Texturing

- 2D texturing
- Bump mapping
- Procedural solid texturing

2D texturing

![Image of 2D texturing concept]

Texture coordinates

![Image of texture coordinates]

Texturing triangle meshes

- Assume triangle vertices have texture coordinates associated with them
- Usually, provide/model/generate texture coordinates using 3D modeling program

Texture look-up

1. Find texture coordinate at intersection using barycentric coordinates $\alpha$, $\beta$, $\gamma$ at hit point

![Image of texture look-up process]
### Texture look-up

2. Interpolate texture using bilinear interpolation

![Texture look-up diagram](image)

### Using textures

- Usually for shading
  - Use color texture values as diffuse reflection coefficients
  - Can also use grey scale texture as gloss map, i.e., shininess in Phong model
- Can also encode other shading parameters in textures

### Texturing

- 2D texturing
- Bump mapping
- Procedural solid texturing

### Bump mapping

Offsets along normal

Texture map stores offsets \(d(u)\)

\[
p'(u) = p(u) + d(u)n(u)
\]

Compute normal \(n'(u)\) and use it for shading
Use original surface for ray tracing

[Bump mapping](http://en.wikipedia.org/wiki/Bump_mapping)
Normal computation

- Desired normal is cross product of tangent vectors of displaced surface
  \[ \mathbf{n}(u, v) = \frac{\partial \mathbf{p}(u,v)}{\partial u} \times \frac{\partial \mathbf{p}(u,v)}{\partial v} \]

- Tangent vectors using product rule
  \[ \frac{\partial \mathbf{p}(u,v)}{\partial u} = \frac{\partial \mathbf{p}(u,v)}{\partial u} + \frac{\partial \mathbf{p}(u,v)}{\partial v} \mathbf{n}(u, v) + d(u, v) \frac{\partial \mathbf{m}(u,v)}{\partial u} \]

- Ignore \( d(u, v) \frac{\partial \mathbf{m}(u,v)}{\partial u} \)
  - Means base surface is assumed to be flat

Tangents of base surface

- 3D coordinates on surface \( \mathbf{p} = (p_x, p_y, p_z) \)
- First find \( \mathbf{p}(u, v) \) for triangle that is hit
  - We know it is linear
  - Solve for each coordinate separately, here \( x \)

  \[ p_x(u,v) = [ u \ v \ 1 ] \begin{bmatrix} a & b & c \end{bmatrix} \]

- Three constraints

\[ \begin{bmatrix} p_{x1} \ p_{x2} \ p_{x3} \end{bmatrix} = \begin{bmatrix} u1 \ v1 \ 1 \\ u2 \ v2 \ 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \]

Tangents of base surface

- The same for \( y \) and \( z \) coordinates

\[ \begin{bmatrix} p_{y1} \ p_{y2} \ p_{y3} \end{bmatrix} = \begin{bmatrix} u1 \ v1 \ 1 \\ u2 \ v2 \ 1 \end{bmatrix} \begin{bmatrix} b \\ c \end{bmatrix} \]

Texturing

- 2D texturing
- Bump mapping
- Procedural solid texturing

Normal computation

- Numerically approximate partial derivative of displacement function by finite differencing
  \[ \frac{\partial \mathbf{m}(u,v)}{\partial u} \approx \frac{\mathbf{m}(u+\Delta u,v) - \mathbf{m}(u,v)}{\Delta u} \]
  - Same for \( \frac{\partial \mathbf{m}(u,v)}{\partial v} \)
  - Note these are scalars

- Also need tangents of base surface
  \[ \frac{\partial \mathbf{p}(u,v)}{\partial u}, \frac{\partial \mathbf{p}(u,v)}{\partial v} \]
  - These are vectors

Tangents of base surface

- Solving for the coefficients \( a, \ldots, i \) using 3x3 matrix inversion
- Desired derivatives are

\[ \frac{\partial \mathbf{p}(u,v)}{\partial u} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}, \quad \frac{\partial \mathbf{p}(u,v)}{\partial v} = \begin{bmatrix} d \\ e \\ f \end{bmatrix} \]
Procedural solid texturing

![Diffuse coefficient $R_d = \text{const}$](image1)

Procedural solid texturing

![Diffuse coefficient $R_d(x, y, z) = N(y)$](image2)

Procedural solid texturing

```cpp
solid : material {
    shade(ray, hit_record)
    ...
}

solid::shade {
    Rd = N(hit_record.p)
    ...
}

• “Procedural”: evaluate $N$ on the fly
• “Solid”: $N$ is a function of hit position $p$, not $u,v$ texture coordinates
```

Procedural solid texturing

```cpp
solid : material {
    shade(ray, hit_record)
    ...
}

solid::shade {
    Rd = N(hit_record.p)
    ...
}
```

How do we choose $N$ to get interesting results?

Noise

- Band limited
- Not scale invariant
- Translation invariant
- Rotation invariant

Noise

- Emitted in 3D
- Map noise to colors

[Ken Perlin]
Perlin noise in 3D
- Create randomly oriented unit vectors (gradients) on a 3D grid
- Use Hermite interpolation to generate a noise value
  - Interpolating function matches given normals
- Leads to band limited interpolant

Perlin noise in 1D
- Blending function
  \[ s(x) = 6x^5 - 15x^4 + 10x^3 \]
- Continuous derivatives at boundaries
  \[ s'(0) = 0, s'(1) = 0, s''(0) = 0, s''(1) = 0 \]

Fractal noise and turbulence
- Fractal noise
  \[ F(x) = \sum_i a_i \cdot N(x \cdot f_i) \]
- Turbulence: use absolute values
  \[ T(x) = \sum_i \|a_i \cdot N(x \cdot f_i)\| \]
- Frequencies usually in octaves
  \[ f_i = 2f_{i-1} \]
- Amplitude
  \[ a_i = p \cdot a_{i-1} \]
- Persistency
  \[ p < 1, \text{ often } p = 0.5 \]
Fractal noise in 1D

\[ N(x) + 0.5N(2x) + 0.25N(4x) + 0.125N(8x) \]

Comparison

- Noise
  - [Ken Perlin](http://www.noisemachine.com/talk1/index.html)
- Turbulence
  - [Ken Perlin](http://mrl.nyu.edu/~perlin/noise/)

More on Perlin noise

- Easy to read intro
  - [http://www.noisemachine.com/talk1/index.html](http://www.noisemachine.com/talk1/index.html)
- Technical paper on 3D implementation
  - [http://mrl.nyu.edu/~perlin/noise/](http://mrl.nyu.edu/~perlin/noise/)
- Java code
  - [http://mrl.nyu.edu/~perlin/noise/](http://mrl.nyu.edu/~perlin/noise/)

Cellular noise

- Place random points in a grid
- Use distance to neighbors as a noise function
  \[ F_n(x) = \text{distance to n-th neighbor} \]
- Fast implementation:
  “A cellular texture basis function”,
  Steven Worley

Turbulence

\[ \text{color} = \sin(x + T(x, y, z)) \]
Cellular noise

Worley F1 [Anson Chu]

Worley F1 in 3D

Cellular noise

Worley F2 [Ansun Chu]

Worley F2-F1 [Ansun Chu]

Cellular noise

Worley F2-F1 [Steve Worley]

Manhattan Distance, F1 [Steve Worley]
Fractal cellular noise

\[ F_n^*(x) = \sum_i a_i F_n(x \cdot f_i) \]

Further reading

- “Texturing and Modeling: A Procedural Approach”, Ebert, Musgrave, Peachey, Perlin, Worley, Mark