Texture Mapping

Computer Graphics, 2014
Lecture 9

Johan Nysjö
Centre for Image analysis
Swedish University of Agricultural Sciences
Uppsala University
What we have rendered so far:

Looks good, but how do we add more details (and colors)?
Texture mapping

- Allows us to add extra details without adding extra geometry (polygons)

Original mesh

Same mesh, but with Blinn-Phong shading + diffuse, gloss, and normalmap textures

Model and texture source: http://www.turbosquid.com
Techniques covered in this lecture

- Basic texture mapping
- Displacement mapping
- Bump and normal mapping
- Environment mapping
- Procedural textures
- Billboardding
Texture mapping

Original model

Texture mapped model

Image-based 2D texture drawn for the unwrapped polygon mesh

Image source: http://www.realtimerendering.com
Textures

- In most real-time rendering applications (e.g., games), textures are image-based 2D arrays of data.
- Can be created from, e.g., photos or hand-drawn images.
- 1D or 3D textures are also used in some applications.
Texture elements

• The elements in a texture are called **texels** (texture pixels) and can hold many different types of information, e.g.,
  - Color (most common)
  - Normals
  - Opacity
  - Intensity
  - Gloss
  - Height
  - Ambient occlusion

Different textures of a brick wall
Texture coordinates

- Texture coordinates range from 0 to 1 and are typically denoted $s$, $t$, and $p$, where
  - $s$ is used for 1D textures
  - $(s,t)$ are used for 2D textures
  - $(s,t,p)$ are used for 3D textures

Texture space for 2D texture maps
Assigning 2D texture coordinates to 3D models

- Each vertex in the rendered model must be assigned a texture coordinate.
- Can be done manually when defining simple geometric objects like planes, cubes, spheres, etc.
- For more complex models, texture coordinates can be assigned semi-automatically by unwrapping the mesh on a 2D grid in a 3D modelling software like Blender. Usually done by the 3D artist who created the model.
- Also possible to use projector functions to generate texture coordinates automatically.
Mesh unwrapping

Original mesh with diffuse shading

Same mesh with diffuse texture

Diffuse texture displayed on the unwrapped mesh
Projector functions

Image source: http://www.realtimerendering.com
Specifying texture coordinates in OpenGL

• Old fixed function OpenGL:
  - `glMultiTexCoord` – sets the current texture coordinates for a vertex

• Modern shader-based OpenGL:
  - Provide texture coordinates as vertex attributes
Mapping a 2D texture on a sphere using longitude/latitude as texcoords
Multiple textures can be applied

Sphere rendered with four different textures
Another multi-texture example
Loading texture images from file

• OpenGL does not provide any function for loading images, but there are plenty of third-party image libraries that can do this for you:
  - LodePNG (see example on next slide)
  - libpng
  - libjpeg
  - FreelImage (supports most popular image formats)
  - SOIL (reads BMP, JPEG, PNG, TGA, DDS, etc)
  - DevIL
Loading a PNG image with the LodePNG library

- LodePNG consists of two files: lodepng.cpp and lodepng.h.
- Include these files in your project/code base, and load the PNG image as

```cpp
#include "lodepng.h"
...
std::string filename = "diffuse.png";

std::vector<unsigned char> data;
unsigned width, height;
unsigned error = lodepng::decode(data, width, height, filename);

if (error != 0) {
    std::cout << "Error: " << lodepng_error_text(error) << std::endl;
    exit(EXIT_FAILURE);
}
...
```
Creating an OpenGL 2D texture from the loaded image

// Generate a name for the texture
GLuint texture;
glGenTextures(1, &texture);

// Bind the texture to the target GL_TEXTURE_2D
glBindTexture(GL_TEXTURE_2D, texture);

// Set sampler parameters
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

// Upload the image data into the texture
glTexImage2D(GL_TEXTURE_2D, 0, // target, level of detail
    GL_RGBA8, // internal format
    width, height, 0, // width, height, border
    GL_RGBA, GL_UNSIGNED_BYTE, // external format, type
    &(data[0])); // pixels

glBindTexture(GL_TEXTURE_2D, 0); // unbind the texture
Activating the texture

- At rendering, you need to select the active texture unit (GL_TEXTURE0, GL_TEXTURE1, etc), bind your texture to this unit*, and assign the texture unit number to a uniform sampler variable in the shader. Example:

```cpp
// Select texture unit 0 as active texture unit
glActiveTexture(GL_TEXTURE0);

// Bind your texture to this unit
glBindTexture(GL_TEXTURE_2D, texture);

// Assign the texture unit number to a uniform sampler variable
// (you can call this variable whatever you want)
program.setUniform1i("u_texture0", 0);

// Draw
...
```

*If you have multiple textures, you can bind the first texture to GL_TEXTURE0, the second to GL_TEXTURE1, and so on.
Accessing the 2D texture in a fragment shader

- Pass the texture coordinate of each vertex as a varying `vec2` variable (e.g., `v_texcoord`) from the vertex shader to the fragment shader.

- In the fragment shader, declare the uniform sampler variable (texture unit) as
  
  ```
  uniform sampler2D u_texture0;
  ```

  and access the 2D texture as
  
  ```
  vec4 color = texture(u_texture0, v_texcoord);
  ```
Links to some useful documentation

- glGenTextures
- glBindTexture
- glGetParameter
- glTexImage1D, glTexImage2D, glTexImage3D
- glActiveTexture
- glDeleteTexture
Texture coordinate range

- Texture coordinates range from 0.0 to 1.0
- What happens if we go outside this range?
Corresponder functions (wrapping)

- OpenGL uses so-called corresponder functions to determine how the image should be wrapped when the texture coordinates lie outside the [0,1) range:
  - GL_CLAMP_TO_EDGE
  - GL_CLAMP_TO_BORDER
  - GL_REPEAT
  - GL_MIRRORED_REPEAT
GL_CLAMP_TO_EDGE
GL_CLAMP_TO_BORDER
GL_MIRRORED_REPEAT
Texture filtering

- The resolution of the texture map rarely matches the resolution of the viewport
- We can have many texels within one pixel (minification)
- or many pixels representing one texel (magnification)
- OpenGL handles this via filtering operations:
  - GL_NEAREST
  - GL_LINEAR
  - GL_NEAREST_MIPMAP, GL_NEAREST_MIPMAP_LINEAR
  - GL_LINEAR_MIPMAP, GL_LINEAR_MIPMAP_LINEAR
Nearest-neighbor (GL_NEAREST) filtering

Original image

Magnification
Bilinear (GL_LINEAR) filtering

Original image

Magnification
Bicubic filtering*

*Not available in OpenGL, but can be implemented in fragment shaders
Texture aliasing artifacts

- The mismatch between texture resolution and viewport resolution can lead to visible **aliasing** or "jaggies" artifacts.
Antialiasing and Mipmapping

- Aliasing artifacts can be efficiently suppressed with a technique called **mipmapping**
- The original texture is filtered down repeatedly into smaller images, often called a **mipmap chain**
- During rendering, OpenGL selects the **mipmap level** that best matches the viewport resolution

Image source: http://www.realtimerendering.com
Mipmap linear filtering

Linear filtering

Mipmap linear filtering (less aliasing!)

Image source: Edward Angel
Mipmaps

- Mipmaps can be generated automatically in OpenGL with the function

  ```c
  void glGenerateMipmap(GLenum target);
  ```

  where `target` is one of the symbolic constants `GL_TEXTURE_2D` or `GL_TEXTURE_CUBE_MAP`

- The resulting texture will be about 33% larger than the original texture

- You should almost always use mipmapping! It improves the visual quality.

- See the [documentation](#) for more information
Displacement mapping

- New geometry (vertices) is added and then displaced (in the normal direction) using a map containing height values.
- Main drawback: expensive to store and render the extra geometry!

Bump mapping

- Uses a 2D grayscale texture containing **height values** to perturb the surface normals of the rendered object.
- More efficient than displacement mapping since no extra geometry (vertices) is needed. The eye is made believe the surface is bumpy (has many details).
- The contour is unchanged.
- Introduced by Blinn in 1978.
Bump mapping

Bump map

Sphere rendered with this bump map

Image source: http://www.realtimerendering.com
Bump mapping vs. displacement mapping

Bump mapping vs. displacement mapping

Look at the contours of the objects and their shadows

Normal mapping

- Similar to bump mapping, but uses a 2D RGB texture containing normal vectors (a so-called normal map) to perturb the surface normals of the rendered object.

- A common usage of normal mapping is to generate ("bake") a normal map of a high-polygon model and apply this normal map on a low-polygon version of the same model.

- Allows us to reduce the number of polygons without reducing the amount of details.
Normal mapping

- The normal is mapped to the surface
Normal map example

- The RGB colors used in the normal map show the orientation of the normals
Applying the brick normal map on a sphere

Without normal mapping

With normal mapping
Normal mapping – case study

- The left bunny contains 69566 triangles, the right one 1392 triangles. How can we make the low-polygon bunny look like the high-polygon bunny (without adding vertices)?
Unwrapping the lowpoly model

Seams marked by the user

Unwrapped mesh
Baking a normal map of the highpoly bunny

Highpoly model and lowpoly model (overlaid)

Normal map of the highpoly model

2D texture
Applying the normal map on the lowpoly bunny (without shading)
Applying the normal map on the lowpoly bunny (with shading)
Looks almost the same as the highpoly model!

Original highpoly model

Lowpoly model with normal map

69566 triangles

1392 triangles
Normal mapping (another example)

original mesh
4M triangles

simplified mesh
500 triangles

simplified mesh
and normal mapping
500 triangles

Bump vs. Normal mapping

Bump mapping
- Gray-value map containing height values
- Smaller size on disk
- Compute normals on-the-fly in the shader function
- Somewhat slower rendering (in general)

Normal mapping
- RGB map containing normal vectors
- Larger size on disk
- Normals are already precomputed
- Somewhat faster rendering (in general)
Bump vs. Normal mapping cont.

- Can get close to identical results
- Normal mapping spends more time before rendering
- Bump mapping spends more time during rendering
- Many prefer normal maps which gives better control. But sometimes you need to use bump maps to get the desired effect.
- The techniques are often confused with each other!
Tangent space

- Which direction should we displace the surface normals in when we apply bump or normal mapping in 3D? Not obvious.
- We need to define a **local coordinate system** for each vertex and displace the normal in that coordinate system.
- The coordinate system used for normal/bump mapping is called **tangent space**.
Tangent space cont.

- The tangent space consists of three orthogonal unit vectors: the surface normal $\mathbf{n}$, the binormal $\mathbf{b}$, and the tangent $\mathbf{t}$.

- See Chapter 7.10 in the course book for more details on how to define the tangent space.
Environment mapping

• An **image-based lighting** (IBL) technique for approximating global illumination effects such as **reflection** and **refraction**

• The incoming light from the environment is stored in textures and mapped onto the reflecting/refracting object
Cube environment mapping

- A cube map is a type of texture that stores the incoming light from the environment in the six sides of a cube.
Cube environment mapping

- Find where a perfect reflection hits the cube map. This can be done by calculating a reflection vector $\mathbf{R}$ from the surface normal $\mathbf{N}$ and the *incident vector* $\mathbf{I}$:

$$\mathbf{R} = \mathbf{I} - 2(\mathbf{N} \cdot \mathbf{I})\mathbf{N}$$

- Use the reflection vector as texture coordinate to fetch the incoming light from the cube map texture

*A vector that points from the camera to the reflecting point on the surface*
Example - Cubemap

Texture courtesy: Humus
Color-coded surface normals
Reflection
Refraction
Procedural textures

• **Procedural** textures are computed **on-the-fly** on the GPU by evaluating a function that describes the pattern of interest.

• Unlike image-based textures, procedural textures require no storage and can be rendered at arbitrary resolution.

• Other advantages: easy to create animated patterns and avoid ugly seams.
Procedural textures, patterns

- Procedural textures are functions that take one or several parameters as input and produce a scalar or a vector as output. The input parameters can be texture coordinates, normals, vertex positions, etc.

- Regular procedural patterns like lines, gradients, circles, or squares can be produced with simple functions.

- Irregular or random-looking procedural patterns can be produced with more complex noise functions.
Perlin noise

• A method invented by Ken Perlin for producing textures from "structured chaos"

• Perlin noise has been used extensively for rendering of natural-looking objects with noisy or fractal patterns, e.g., terrain, vegetation, fur, clouds, water, smoke, fire, etc
Perlin noise

2D Perlin noise, one octave

2D Perlin noise, five octaves
Demo: Vertex displacement with animated 3D Perlin noise
Worley noise

- Introduced by Steven Worley in 1996
- Produces a distance map to irregularly positioned feature points
- The distance map can be used to create patterns that look like cells, scales, blobs, etc
- Worley noise generates a different class of patterns than Perlin noise, but has similar applications
Worley noise

2D Worley noise

2D Worley noise, thresholded
Demo: Vertex displacement with animated 3D Worley noise
Procedural noise textures in GLSL

- GLSL implementations of Perlin and Worley noise can be obtained from the following Git repositories:
  - Link 1
  - Link 2 (includes a few demos)
- Just include the noise functions in your vertex or fragment shader
Generating noise textures

- Two approaches:
  - Compute the noise on the CPU and store the result in a 1D, 2D, or 3D texture, which is then uploaded to the GPU memory and used as a conventional texture.
  - Compute the noise on-the-fly on the GPU in a vertex or fragment shader. Possible to do in real-time on modern GPUs.
Skyboxes

- Used for rendering backgrounds in 3D scenes
- Basic idea: place the viewer inside a large cube, and use, e.g., cube mapping to project 2D background images on the cube's faces
- Creates an illusion of a 3D surrounding
- Skyboxes can be stationary or move along with the viewer
- Skydomes are similar, but are constructed with spheres or hemispheres
Skybox texture example
Alpha mapping and billboardling

- Instead of representing objects such as trees or grass as solid surfaces, we can use a set of **billboards** (quads with semi-transparent textures) to create an illusion of 3D objects.

Image source: http://www.realtimerendering.com
Impostor rendering

- An **impostor** of, e.g., a sphere can be created by rendering an image of the sphere as a screen- or world-aligned billboard. Thousands or millions of such billboards can be rendered at little cost.

Image source: http://www.realtimerendering.com
Particle systems

• Used for modeling realistic visual effects such as smoke, fire, water, dust, explosions, cloth, etc. Very common in games.

• Basic idea:
  - Define a set of initial attributes (position, color, opacity, size, etc) for each particle
  - In each frame, update the particle simulation on the CPU/GPU and render the particles as semi-transparent textured quads

Image source: http://www.realtimerendering.com
Particle systems, example
Combining different texture mapping techniques

Environment mapping + skybox + animated 3D Perlin noise
Volume rendering

- The 3D texture mapping capabilities of modern GPUs enable efficient visualization of volumetric (3D) images.
- One of the most common volume rendering techniques is **GPU-accelerated ray-casting**.
- You will learn more about this topic in Lecture 10 (and in the project if you select the volume rendering task).
Resources

• 3D models and textures:
  - http://www.turbosquid.com (non-free and free samples)
  - http://thefree3dmodels.com
  - https://3dwarehouse.sketchup.com

• Cube map textures: