Homework 3 out Monday

Textures III, Procedural Approaches
Week 8, Fri Mar 4
http://www.ugrad.cs.ubc.ca/~cs314/VJan2005

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

Review: Texture Objects and Binding
- texture objects
  - texture management: switch bind, not reloading
  - can prioritize textures to keep in memory
  - Q: what happens to textures kicked out of memory?
    - A: resident memory (on graphics card) vs. nonresident (on CPU)
    - details hidden from developers by OpenGL

Review: Perspective Correct Interpolation
- screen space interpolation incorrect
  \[ s = \frac{ax_0 + bx_1 + cx_2}{\alpha x_0 + \beta x_1 + \gamma x_2} \]

Review: Reconstruction
- how to deal with:
  - pixels that are much larger than texels?
    - apply filtering, “averaging”
  - pixels that are much smaller than texels?
    - interpolate
Review: MIPmapping
- image pyramid, precompute averaged versions

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

Review: Displacement Mapping
- bump mapping gets silhouettes wrong
  - shadows wrong too
- change surface geometry instead
  - only recently available with realtime graphics
  - need to subdivide surface

Review: Environment Mapping
- cheap way to achieve reflective effect
- generate image of surrounding
- map to object as texture

Review: Sphere Mapping
- texture is distorted fish-eye view
  - point camera at mirrored sphere
  - spherical texture coordinates

Review: Cube Mapping
- 6 planar textures, sides of cube
  - point camera outwards to 6 faces
    - use largest magnitude of vector to pick face
    - other two coordinates for (s,t) texel location

Page 2
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.)

Volumetric Texture Principles

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

Volumetric Bump Mapping

Marble

Bump

Procedural Textures

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

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

Procedural Texture Effects

- simple marble

function boring_marble(point)
  x = point.x;
  return marble_color(sin(x));
  // marble_color maps scalars to colors
Perlin Noise: Procedural Textures
- several good explanations
  - FCG Section 10.1
  - http://www.noisemachine.com/talk1
  - http://freeimage.us/m/huge alist/2/model/m_perlin.htm

Perlin Noise: Coherency
- smooth not abrupt changes

Perlin Noise: Turbulence
- multiple feature sizes
  - add scaled copies of noise

Perlin Noise: Turbulence
- multiple feature sizes
  - add scaled copies of noise

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

function turbulence(p)
    t = 0; scale = 1;
    while (scale > pixelsize) {
        t += abs(Noise(p/scale)*scale);
        scale/=2;
    }
    return t;
Interpolating Textures
- nearest neighbor
- bilinear
- hermite

Optimization
- save memory and time
- conceptually:
  - 2D or 3D grid
  - populate with random number generator
- actually:
  - precompute two 1D arrays of size n (typical size 256)
    - random unit vectors
    - permutation of integers 0 to n-1
  - lookup
    - $g(i, j, k) = G_k(i + P_j(j + P_k(i) \mod n) \mod n)$

Procedural Approaches

Vector Offsets From Grid
- weighted average of gradients
- random unit vectors

Perlin Marble
- use turbulence, which in turn uses noise:
  
  ```
  function marble(point)
  x = point.x + turbulence(point);
  return marble_color(sin(x))
  ```

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

Page 5
Fractal Landscapes
- fractals: not just for “showing math”
  - triangle subdivision
  - vertex displacement
  - recursive until termination condition

http://www.fractal-landscapes.co.uk/images.html

Self-Similarity
- infinite nesting of structure on all scales

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

D = log(N)/log(r) D = log(4)/log(3) = 1.26
http://www.vanderbilt.edu/AnS/psychology/cogsci/chass/workshop/Fractals.html

Language-Based Generation
- L-Systems: after Lindenmayer
  - Koch snowflake: F : > FLFRRFLF
  - F: forward, R: right, L: left

- Mariano’s Bush:
  \[ F=FF[-F+F+F]+[+F-F-F] \]
  - angle 16

http://spanky.triumf.ca/www/fractint/sys/plants.html

1D: Midpoint Displacement
- divide in half
- randomly displace
- scale variance by half

http://www.gameprogrammer.com/fractal.html

2D: Diamond-Square
- diamond step
  - generate a new value at square midpoint
    - average corner values + random amount
    - gives diamonds when have multiple squares in grid

- square step
  - generate new value at diamond midpoint
    - average corner values + random amount
    - gives squares again in grid
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, explosions
- group dynamics: behavioral
  - birds/bats flock, fish school, human crowd, dinosaur/elephant stampede

Particle Systems Demos
- general particle systems
  - http://www.wondertouch.com
- boids: bird-like objects
  - http://www.red3d.com/cwr/boids/

Particle Life Cycle
- generation
  - randomly within “fuzzy” location
  - initial attribute values: random or fixed
- dynamics
  - attributes of each particle may vary over time
    - color darker 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

Procedural Approaches Summary
- Perlin noise
- fractals
- L-systems
- particle systems
- not at all a complete list!
  - big subject: entire classes on this alone