University of British Columbia
CPSC 314 Computer Graphics
Jan-Apr 2007

Tamara Munzner

Textures II

Week 9, Fri Mar 16

http://www.ugrad.cs.ubc.ca/~cs314/Vjan2007

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

Corrected Correction: HSI/HSV and RGB

- HSV/HSI conversion from RGB
  - hue same in both
  - value is max, intensity is average

$$H = \cos^{-1}\left[\frac{\frac{1}{2}[(R-G)+(R-B)]}{\sqrt{(R-G)^2+(R-B)(G-B)}}\right]$$

- HSI: $S = 1 - \frac{\min(R,G,B)}{I}$ $\quad I = \frac{R+G+B}{3}$
- HSV: $S = 1 - \frac{\min(R,G,B)}{V}$ $\quad V = \max(R,G,B)$

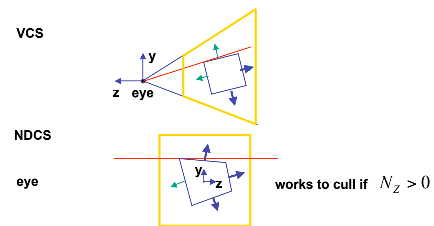
News

- H3 Q2: OK to use either HSV or HSI

News

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

Review: Back-face Culling



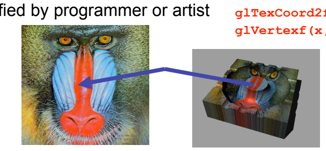
Review: Invisible Primitives

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

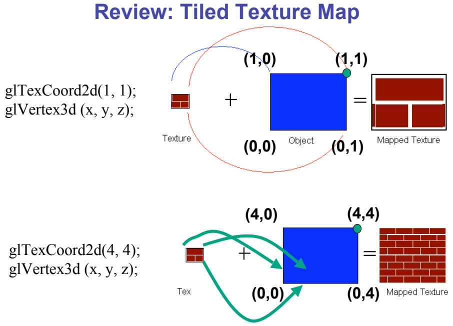
Review: Texture Coordinates

- texture image: 2D array of color values (texels)
- assigning texture coordinates (s,t) at vertex with object coordinates (x,y,z,w)
  - use interpolated (s,t) for texel lookup at each pixel
  - use value to modify a polygon's color
    - or other surface property
  - specified by programmer or artist

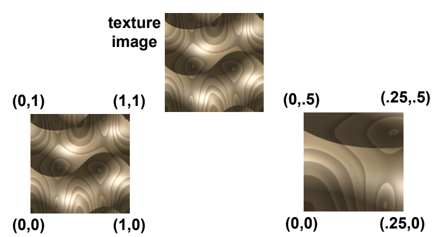
```
glTexCoord2f(s,t)
glVertexf(x,y,z,w)
```



Review: Tiled Texture Map



Review: Fractional Texture Coordinates



Review: Texture

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

```
glMatrixMode( GL_TEXTURE );
```


Texturing II

Texture Pipeline

(x, y, z)
Object position
(-2.3, 7.1, 17.7)

(s, t) Parameter space (0.32, 0.29) → (s', t') Transformed parameter space (0.52, 0.49) → Texel space (81, 74) → Texel color (0.9,0.8,0.7) → Final color (0.45,0.4,0.35)

Object color (0.5,0.5,0.5) → Final color (0.45,0.4,0.35)

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

Basic OpenGL Texturing

- create a texture object and fill it with texture data:
  - `glGenTextures(num, &indices)` to get identifiers for the objects
  - `glBindTexture(GL_TEXTURE_2D, identifier)` to bind
    - following texture commands refer to the bound texture
  - `glTexParameteri(GL_TEXTURE_2D, …, …)` to specify parameters for use when applying the texture
  - `glTexImage2D(GL_TEXTURE_2D, ….)` to specify the texture data (the image itself)
- enable texturing: `glEnable(GL_TEXTURE_2D)`
- state how the texture will be used:
  - `glTexEnvf(…)`
- specify texture coordinates for the polygon:
  - use `glTexCoord2f(s,t)` before each vertex:
    - `glTexCoord2f(0,0); glVertex3f(x,y,z);`

Low-Level Details

- large range of functions for controlling layout of texture data
  - state how the data in your image is arranged
  - e.g.: `glPixelStorei(GL_UNPACK_ALIGNMENT, 1)` tells OpenGL not to skip bytes at the end of a row
  - you must state how you want the texture to be put in memory: how many bits per "pixel", which channels,…
- textures must be square and size a power of 2
  - common sizes are 32x32, 64x64, 256x256
  - smaller uses less memory, and there is a finite amount of texture memory on graphics cards
- ok to use texture template sample code for project 4
  - http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=09

Texture Mapping

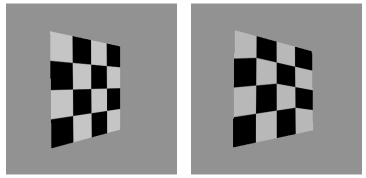
- texture coordinates
  - specified at vertices

```
glTexCoord2f(s,t);
glVertexf(x,y,z);
```

  - interpolated across triangle (like R,G,B,Z)
    - …well not quite!

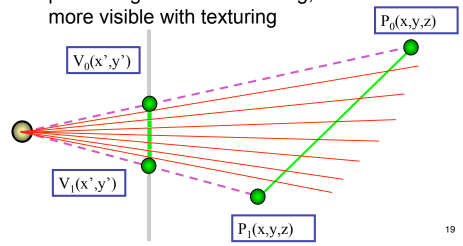
Texture Mapping

- texture coordinate interpolation
  - perspective foreshortening problem



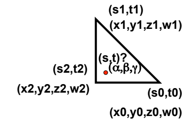
Interpolation: Screen vs. World Space

- screen space interpolation incorrect
  - problem ignored with shading, but artifacts more visible with texturing



Texture Coordinate Interpolation

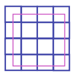
- perspective correct interpolation
  - α, β, γ :
    - barycentric coordinates of a point **P** in a triangle
  - *s0, s1, s2* :
    - texture coordinates of vertices
  - *w0, w1,w2* :
    - homogeneous coordinates of vertices



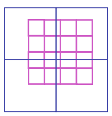
$$s = \frac{\alpha \cdot s_0 / w_0 + \beta \cdot s_1 / w_1 + \gamma \cdot s_2 / w_2}{\alpha / w_0 + \beta / w_1 + \gamma / w_2}$$

Reconstruction

- how to deal with:
  - pixels that are much larger than texels?
    - apply filtering, "averaging"

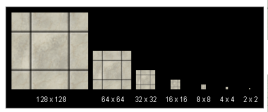


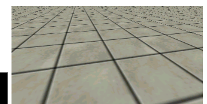
  - pixels that are much smaller than texels ?
    - interpolate



MIPmapping

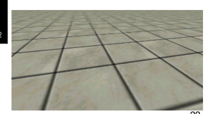
use "image pyramid" to precompute averaged versions of the texture





Without MIP-mapping

store whole pyramid in single block of memory

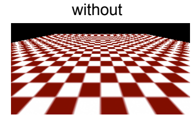


With MIP-mapping

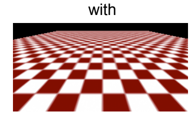
MIPmaps

- **multum in parvo** -- many things in a small place
  - prespecify a series of prefiltered texture maps of decreasing resolutions
  - requires more texture storage
  - avoid shimmering and flashing as objects move
- `gluBuild2DMipmaps`
  - automatically constructs a family of textures from original texture size down to 1x1

without

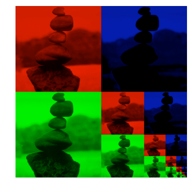


with



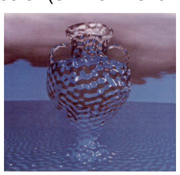
MIPmap storage

- only 1/3 more space required



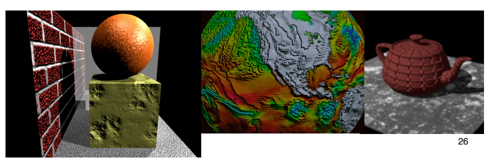
Texture Parameters

- in addition to color can control other material/object properties
  - surface normal (bump mapping)
  - reflected color (environment mapping)

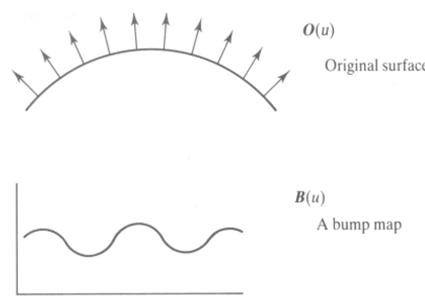


Bump Mapping: Normals As Texture

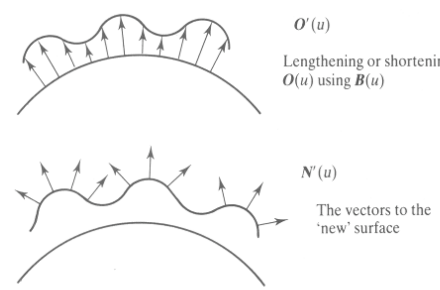
- object surface often not smooth – to recreate correctly need complex geometry model
- can control shape "effect" by locally perturbing surface normal
  - random perturbation
  - directional change over region



Bump Mapping

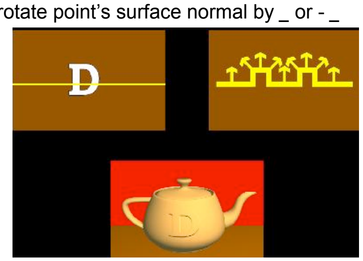


Bump Mapping



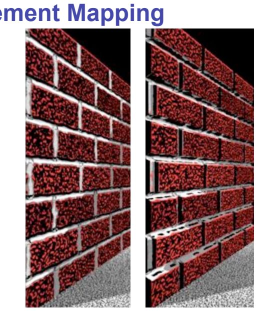
Embossing

- at transitions
  - rotate point's surface normal by _ or - _



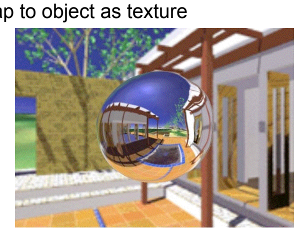
Displacement Mapping

- 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

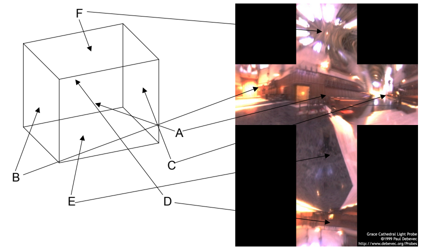
Sphere Mapping

- 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

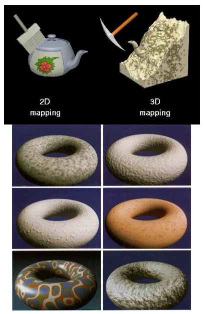


Cube Mapping

- 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 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

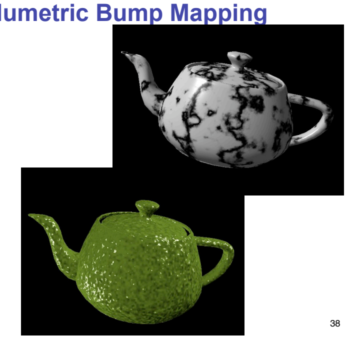
- define texture pattern over 3D domain - 3D space containing the object
  - texture function can be digitized or procedural
  - for each point on object compute texture from point location in space
- common for natural material/irregular textures (stone, wood,etc…)



Volumetric Bump Mapping

Marble

Bump



Volumetric Texture Principles

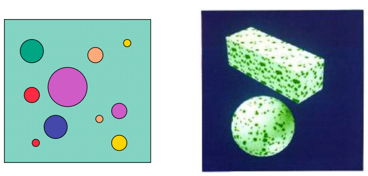
- 3D function $\rho(x,y,z)$
- texture space – 3D space that holds the texture (discrete or continuous)
- rendering: for each rendered point P(x,y,z) compute $\rho(x,y,z)$
- volumetric texture mapping function/space transformed with objects

Procedural Textures

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

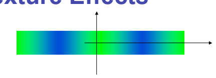
Procedural Texture Effects: Bombing

- 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

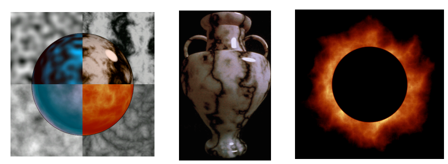
- simple marble



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

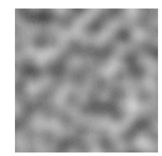


http://mrl.nyu.edu/~perlin/planet/

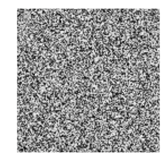
Perlin Noise: Coherency

- smooth not abrupt changes

coherent

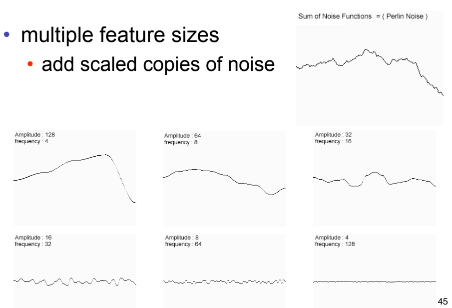


white noise



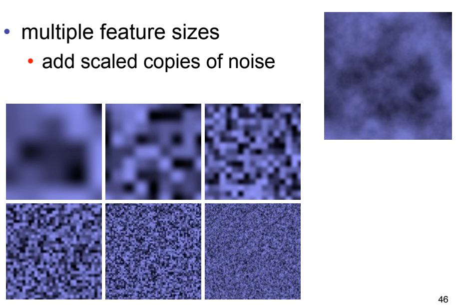
Perlin Noise: Turbulence

- multiple feature sizes
  - add scaled copies of noise



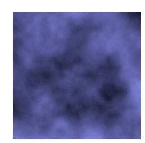
Perlin Noise: Turbulence

- multiple feature sizes
  - add scaled copies of noise



Perlin Noise: Turbulence

- multiple feature sizes
  - add scaled copies of noise



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

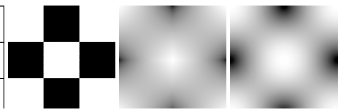

Generating Coherent Noise

- 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

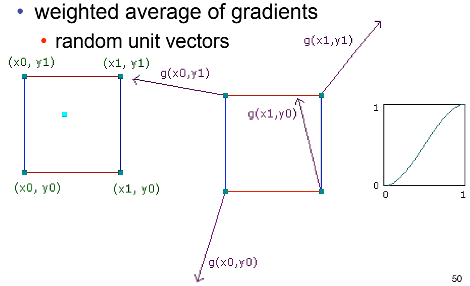
- nearest neighbor
- bilinear
- hermite

1	0	1
0	1	0
1	0	1



Vector Offsets From Grid

- weighted average of gradients
  - random unit vectors



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:

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

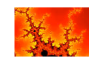
Procedural Approaches

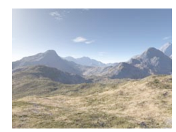
Procedural Modeling

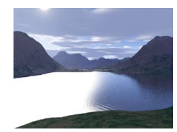
- textures, geometry
  - nonprocedural: explicitly stored in memory
- procedural approach
  - compute something on the fly
  - often less memory cost
  - visual richness
- fractals, particle systems, noise

Fractal Landscapes

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



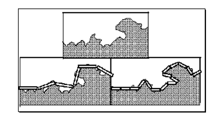




http://www.fractal-landscapes.co.uk/images.html

Self-Similarity

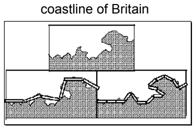
- infinite nesting of structure on all scales



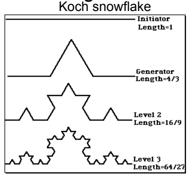
Fractal Dimension

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

coastline of Britain



Koch snowflake

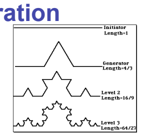


D = log(N)/log(r) D = log(4)/log(3) = 1.26

http://www.vanderbilt.edu/AnS/psychology/cogsci/chaos/workshop/Fractals.html

Language-Based Generation

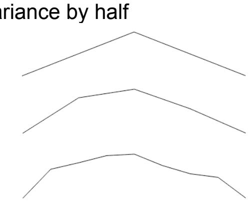
- L-Systems: after Lindenmayer
  - Koch snowflake: F :- FLFRRFLF
    - F: forward, R: right, L: left
  - Mariano’s Bush:
    F=FF-[-F+F+F]+[+F-F-F] }
    - angle 16



http://spanky.triumf.ca/www/fractint/lsys/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