Reading for Last Time and Today

- FG Chap 11 Texture Mapping
  - except 11.8
- RB Chap Texture Mapping
- FCG Sect 16.6 Procedural Techniques
- FCG Sect 16.7 Groups of Objects

Corrected Correction: HSI/HSV and RGB

- HSI: \[ S = 1 - \frac{\min(R,G,B)}{I} \]
- HSV: \[ S = 1 - \frac{\min(R,G,B)}{V} \]

News

- Project 3 grading slot signup
  - Mon 11-12
  - Tue 10-12:30, 4-6
  - Wed 11-12, 2:30-4

Review: Invisible Primitives

- why might a polygon be invisible?
  - polygon outside the field of view / frustum
  - solved by clipping
  - polygon is back-facing
  - solved by backface culling
  - polygon is occluded by object(s) nearer the viewpoint
  - solved by hidden surface removal

Review: Fractional Texture Coordinates

- action when s or t is outside [0…1] interval
- tiling
- clamping
- functions
- replace/decal
- modulate
- blend
- texture matrix stack

Basic OpenGL Texturing

- create a texture object and fill it with texture data:
  - glColorTextures( num, &indices )
  - set texture parameters with glTexParameter( Texture parameter, parameter value, ... )
  - assign texture to object
    - with glBindTexture( Texture target, texture number )
    - use glTexImage2D( GL_TEXTURE_2D, ... ) to specify parameters when applying the texture
  - use glTexCoord( s, t ) at vertex with ( s, t ) at vertex with

Low-Level Details

- large range of functions for controlling layout of texture data
- specify how the data in your image is arranged
- e.g. with glBindTexture( GL_TEXTURE_2D, ... ) to specify parameters when applying the texture
- textures must be square and size a power of 2
- common sizes are 32x32, 64x64, 256x256
- smaller textures are more memory-efficient, and there is a finite amount of texture memory on graphics cards
- ok to use texture template sample code for project 4

Texture Objects and Binding

- texture object
  - an OpenGL data type that keeps textures resident in memory and
  - provides identifiers to easily access them
  - provides efficiency gains over having to repeatedly load and
  - reload a texture
  - you can prioritize textures to keep in memory
- OpenGL uses least recently used (LRU) if no priority is assigned
- texture binding
  - which texture to use right now
  - switch between preloaded textures

Technical Details

- texture image: 2D array of color values (texels)
- assigning texture coordinates ( s, t ) at vertex with
  - object coordinates ( x, y, z )
- use interpolated ( s, t ) for texel lookup at each pixel
- use texture matrix stack
- ok to use texture template sample code for project 4

Texture Objects and Binding

- texture object
  - an OpenGL data type that keeps textures resident in memory and
  - provides identifiers to easily access them
  - provides efficiency gains over having to repeatedly load and
  - reload a texture
  - you can prioritize textures to keep in memory
- OpenGL uses least recently used (LRU) if no priority is assigned
- texture binding
  - which texture to use right now
  - switch between preloaded textures

Technical Details

- texture image: 2D array of color values (texels)
- assigning texture coordinates ( s, t ) at vertex with
  - object coordinates ( x, y, z )
- use interpolated ( s, t ) for texel lookup at each pixel
- use texture matrix stack
- ok to use texture template sample code for project 4

Texture Objects and Binding

- texture object
  - an OpenGL data type that keeps textures resident in memory and
  - provides identifiers to easily access them
  - provides efficiency gains over having to repeatedly load and
  - reload a texture
  - you can prioritize textures to keep in memory
- OpenGL uses least recently used (LRU) if no priority is assigned
- texture binding
  - which texture to use right now
  - switch between preloaded textures

Technical Details

- texture image: 2D array of color values (texels)
- assigning texture coordinates ( s, t ) at vertex with
  - object coordinates ( x, y, z )
- use interpolated ( s, t ) for texel lookup at each pixel
- use texture matrix stack
- ok to use texture template sample code for project 4
**Texture Mapping**
- texture coordinates
- specified at vertices
  
  \texttt{glTexCoord2f(s,t);}
  
  \texttt{glVertexf(x,y,z);}
- interpolated across triangle (like R,G,B,Z)
  - ...well not quite!

**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

**MIPmapping**
- use “image pyramid” to precompute
  averaged versions of the texture
- store whole pyramid in single block of memory

**Reconstruction**
- how to deal with:
  - pixels that are much larger than texels?
  - apply filtering, “averaging”
  
  \[
  \frac{0}{1/2/1/2} \]
  
  - pixels that are much smaller than texels?
  - interpolate

**MIPmaps**
- multum in parvo – many things in a small place
- precompute a series of prefiltered texture maps of decreasing
  resolutions
- requires more texture storage
- avoid shimmering and flashing as objects move
- \texttt{gluBuild2DMipmaps}:
  - automatically constructs a family of textures from original
  texture size down to 1x1
  - only 1/3 more space required

**Texture Coordinate Interpolation**
- perspective correct interpolation
  - \(a, b, \gamma\) barycentric coordinates of a point \(P\) in a triangle
  - \(s_0, s_1, s_2\):
    - texture coordinates of vertices
    - \(w_0, w_1, w_2\):
      - homogeneous coordinates of vertices
  - \(s = \frac{a \cdot x_s + b \cdot y_s + \gamma \cdot z_s}{w_0 + w_1 + w_2}\)

**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
  - apply filtering, “averaging”

**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
      - \texttt{GL_SPHERE_MAP, GL_CUBE_MAP}
  - others possible too

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

**MIPmap storage**
- only 1/3 more space required

**Texture Coordinate Interpolation**
- perspective correct interpolation
  - \(a, b, \gamma\) barycentric coordinates of a point \(P\) in a triangle
  - \(s_0, s_1, s_2\):
    - texture coordinates of vertices
    - \(w_0, w_1, w_2\):
      - homogeneous coordinates of vertices
  - \(s = \frac{a \cdot x_s + b \cdot y_s + \gamma \cdot z_s}{w_0 + w_1 + w_2}\)
**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;
```

**Cube Mapping**
- direction of reflection vector $r$ selects the face of the cube to be indexed
  - coordinate 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

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

**Cube Mapping**
- 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: Procedural Textures**
- several good explanations
  - FCG Section 10.1
  - http://mrl.nyu.edu/~perlin/planet/

**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 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
```

**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...)

**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

**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...)

**Procedural Texture Effects**
- several good explanations
  - FCG Section 10.1
  - http://www.noisemachine.com/talk1
  - http://freeimage.virgin.net/hugo.elias/models/m_perlin.htm

**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

**Perlin Noise: Coherency**
- smooth not abrupt changes

**Perlin Noise: Coherency**
- coherent
- white noise

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

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

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

**Procedural Texture Effects**
- just three main ideas
  - nice interpolation
  - use vector offsets to make grid irregular
  - optimization
    - sneaky use of 1D arrays instead of 2D/3D one
Fractal Dimension

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

Procedural Approaches

- nearest neighbor
- bilinear
- hermite

Vector Offsets From Grid

- weighted average of gradients
- random unit vectors
- \((x_0, y_0), (x_1, y_1), g(x_0, y_0), g(x_1, y_1)\)

Interpolating Textures

- nearest neighbor
- bilinear
- hermite

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

Perlin Marble

- use turbulence, which in turn uses noise:
  - function marble(point)
    - \( x = \text{point}.x + \text{turbulence}(\text{point}) \)
    - return marble_color(sin(x))

Language-Based Generation

- L-Systems: after Lindenmayer
  - Koch snowflake: \( F \rightarrow FLFRRFLF \)
  - \( F \): forward, \( R \): right, \( L \): left
  - Mariano’s Bush: \( F = FF-[-F+F+F]-[+F-F-F] \)
    - angle 16

Fractal Landscapes

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

1D: Midpoint Displacement

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

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

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
    - \( g(i, j, k) = \{ i + P[j + P[k] \mod n] \} \mod n \)

Self-Similarity

- infinite nesting of structure on all scales