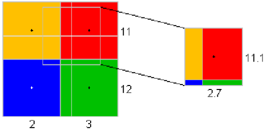


Bilinear Filtering

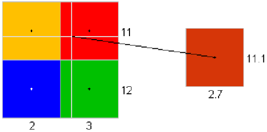
Determination of the resulting pixel color from 4 texels (xy = bilinear) by *bilinear interpolation*

1. Calculation of the *normalized areas* of the 4 texel components
2. Calculation with the color matrix of the four texel matrix
3. Result = linearly interpolated color





$$
\begin{array}{r}
\text{Rot}\rightarrow \\
\text{Grün}\rightarrow \\
\text{Blau}\rightarrow
\end{array}
\begin{bmatrix} 255 & 255 & 0 & 0 \\ 192 & 0 & 0 & 192 \\ 0 & 0 & 255 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0.21 \\ 0.63 \\ 0.03 \\ 0.07 \end{bmatrix} = \begin{bmatrix} 214.20 \\ 53.76 \\ 7.65 \end{bmatrix}
$$



MipMaps

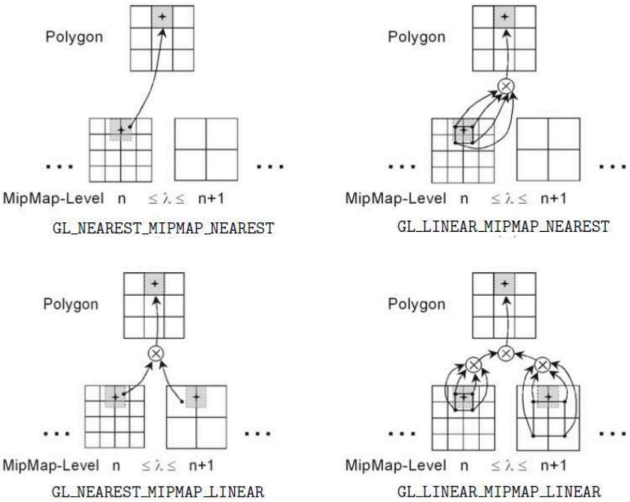
- Lowest MipMap level (0) = original texture
- Other levels are each ½ in edge length reduced
- All MipMap level textures belong to a texture in OpenGL (i.e. n textures at MipMap level n)
- Selection of the appropriate MipMap levels λ
  - Depending on the ratio ρ = Pixel/Texel,

$$\lambda = \log_2(\max(1/\rho_x, 1/\rho_y))$$

  - E.g., $\rho = 1/4 \rightarrow \lambda = 2$
- Disadvantage of MipMaps:
  - Memory requirement increases by 1/3
  - this is due to the limit of the geometric series $1/3 = 1/4 + 1/16 + 1/64 + ...$

Bild 13.6: Gauß-Pyramiden-Textur (MipMap): Die Original-Textur mit der höchsten Auflösung ist ganz links zu sehen (MipMap-Level 0). Die verkleinerten Varianten (MipMap-Levels 1, 2, 3, ...) nach rechts hin wurden zuerst tiefpass-gefiltert und dann in jeder Dimension um den Faktor 2 unterabgetastet. Stapelt man die immer kleineren MipMap-Levels übereinander, entsteht eine Art Pyramide (daher der Name).

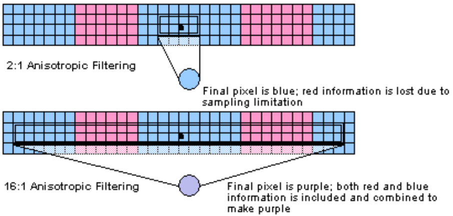
MipMapping Filtering Minification Examples



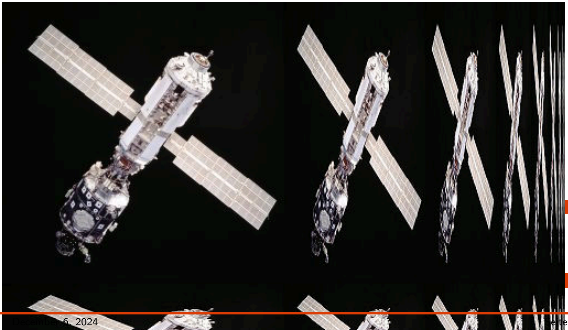
- Bilinear Filtering
- Trilinear Filtering

Anisotropic Filtering

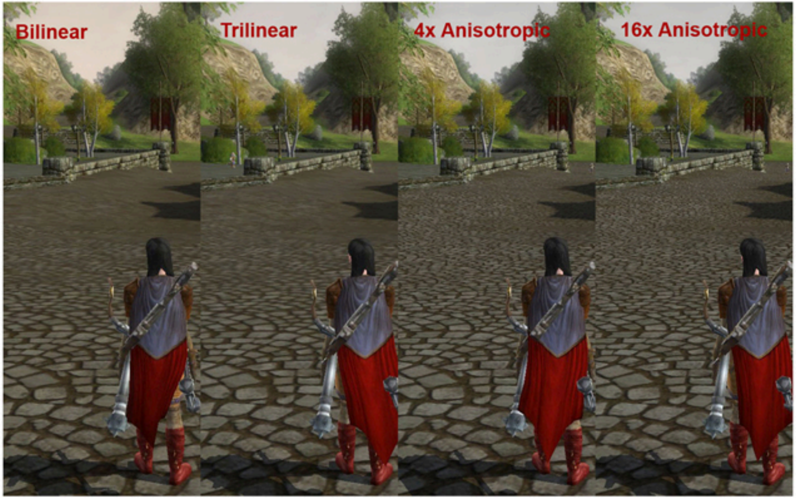
- Calculation of "non-square" pixel sets
- Factor gives maximum ratio of the sides
  - 2x e.g.: 2x4 pixels
  - 4x e.g.: 2x8 pixels
  - 8x e.g.: 2x16 pixels
  - 16x e.g.: 2x32 pixels
- Rectangle is selected depending on angle of view
- Similarly: RIP Mapping (bottom picture)



Source: ATI



MipMapping Filtering compared



Source: NVIDIA | http://www.geforce.com/whats-new/guides/aa-af-guide#1

Multitexturing in OpenGL

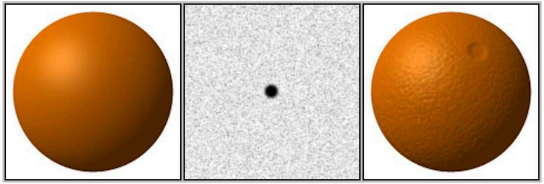
Multi-Texturing Examples

Figure 6.7 Two textures used in the multitexture example

Figure 6.8 Output of the simple multitexture example

OpenGL Redbook

Nischwitz, 2011



Von Bump-map-demo-smooth.png, Orange-bumpmap.png and Bump-map-demo-bumpy.png: Original uploader was Brion VIBBER at en.wikipediaLater version(s) were uploaded by McLoaf at en.wikipedia.derivative work: GDallimore (talk) - Bump-map-demo-smooth.png, Orange-bumpmap.png and Bump-map-demo-bumpy.png, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=11747953

Multitexturing - Procedure

1. Spezifikation der Texturen mit statischen Attributen und Binding
    - Bildquelle, Filtering, Wrapping

2. Generierung der Texturen, Lokalisierung im Shader, Festlegung der aktiven Textureinheit, Binden der Texturen

```
glGenTextures(); glGetUniformLocation(), glUniform1i();
ActiveTexture(); glBindTexture();
```

3. Übertragung der jeweiligen Textur-Koordinaten, z.B. interleaved im VAO

```
glGenBuffers(); glBindBuffer(); glBufferData();
glGenVertexArray; glGenVertexArray;
glVertexAttribPointer(Vertex-Koordinaten); glEnableVertexAttribArray(0);
glVertexAttribPointer(Vertex-Farben); glEnableVertexAttribArray(1);
glVertexAttribPointer(Textur-Koordinaten_Textur0);
glEnableVertexAttribArray(2);
glVertexAttribPointer(Textur-Koordinaten_Textur1);
glEnableVertexAttribArray(3);
```

4. Zeichnen

```
glDrawArrays(GL_TRIANGLE_FAN, 0, 4);
```

5. Beschreibung der Verrechnung der Texturen im Fragment Shader

Multitexturing– Passing Multiple Textures

- Es können mehrere Texturen als Aktiv gesetzt werden
  - Aktiviere TMU0 und binde Textur 0:

    ```
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, tex0);
    ```

  - Aktiviere TMU1 und binde Textur 1:

    ```
    glActiveTexture(GL_TEXTURE1);
    glBindTexture(GL_TEXTURE_2D, tex1);
    ```

- Texturen an Fragment Shader übergeben

  ```
  GLint tex0_uniform_loc = glGetUniformLocation(prog, "tex0");
  glUniform1i(tex0_uniform_loc, 0);
  ```

- Im Fragment-Shader:

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

`glUniform1i` ist erforderlich um auf `sampler2D` Datenstruktur zuzugreifen