**University of British Columbia**  
**CPSC 314 Computer Graphics**  
**Jan-Apr 2007**  
**Tamara Munzner**  
**Textures III, Procedural Approaches**  
**Week 10, Mon Mar 19**  

**Reading for Last Time and Today**
- FCG Chap 11 Texture Mapping
  - except 11.8
- RB Chap Texture Mapping
- FCG Sect 16.6 Procedural Techniques
- FCG Sect 16.7 Groups of Objects

**Final Clarification: HSI/HSV and RGB**
- HSI/HSI conversion from RGB
  - hue same in both
  - value is max, intensity is average
  \[
  H = \cos^{-1}\left[\frac{1}{2}\left(\frac{R - G}{R - B} + \frac{R - B}{R - G}\right)\right]
  \]
  \[
  S = 1 - \min(R, G, B)
  \]
  \[
  V = \max(R, G, B)
  \]
  
- If (B > G), \( H = 360 - H \)

**News**
- Project 3 grading slot signups
  - Mon 11-12
  - Tue 10-12:30, 4-6
  - Wed 11-12, 2:30-4
  - go to lab after class to sign up if you weren’t here on Friday
  - everybody needs to sign up for grading slot!

**Review: MIPmapping**
- image pyramid, precompute averaged versions

**Review: Basic OpenGL Texturing**
- setup
  - generate identifier: glGenTextures
  - load image data: glTexImage2D
  - set texture parameters (tile/clamp/...): glTexParameteri
  - set texture drawing mode (modulate/...): glTexEnvf
  - drawing
    - enable: glEnable
    - bind specific texture: glBindTexture
    - specify texture coordinates before each vertex: glTexCoord

**Review: Perspective Correct Interpolation**
- screen space interpolation incorrect

**Review: Bump Mapping: Normals As Texture**
- create illusion of complex geometry model
- control shape effect by locally perturbing surface normal

**Environment Mapping**
- used to model object that reflects surrounding textures to the eye
  - movie example: cyborg in Terminator 2
  - different approaches
    - sphere, cube most popular
    - OpenGL support
      - GL_SPHERE_MAP, GL_CUBE_MAP
    - others possible too

**Sphere Mapping**
- texture is distorted fish-eye view
  - point camera at mirrored sphere
  - spherical texture mapping creates texture coordinates that correctly index into this texture map

**Displacement Mapping**
- cheap way to achieve reflective effect
  - change surface geometry instead
  - only recently available with realtime graphics
  - need to subdivide surface

**Environment Mapping**
- used to model object that reflects surrounding textures to the eye
  - movie example: cyborg in Terminator 2
  - different approaches
    - sphere, cube most popular
    - OpenGL support
      - GL_SPHERE_MAP, GL_CUBE_MAP
    - others possible too

**Environment Mapping**
- used to model object that reflects surrounding textures to the eye
  - movie example: cyborg in Terminator 2
  - different approaches
    - sphere, cube most popular
    - OpenGL support
      - GL_SPHERE_MAP, GL_CUBE_MAP
    - others possible too

**Environment Mapping**
- used to model object that reflects surrounding textures to the eye
  - movie example: cyborg in Terminator 2
  - different approaches
    - sphere, cube most popular
    - OpenGL support
      - GL_SPHERE_MAP, GL_CUBE_MAP
    - others possible too
**Cube Mapping**

- 6 planar textures, sides of cube
- point camera in 6 different directions, facing out from origin

**Procedural Texture Effects: Bombing**

- randomly drop bombs of various shapes, sizes and orientation into texture space (store data in table)
  - for point P search table and determine if inside shape
    - if so, color by shape
    - otherwise, color by objects color

**Volumetric Bump Mapping**

Marble

Bump

**Cube Mapping**

- direction of reflection vector $r$ selects the face of the cube to be indexed
  - co-ordinate with largest magnitude
    - e.g., the vector (-0.2, 0.5, -0.84) selects the -Z face
  - remaining two coordinates (normalized by the 3rd coordinate) selects the pixel from the face.
    - e.g., (-0.2, 0.5) gets mapped to (0.38, 0.80).
- difficulty in interpolating across faces

**Volumetric Texture Principles**

- 3D function $p(x,y,z)$
- texture space – 3D space that holds the texture (discrete or continuous)
- rendering: for each rendered point $P(x,y,z)$ compute $p(x,y,z)$
- volumetric texture mapping function/space transformed with objects

**Perlin Noise: Procedural Textures**

- several good explanations
  - FCG Section 10.1
  - http://www.noisemachine.com/talk1
  - http://freepace.virgin.net/hugo.elias/models/m_perlin.htm

**Procedural Approaches**

- generate "image" on the fly, instead of loading from disk
  - often saves space
  - allows arbitrary level of detail

**Procedural Texture Effects: Bombing**

- function boring_marble(point)
  - x = point.x;
  - return marble_color(sin(x));
  // marble_color maps scalars to colors

**Volumetric Texture**

- define texture pattern over 3D domain - 3D space containing the object
  - texture function can be digitized or procedural
  - for each point on object compute texture from point location in space
  - common for natural material/irregular textures (stone, wood, etc…)

**Perlin Noise: Coherency**

- smooth not abrupt changes

**Procedural Texture Effects**

- simple marble
  - function turbulence(p)
    - t = 0; scale = 1;
    - while (scale > pixelsize) {
      - t += abs(Noise(p/scale)*scale);
      - scale/=2;
    } return t;

**Generating Coherent Noise**

- just three main ideas
  - nice interpolation
  - use vector offsets to make grid irregular
  - optimization
    - sneaky use of 1D arrays instead of 2D/3D one
**Interpolating Textures**
- nearest neighbor
- bilinear
- hermite

**Fractal Landscapes**
- fractals: not just for “showing math”
  - triangle subdivision
  - vertex displacement
  - recursive until termination condition

**Language-Based Generation**
- L-Systems: after Lindenmayer
  - Koch snowflake: F -> FLFRRFLF
  - F: forward, R: right, L: left
  - Mariano’s Bush:
    F=F-{F+F+F}+[F+F+F+F]
    - angle 16

**Optimization**
- save memory and time
- conceptually:
  - populate with random number generator
- actually:
  - precompute two 1D arrays of size n (typical size 256)
  - permutation of integers 0 to n-1
  - lookup
    - \( g(i,j,k) = g( (i + F(0\cdot j + F(0\cdot k)) \mod n \mod n) \) } mod n }

**Perlin Marble**
- use turbulence, which in turn uses noise:

```javascript
function marble(point)
    x = point.x + turbulence(point);
    return marble_color(sin(x))
```

**Meaningful Names**
- procedural approach
  - compute something on the fly
  - often less memory cost
  - visual richness

**Procedural Modeling**
- textures, geometry
  - nonprocedural: explicitly stored in memory
- procedural approach
  - compute something on the fly
  - often less memory cost
  - visual richness
- fractals, particle systems, noise

**Fractal Dimension**
- \( D = \log(N)/\log(r) \)
  - \( N \) = measure, \( r \) = subdivision scale
  - Hausdorff dimension: noninteger

**Particle Systems**
- loosely defined
  - modeling, or rendering, or animation
  - key criteria
    - collection of particles
    - random element controls attributes
      - position, velocity (speed and direction), color, lifetime, age, shape, size, transparency
      - predefined stochastic limits: bounds, variance, type of distribution

**Particle System Examples**
- objects changing fluidly over time
  - fire, steam, smoke, water
- objects fluid in form
  - grass, hair, dust
- physical processes
  - waterfalls, fireworks, explosion
- group dynamics: behavioral
  - birds/bats flock, fish school, human crowd, dinosaur/elephant stampede

**Particle Systems Demos**
- general particle systems
  - [http://www.wonderlouch.com](http://www.wonderlouch.com)
- boids: bird-like objects

**Optimization**
- save memory and time
- conceptually:
  - populate with random number generator
- actually:
  - precompute two 1D arrays of size n (typical size 256)
  - permutation of integers 0 to n-1
  - lookup
    - \( g(i,j,k) = g( (i + F(0\cdot j + F(0\cdot k)) \mod n \mod n) \) } mod n }

**Particle Life Cycle**
- generation
  - randomly within “fuzzy” location
  - initial attribute values: random or fixed
- dynamics
  - attributes of each particle may vary over time
  - color derer as particle cools off after explosion
  - can also depend on other attributes
    - position: previous particle position + velocity + time
    - death
      - age and lifetime for each particle (in frames)
      - or if out of bounds, too dark to see, etc

**Particle System Rendering**
- expensive to render thousands of particles
- simplify: avoid hidden surface calculations
- each particle has small graphical primitive (blob)
- pixel color: sum of all particles mapping to it
- some effects easy
  - temporal anti-aliasing (motion blur)
  - normally expensive: supersampling over time
  - position, velocity known for each particle
  - just render as streak

**Self-Similarity**
- infinite nesting of structure on all scales

**Perlin Marble**
- use turbulence, which in turn uses noise:

```javascript
function marble(point)
    x = point.x + turbulence(point);
    return marble_color(sin(x))
```
Procedural Approaches Summary

- Perlin noise
- fractals
- L-systems
- particle systems

- not at all a complete list!
  - big subject: entire classes on this alone