Textures III

Week 10, Wed Mar 24

http://www.ugrad.cs.ubc.ca/~cs314/Vjan2010
News

• signup sheet for P3 grading
  • Mon/today/Fri signups in class
  • or send email to dingkai AT cs
    • by 48 hours after the due date or you'll lose marks

• (P4 went out Monday)
Review: Basic OpenGL Texturing

• setup
  • generate identifier: glGenTextures
  • load image data: glTexImage2D
  • set texture parameters (tile/clamp/...): glTexParameteri
  • set texture drawing mode (modulate/replace/...): glTexEnvf

• drawing
  • enable: glEnable
  • bind specific texture: glBindTexture
  • specify texture coordinates before each vertex: glTexCoord2f
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: MIP-mapping

- image pyramid, precompute averaged versions

Without MIP-mapping

With MIP-mapping

\(^5\)
Texture Parameters

- in addition to color can control other material/object properties
  - surface normal (bump mapping)
  - reflected color (environment mapping)
Bump Mapping: Normals As Texture

• object surface often not smooth – to recreate correctly need complex geometry model
• can control shape “effect” by locally perturbing surface normal
  • random perturbation
  • directional change over region
Bump Mapping

Original surface

A bump map
Bump Mapping

$O'(u)$
Lengthening or shortening $O(u)$ using $B(u)$

$N'(u)$
The vectors to the ‘new’ surface
Embossing

• at transitions
  • rotate point’s surface normal by $\theta$ or $-\theta$
Displacement Mapping

• bump mapping gets silhouettes wrong
  • shadows wrong too
• change surface geometry instead
  • only recently available with realtime graphics
  • need to subdivide surface
Environment Mapping

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

- 6 planar textures, sides of cube
  - point camera in 6 different directions, facing out from origin
Cube Mapping
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 3$^\text{rd}$ 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

- 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…)

![Examples of volumetric textures](image-url)
Volumetric Bump Mapping

Marble

Bump
Volumetric Texture Principles

• 3D function \( \rho(x, y, z) \)
• 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
Procedural Approaches
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

```plaintext
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://freespace.virgin.net/hugo.elias/models/m_perlin.htm
    • http://www.robo-murito.net/code/perlin-noise-math-faq.html

http://mrl.nyu.edu/~perlin/planet/
Perlin Noise: Coherency

- smooth not abrupt changes

coherent

white noise
Perlin Noise: Turbulence

- multiple feature sizes
  - add scaled copies of noise

![Graphs showing Turbulence with different amplitudes and frequencies]
Perlin Noise: Turbulence

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

- multiple feature sizes
  - add scaled copies of noise

```plaintext
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
Vector Offsets From Grid

- weighted average of gradients
  - random unit vectors

\[
g(x_1, y_1) \quad g(x_0, y_1) \quad g(x_1, y_0) \quad g(x_0, y_0)
\]
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
    • \( g(i, j, k) = G[ (i + P[ (j + P[k]) \mod n ]) \mod n ] \)
Perlin Marble

- use turbulence, which in turn uses noise:

```python
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
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

- $D = \frac{\log(N)}{\log(r)}$
  - $N$ = measure, $r$ = subdivision scale
- Hausdorff dimension: noninteger
  - $D = \frac{\log(4)}{\log(3)} = 1.26$

coastline of Britain

Koch snowflake

http://www.vanderbilt.edu/AnS/psychology/cogsci/chaos/workshop/Fractals.html
Language-Based Generation

- **L-Systems:** after Lindenmayer
  - Koch snowflake: $F ::= \text{FLFRRFLF}$
    - $F$: forward, $R$: right, $L$: left
  - Mariano’s Bush:
    \[ F = FF - [-F + F + F] + [+F - F - F] \]
    - angle 16

http://spanky.triumf.ca/www/fractint/lsys/plants.html
1D: Midpoint Displacement

- divide in half
- randomly displace
- scale variance by half

http://www.gameprogrammer.com/fractal.html
2D: Diamond-Square

- fractal terrain with diamond-square approach
  - generate a new value at midpoint
  - average corner values + random displacement
  - scale variance by half each time
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