Texture Mapping
Texture Mapping

- An effective method for adding surface detail, mapping texture patterns on the surface of (smooth) objects

- Real objects have details with very high frequencies
  - The surface of an orange is bumpy
  - The surface of a wooden table carries the wood’s colors
  - Mirrors and other shiny surfaces reflect the environment

- Idea: Modify the shading equation at several places
  - Surface color
  - Surface normal
  - Transparency
  - Reflectance
(Anti-)Aliasing

Antialiasing demo
(Anti-)Aliasing
Aliasing
Anti-Aliasing
Texture Coordinates

- Associates a portion of the texture with a polygon
- Assign texture coordinates to polygon vertices
Interpolating Texture Coordinates

\[ c \ C_1 + d \ C_2 + e \ C_3 \]
\[ (c + d + e = 1) \]

\[ a \ C_1 + (1 - a) \ C_2 \]
\[ b \ C_1 + (1 - b) \ C_3 \]
Parameterization

- We need a mapping between the model and the image

$$F : Model \in R^3 \rightarrow Image \in R^2$$

- With parametric surfaces this mapping is trivial
  - Why?

- What if we have polygonal models?
  - The mapping is not trivial at all
  - This process is called parameterization.
  - An active research topics in CG
Parameterization

- Transform the mesh into a canonical shape
  - Part of plane, sphere, cylinder
Parameterization

- Transform the mesh into a canonical shape
  - Part of plane, sphere, cylinder
- Not all parameterizations are good
  - Bijection
  - Minimize distortion
  - The trivial ones you learned about in lecture are not always good (why?)
- What can be done?
  - Conservation of (relative) distances (isometry)
    - Not always possible
  - Conservation of angles (conformal maps)
    - Not always what you want.
  - Other ideas?
Parameterization

- Flattened version of the camel on the right
- The colors encode the position in space
Computing texture coordinates

- During scan-conversion The texture is sampled
- Assign texture coordinates
  - A mapping of each vertex to the image
  - Interpolate the texture coordinates
    - Same way color is done

\[(u_0, v_0), (u_1, v_1), (u_2, v_2)\]

\[(0,0), (1,0), (0,1), (1,1)\]
How do we sample?

- Not every point on the surface has a pixel on the image
- Given an image and a real coordinate return a value

Common schemes

- Nearest neighbor
  - Return the pixel value that is the closest
- Bilinear interpolation
  - Given \((u, v)\) find its four neighbor pixels
  - Compute interpolation parameters
  - Compute the final color as a blend
- Higher order sampling schemes
  - Bicubic interpolation
  - Gaussian kernels
  - Might be too expensive in real-time apps
Bilinear Interpolation
Bicubic Interpolation
Sampling Schemes

- What is the parametric domain?
  - \([0,1] \times [0,1]\)
  - So what does it mean to have texture coordinates of \((1.1, 2)\)
  - Clamping Vs Repeating

Texture Mapping - Center for Graphics and Geometric Computing, Technion
Texture Aliasing

many texels to one pixel

one texel to many pixels
Sampling Schemes

- **Mipmapping**
  - MIP – Multum In Parvo
  - “Much in a small space”

- **The idea:**
  - Hold a pyramid of images (a mipmap)
  - Choose a layer based on distance
  - What is it good for?

- **Trilinear interpolation**
  - Interpolate between layer as well

- **Anisotropic interpolation**
  - What?
Textures in OpenGL

OpenGL

- Activate texture unit, create new texture object and bind it

```cpp
glActiveTexture(GL_TEXTURE0);
GLuint tex;
glGenTextures(1, &tex);
glBindTexture(GL_TEXTURE_2D, tex)
```

- Fill the texture buffer

```cpp
GLubyte texels[512][512][3];
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 512, 512, 0, GL_RGB, GL_UNSIGNED_BYTE, texels);
```
Textures in OpenGL

- **OpenGL**
  - Set texture coordinates as vertex attributes
    ```
texcoord_loc = glGetUniformLocation(program, "texcoord");
 glEnableVertexAttribArray(texcoord_loc);
 glVertexAttribPointer(texcoord_loc, 2, GL_FLOAT, GL_FALSE, 0, 0);
    ```
  - Set texture map
    ```
Tex_loc = glGetUniformLocation(program,"texMap");
 glUniform1i(tex_loc, i);
    ```
Textures in OpenGL

- GLSL
  - Vertex shader

```glsl
in vec2 texcoord;
in vec4 vPosition;
...
out vec2 st;
...
void main()
{
  ...  
gl_Position=vPosition;
st=texcoord;
}
```
Textures in OpenGL

GLSL

- Fragment shader

```glsl
in vec2 st;
uniform sampler2D texMap
out vec4 color;
void main()
{
    color = texture2D(texMap, st);
}
```
Environment Mapping
Flight of the Navigator 1986

Terminator 2
1991
Displacement Mapping

Add “texture” to the surface geometry
Normal Mapping

Add “texture” to the surface normals

\[ I = I_a k_a + I_p \left( k_d (N \cdot L) + k_s (R_N \cdot V)^n \right) \]
Normal Mapping

4M faces

8K faces

8K faces, normal-mapped

normal-map
Normal vs. Displacement
Normal Mapping
Normal Mapping

- Original mesh: 4M triangles
- Simplified mesh: 500 triangles
- Simplified mesh and normal mapping: 500 triangles
Normal Mapping
Combined

Normal + Displacement Mapped

Texture Mapped

normal map
Volumetric Textures

- A volumetric texture mapping involves a 3D function $\rho$ which gives for each point $P$ its corresponding color:
  $$\rho = \rho(x,y,z)$$

- Texture Space – the 3D space that holds the texture (discrete or continuous)

- For each point $P(x_0,y_0,z_0)$ of the object’s surface, the color at the corresponding point $\rho(x,y,z)$ is computed.
Transformations

- When an object with a volumetric texture mapping is transformed, the volumetric texture function must also be transformed
  - The function $\rho$ must follow the transformations of the object
  - The texture map is in the object space
  - For animation effects, this transformation is not always wanted (clouds moving, water motion)
Procedural Texture Generation

- Use a mathematical process to create texture
  - Define only a small number of parameters
  - Can also use random numbers

- Advantages
  - Easy to create unlimited numbers of textures
  - Continuous textures
  - Compact representation

- Disadvantages
  - Difficult to program
  - Difficult to predict
  - Slower to evaluate

- See also procedural geometry and music
Basic Procedural Textures

- **Simple marble**
  - function simple_marble(x)
    - return marble_color(sin(x));
  - // marble_color maps scalars to colors

- **Bombing**
  - Randomly drop bombs of various shapes, sizes and orientation onto the texture space (store data in table)
  - Replicate the texture space
  - For a given point in space, search table and determine if inside shape. If so, color by shape
  - Otherwise, color by objects color
Basic Procedural Textures

- **Marble**
  
  ```
  function simple_marble(x)
  return marble_color(sin(x));
  // marble_color maps scalars to colors
  ```

- **Bombing**
  
  Randomly drop bombs of various shapes, sizes, and orientation onto the texture space (store data in table).

  1. Replicate the texture space.
  2. For a given point in space, search the table and determine if inside shape. Color by shape.
  3. Otherwise, color by objects color.
Basic Procedural Textures

- Bombing example:
Basic Procedural Textures

- Wood
  - The function $\rho$ is an orthogonal projection of a set of concentric circles
  - Vary the thickness and spacing to create realism
  - Example:
Perlin Noise

- A function with random behavior
  - A kind of procedural texture by itself
- Total randomness is not very natural
  - An Image Synthesize/Ken Perlin (SIGGRAPH’85)
  - “Coherent noise”
    - Noise that changes smoothly
- How do we generate “smooth” noise?
- Interpolation
  - For each \( x, y, z \in \mathbb{N}, H(x, y, z) = \text{Random number} \)
  - If \((x, y, z)\) are all integers:
    - \( \text{Noise}(x, y, z) = H(x, y, z) \)
  - Otherwise:
    - \( \text{Noise}(x, y, z) = \text{a trilinear (cubic) interpolation of } H(x, y, z) \)
The Function Dnoise

The function Dnoise is the derivative of the function Noise:

\[ D\text{noise}(x,y,z) = \left( \frac{d\text{Noise}}{dx}, \frac{d\text{Noise}}{dy}, \frac{d\text{Noise}}{dz} \right); \]

This generates a vector of three values at each point.
Turbulence Function

- We take for example the 1D case: $f(x)$
- Turbulence = $f(x) + 2^{-1}f(2^1x) + 2^{-2}f(2^2x) + 2^{-3}f(2^3x) + \ldots$
- Sum of scaled versions of $f(x)$
  - Higher frequency, lower amplitude

```c
function turbulence(p)
    t = 0;
    scale = 1;
    while (scale > pixelsize) {
        t += abs(Noise(p/scale)*scale);
        scale/=2;
    }
    return t;
```
Turbulence (cont.)

- Analyze the simple 2D domain of $\sin(x)$

\[
g_1(x) := \sin(x) \\
\]
\[
g_2(x) := \sin(x) + \frac{1}{2} \sin(2 \cdot x) + \frac{1}{4} \sin(4 \cdot x) + \frac{1}{8} \sin(8 \cdot x) + \frac{1}{16} \sin(16 \cdot x) + \frac{1}{32} \sin(32 \cdot x)
\]
Turbulence (cont.)

- Analyze the simple 2D domain of $|\sin(x)|$
- Fractal behavior, double frequency and half amplitude

\[ x = 0, 0.1 \pi, 2 \]

\[
f_1(x) = |\sin(x)|
\]

\[
f_2(x) = |\sin(x)| + \frac{1}{2} |\sin(2x)|
\]

\[
f_3(x) = |\sin(x)| + \frac{1}{2} |\sin(2x)| + \frac{1}{4} |\sin(4x)|
\]

\[
f_4(x) = |\sin(x)| + \frac{1}{2} |\sin(2x)| + \frac{1}{4} |\sin(4x)| + \frac{1}{8} |\sin(8x)|
\]

\[
f_5(x) = |\sin(x)| + \frac{1}{2} |\sin(2x)| + \frac{1}{4} |\sin(4x)| + \frac{1}{8} |\sin(8x)| + \frac{1}{16} |\sin(16x)|
\]

\[
f_6(x) = |\sin(x)| + \frac{1}{2} |\sin(2x)| + \frac{1}{4} |\sin(4x)| + \frac{1}{8} |\sin(8x)| + \frac{1}{16} |\sin(16x)| + \frac{1}{32} |\sin(32x)|
\]
1D Noise function

Sum of Noise Functions = (Perlin Noise)
Function Turbulence (cont.)
More Textures

- Marble

```cpp
function marble(p)
    x = p.x + turbulence(p);
    return marble_color(sin(x))
```
More Textures

- Wood (better one)

```plaintext
function wood(p)
    x = (p.x^2+p.y^2) + turbulence(point);
    return wood_color(sin(x))
```

Wood texture function implementation.
More Textures

- Bumpy surface:
  - normal += Dnoise(point)
Another Example...

- \( \text{color} = \text{Colorful}(\text{Noise}(k*\text{point})) \)
  // scaled the texture by multiplying the domain of
  // \text{Noise}() by a constant \( k \)