Texture mapping

How to add details without adding extra geometry
High-detailed objects

- One way to make objects look good is by using a high resolution version
  - i.e., lots of polygons
- This is expensive!
- And sometimes unnecessary!

Techniques

- When the number of polygons is reduced, we need some new way to make the object look good:
  - Texture mapping
  - Bump mapping
  - Normal mapping

- We can also add some extra features:
  - Environment mapping
Texture mapping

Original model

Texture mapped model

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

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 common
Texture elements

- The elements in a texture are called **texels** (texture pixels) and can hold many different types of information:
  - Color (most common)
  - Normals
  - Opacity
  - Intensity
  - Positions
  - Height
  - etc

Different textures of a brick wall
Texture coordinates

- Texture coordinates range from 0 to 1 and are typically denoted as $s$, $t$, 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

- Can be done programmatically when defining simple geometric objects like planes or cubes
- 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
Projector functions

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

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

- Modern shader-based OpenGL:
  - Provide texture coordinates as vertex attributes (more about this in lecture 10)
Loading texture images from file

- OpenGL does not provide functions 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. Just include them in your project/code base, and load the PNG image as

```cpp
#include "lodepng.h"
...
std::string filename = "718smiley.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 texture from the loaded image

... 

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

...
Activating the texture

- At rendering, you need to select the active texture unit (GL_TEXTURE0, GL_TEXTURE1, etc) and bind your texture to this unit. Example:

```c
...  
  // Select active texture unit
  glEnable(GL_TEXTURE0);

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

  // Draw
  ...
```
Links to some useful documentation

- `glGenTextures`
- `glBindTexture`
- `glTexParameter`
- `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_REPEAT
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 that correspond to one texel (magnification). This can lead to nasty sampling artifacts (see next slide).

- OpenGL handles this via filtering operations:
  - GL_NEAREST
  - GL_LINEAR
  - GL_NEAREST_MIPMAP, GL_NEAREST_MIPMAP_LINEAR,
  - GL_LINEAR_MIPMAP, GL_LINEAR_MIPMAP_LINEAR
Texture aliasing artifacts

Nearest sampling

Linear filtering (better, but still aliasing)

Image courtesy: Edward Angel
Antialiasing and Mipmapping

- Aliasing artifacts can be efficiently suppressed with a method called *mipmapping*
- The original texture is filtered down repeatedly into smaller images, often called a *mipmap chain*
- During rendering, the mipmap level is selected based on viewing distance

Source: http://www.realtimerendering.com
Mipmapping linear filtering

Mipmapping linear filtering (less aliasing!)

Image courtesy: Edward Angel
Mipmaps

- Mipmaps can be generated automatically in OpenGL with the function
  
  ```
  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

- See the documentation for more information
Texture mapping a sphere

2D texture

Texture mapping
Multiple textures can be applied

Sphere rendered with four different textures
Alpha mapping and billboarding

Source: http://www.realtimerendering.com
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 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 to believe the surface is bumpy (has many details).
- The contour is unchanged.
- Introduced by Blinn in 1978.
Bump mapping
Bump mapping (another example)

Bump map

Sphere rendered with this bump map
Yet another example

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 perturbe 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 in map | Surface normal | Mapped normal
---|---|---
Normal in map | Surface normal | Mapped normal
Normal map example

- The RGB colors used in the normal map show the orientation of the normals

![Normal map of a brick wall](image)
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

Normal mapping
- RGB map containing normal vectors
- Larger size on disk
- Normals are already precomputed
- Somewhat faster rendering
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 perturbe the surface normals in when we apply bump or normal mapping in 3D? Not obvious.

• We need to define a local coordinate system (tangent space) for each vertex and perturb the normal in this space.

• 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 9.13.1 in the course book for more details on how to define the tangent space
Tangent space cont.

Source: http://www.realtimerendering.com
Environment mapping

- Map the surrounding world on the object
Cube environment mapping

- Create a cube map texture by rendering the world in 6 directions (or use an existing cube map)
Cube environment mapping

- Find where a perfect reflection hits the cube map. This can be derived from the surface normals
- Compute light on the surface by sampling the texture
Environment mapping - Cubemap

Texture courtesy: Humus
Environment mapping - Normals
Environment mapping - Reflection
Environment mapping - Refraction
Procedural textures

- Textures used in real-time rendering applications are often acquired with some imaging device, pre-computed, or created by an artist...
- but they can also be generated procedurally during the rendering by evaluating a function that describes the pattern of interest.
- Procedural textures are computed on-the-fly on the GPU. They 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

- Ken Perlin invented a method for producing textures from "structured chaos"
- The method is often referred to as Perlin noise
- Perlin noise has been used extensively for rendering of natural-looking objects with noisy or fractal patterns, e.g., mountains, vegetation, fur, clouds, water, smoke, fire, etc
- The following slides show some examples:
Perlin noise

2D Perlin noise, one octave

2D Perlin noise, five octaves
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
Vertex displacement with animated 3D Worley noise
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 to render 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

Texture courtesy: unknown
Particle systems

- Represent each particle as a semi-transparent textured quad

Source: http://www.realtimerendering.com
Combining texture mapping techniques

- You can combine different texture mapping techniques, e.g., bump mapping and environment mapping
- The following slide shows an example:
Environment mapping + skybox + animated 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 Lab 4
Some concluding remarks

- Texture mapping techniques allow us to introduce extra details without using extra vertices.
- The improved visual effect is often worth the extra implementation effort and computational cost.
- There are lots of techniques that we haven't covered here, for instance:
  - Parallax mapping
  - Relief mapping
  - Global illumination (lightmaps, shadow mapping)
  - Render to texture (for, e.g., image postprocessing)