Computergrafik

Matthias Zwicker
Universität Bern
Herbst 2016
Today

• Basic shader for texture mapping
• Texture coordinate assignment
• Antialiasing
• Fancy textures
Texture mapping

- Glue textures (images) onto surfaces
- Same triangles, much more interesting and detailed appearance
- Think of colors as reflectance coefficients
Texture mapping in OpenGL

- Initializing and loading texture requires series of OpenGL API calls
  - glPixelStorei
  - glGenTextures
  - glBindTexture
  - glTexImage2D
  - etc...

- Look up details when you need them

- Learn from example code, GLTexture.java

- Documentation [http://www.opengl.org/documentation/](http://www.opengl.org/documentation/)

[Open GL Programming Guide](http://www.glprogramming.com/red/)
Basic shaders for texturing

// Need to initialize texture using OpenGL API calls, which are
// implemented in GLTexture.java. Need to pass “uniform” parameters
// to shaders, as in GLRenderContext.java

// Vertex shader
uniform mat4 modelview;
uniform mat4 projection;
in vec2 texcoords;
in vec4 position;
out frag_texcoords;

void main()
{
    gl_Position = projection * modelview * position; // predefined output
    frag_texcoords = texcoords; // pass texture coords. to fragment shader
}

// Fragment shader
uniform sampler2D tex; // “tex” is reference to texture, set by host
in frag_texcoords;
out frag_color;
void main()
{
    frag_color = texture(tex, frag_texcoords); // “texture” is a GLSL fnct.
}
Today

- Basic shader for texture mapping
- Texture coordinate assignment
- Texture filtering
- Fancy textures
Texture coordinate assignment

• Surface parameterization
  - Mapping between 3D positions on surface and 2D texture coordinates
  - In practice, defined by texture coordinates of triangle vertices

• Various options to establish a parameterization
Parametric surfaces

http://en.wikipedia.org/wiki/Parametric_surface

- Surface position \( x, y, z \) given by three functions
  
  \[
  x = f_x(u, v) \quad y = f_y(u, v) \quad z = f_z(u, v)
  \]

  of parameters \( u, v \)

- Very common in computer aided design (CAD)
- Use \( (u, v) \) parameters as texture coordinates
- Later in class: Bézier surfaces
As a function of vertex positions

• In general, may compute $u$ and $v$ using two functions of vertex positions $x, y, z$

$$u = f_u(x, y, z), \quad v = f_v(x, y, z)$$

• How to define $f_u, f_v$?
Linear functions

- Simplest form: linear function (transformation) of vertex $x, y, z$ coordinates
- For example, orthographic transformation

\[
\begin{bmatrix}
u \\ v
\end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}
\]
Projective transformation

• Use perspective projection of $x, y, z$ coordinates

\[
\begin{bmatrix}
  u' \\
  v' \\
  w
\end{bmatrix} = M \begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix}, \quad u = u'/w, v = v'/w
\]

• Useful to achieve “fake” lighting effects
Spherical mapping

- Use, e.g., spherical coordinates for sphere
- Place object in sphere
- "shrink-wrap" sphere to object
  - Shoot ray from center of sphere through each vertex
  - Spherical coordinates of the ray are texture coordinates for vertex
Cylindrical mapping

- Similar as spherical mapping, but with cylinder
- Useful for faces
Skin mapping

- Techniques to unfold surface onto plane
  - Minimize “distortions”
  - Preserve area, angle
- Sophisticated math
- Functionality usually provided by 3D modeling tools (Maya, Blender, etc.)
Today

- Basic shader for texture mapping
- Texture coordinate assignment
- Anti-aliasing
- Fancy textures
What is going on here?
Aliasing

Sufficiently sampled, no aliasing

http://en.wikipedia.org/wiki/Aliasing
Aliasing

Sufficiently sampled, no aliasing

Insufficiently sampled, aliasing

High frequencies in the input appear as low frequencies in the sampled signal

http://en.wikipedia.org/wiki/Aliasing
Antialiasing: intuition

- Pixel may cover large area on triangle in camera space
Antialiasing: intuition

- Pixel may cover large area on triangle in camera space
- Corresponds to many texels in texture space
- Should compute “average” of texels over pixel area

![Diagram showing image plane, camera space, and texture space with pixel area and texels.]
Antialiasing: the math

- **Pixels are samples, not little squares**
  [http://alvyray.com/Memos/CG/Microsoft/6_pixel.pdf](http://alvyray.com/Memos/CG/Microsoft/6_pixel.pdf)

- Use **frequency analysis** to explain sampling artifacts
  - **Fourier transforms**

- If you are interested
  - Heckbert, “Fundamentals of texture mapping”
  - Glassner, “Principles of digital image synthesis”
Antialiasing

• Can be achieved by „averaging“ texels over pixel area

• Problems, disadvantages?
Antialiasing using mipmaps

• Averaging over texels during rendering is expensive
  - Many texels as objects get smaller
  - Large memory access and computation cost

• Precompute and store “averaged” (filtered) textures
  - MIP stands for “multum in parvo” (Williams 1983)

• Practical solution to aliasing problem
  - Fast and simple
  - Available in OpenGL, implemented in GPUs
  - Reasonable quality
Mipmaps

Before rendering

• **Precompute** and store several filtered versions of textures (mipmaps)

• Filtering performs “local averaging”
  - Simplest: **box filter**, uniform weighting in a square window; replace each pixel by average of pixels in its neighborhood

• Use higher quality filter to avoid aliasing

• Precompute several filtered textures with different sizes of filtering window
Mipmaps

Double the size of the filtering window from level to level!
Computing mipmaps

- Filtering implemented using convolution
  

- Input function $f$, convolution kernel (filter) $g$

- Continuous formulation

  $$(f * g)(t) \overset{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau) \cdot g(t - \tau) \, d\tau$$

- Discrete formulation

  $$(f * g)[n] \overset{\text{def}}{=} \sum_{m=-\infty}^{\infty} f[m] \cdot g[n - m]$$

- Two-dimensional convolution is a straightforward extension
Computing mipmaps

• Filtered textures are blurry
  - Reduce resolution by factor 2 successively without losing information

• Increases memory cost only by 1/3
  - \( \frac{1}{3} = \frac{1}{4} + \frac{1}{16} + \frac{1}{64} + \ldots \)

• Width, height of texture needs to be power of two
Example

- Resolutions 512x512, 256x256, 128x128, 64x64, 32x32

“multum in parvo”
Example

- 1 texel in level 4 is an average of $4^4 = 256$ texels in level 0

“multum in parvo”
Rendering with mipmaps

- Interpolate texture coordinate of each pixel as before
- Compute approximate size of pixel in texture space
- Look-up color in nearest mipmap
  - E.g., if pixel corresponds to 10x10 texels use mip-map level 3
  - Use nearest neighbor or bilinear interpolation as before
Mipmapping

- Image plane
- Camera space
- Texture space

- Pixel area
- Texels

- Mip-map level 0
- Mip-map level 1
- Mip-map level 2
- Mip-map level 3
Size of a pixel in texture space

- Given by partial derivatives of mapping $u(x, y), v(x, y)$
Nearest mipmap, nearest neighbor

- Visible transition between mipmap levels
Nearest mipmap, bilinear

- Visible transition between mipmap levels
Trilinear mipmapping

http://en.wikipedia.org/wiki/Trilinear_filtering

- Use two nearest mipmap levels
  - E.g., if pixel corresponds to 10x10 texels, use mipmap level 3 and 4

- Perform bilinear interpolation in both mipmaps

- Linearly blend between the results

- Requires access to 8 texels for each pixel

- Standard method, supported by hardware with no performance penalty
Trilinear mipmapping

• Smooth transition between mipmap levels
Note on OpenGL

- Distinguishes between minification and magnification
  - Minification: a texel is smaller than a pixel
  - Magnification: a texel is larger than a pixel
  - Minification, magnification may vary across pixels of individual triangles

- OpenGL allows you to specify different interpolation techniques separately
  - glTexParameteri
Are we satisfied?

Trilinear mipmapping
Mipmapping limitations

- Mipmap texels always represent square areas
- Pixel area is not always square in texture space
- Mipmapping makes trade-off between aliasing and blurriness

A circular pixel is back-projected to an ellipse
Anisotropic texture filtering

https://en.wikipedia.org/wiki/Anisotropic_filtering

- Average texture over elliptical area
  - Higher quality than trilinear mip-mapping
  - More expensive

- Anisotropic filtering in hardware
  - Take several bilinear probes approximating the ellipse
  - Reduces rendering performance on current GPUs
Comparison

• Animation
Today

• Basic shader for texture mapping
• Texture coordinate assignment
• Antialiasing
• Fancy textures
Fancy textures

- Textures most commonly used to modulate ambient and diffuse reflection
- E.g., diffuse fragment shader with texture

```glsl
in vec3 normal, lightstrength, lightDir;
uniform sampler2D tex;
out fragColor;

void main()
{
    fragColor = lightstrength * 
    max(dot(normal, normalize(lightDir)),0.0) * 
    texture(tex, texcoords);  // texture as diffuse coeff.
}
```

- Other applications?
Bump mapping

- Texture map contains normal perturbations
- No modification of geometry
  - Visible mostly at silhouettes
- Render using per-pixel shading, fragment shader
  - Normal in each pixel is modified using texture map
    (later in course)
Displacement mapping

- Texture map contains local height field
- Modifies geometry
  - Correct silhouettes, shadows
- Requires complicated fragment shader
Other effects

Multi-texturing

• Several layers of textures for different effects
  - Scratches, dents, rust, ...
  - Illumination textures

Animated textures

• Raindrops

• A TV screen, projector in a 3D scene
Next time

- Scene graphs and hierarchies