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
(Anti-)Aliasing
Aliasing
Anti-Aliasing

a  a  a
Texture Coordinates

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

![Texture Coordinates Diagram](image)
(0.3,0.4) 
(0.6,0.8) 
(0.8,0.2) 
(0.57,0.47) 

(0,0) 
(1,0) 
(1,1) 

texels
Interpolating Texture Coordinates

\[ a \ C_1 + (1 - a) \ C_2 \]

\[ c \ C_1 + d \ C_2 + e \ C_3 \]

\[ (c + d + e = 1) \]

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

- We need a mapping between the model and the image

\[ F : \text{Model} \in \mathbb{R}^3 \rightarrow \text{Image} \in \mathbb{R}^2 \]

- 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 topic 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

Texture Coordinates:

- $(u_0, v_0)$
- $(u_1, v_1)$
- $(u_2, v_2)$

The texture coordinates are mapped to the image using the same method as interpolating colors.
Sampling Scheme

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
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 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
  - reduces blur and preserves detail at extreme viewing angles
Sampling Scheme

- Aliasing problems?
  - If the texture contains high frequencies and sampled at low rate sampling artifacts may occur
  - Use more samples
    - Compute multiple samples from each texture coordinate pair by adding jitter
  - Use post filtering - smooth the output image.

- Sampling is rather expensive operation
  - Use it only when you need
    - After Z computation?
Environment Mapping
Flight of the Navigator 1986

Terminator 2 1991
Displacement Mapping

Add “texture” to the surface geometry
Displacement mapping - example
Normal Mapping

Add “texture” to the surface normals

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

Normal + Displacement Mapped

Texture Mapped
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)
  - 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

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:

```
rings
added eccentricity
added twist
added tilt
```
Perlin Noise

- used to simulate elements from nature
  - Not consuming memory
- 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 \( x, y, z \in \mathbb{N} \), \( H(x, y, z) = \) Random number
  - If \((x,y,z)\) are all integers:
    - \(\text{Noise}(x,y,z) = H(x,y,z)\)
  - Otherwise:
    - \(\text{Noise}(x,y,z) = \) a trilinear (cubic) interpolation of \(H(x,y,z)\)
Function Noise (cont.)
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

```plaintext
function turbulence(p)
    t = 0;
    scale = 1;
    Do N times
        t += 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

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

\[ f_1(x) := |\sin(x)| \]
\[ f_2(x) := |\sin(x)| + \frac{1}{2} |\sin(2x)| \]
\[ f_3(x) := |\sin(x)| + \frac{1}{2} |\sin(2x)| + \frac{1}{4} |\sin(4x)| \]
\[ f_4(x) := |\sin(x)| + \frac{1}{2} |\sin(2x)| + \frac{1}{4} |\sin(4x)| + \frac{1}{8} |\sin(8x)| \]
\[ f_5(x) := |\sin(x)| + \frac{1}{2} |\sin(2x)| + \frac{1}{4} |\sin(4x)| + \frac{1}{8} |\sin(8x)| + \frac{1}{16} |\sin(16x)| \]
\[ f_6(x) := |\sin(x)| + \frac{1}{2} |\sin(2x)| + \frac{1}{4} |\sin(4x)| + \frac{1}{8} |\sin(8x)| + \frac{1}{16} |\sin(16x)| + \frac{1}{32} |\sin(32x)| \]
1D Noise function

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

- Marble

```plaintext
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))
```
More Textures

- Bumpy surface:
  - normal += Dnoise(point)
Normal Bump

Marble

Bump
Another Example...

- color = Colorful(Noise(k*point))
  // scaled the texture by multiplying the domain of
  // Noise() by a constant k
Texture Mapping - Center for Graphics and Geometric Computing, Technion

Glass

Corona