University of British Columbia
CPSC 314 Computer Graphics
Jan-Apr 2013

Tamara Munzner

Textures

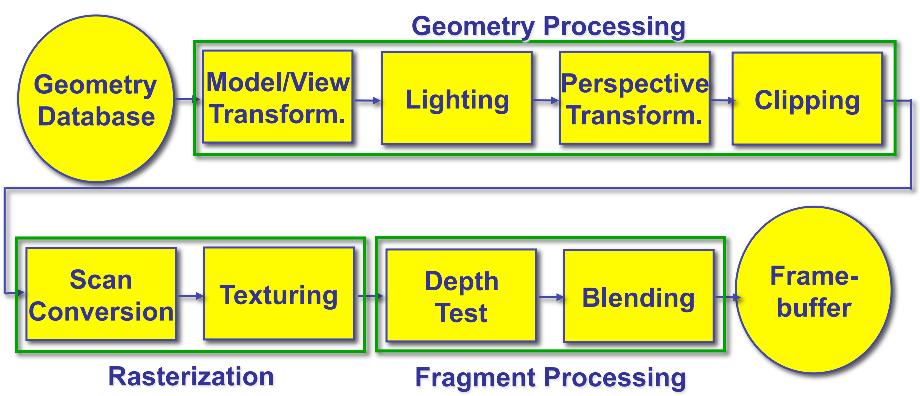
http://www.ugrad.cs.ubc.ca/~cs314/Vjan2013

Reading for Texture Mapping

- FCG Chap 11 Texture Mapping
  - except 11.7 (except 11.8, 2nd ed)
- RB Chap Texture Mapping

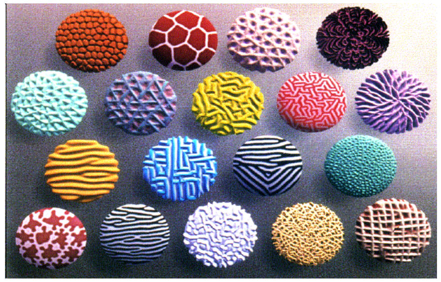
Texturing

Rendering Pipeline



Texture Mapping

- real life objects have nonuniform colors, normals
- to generate realistic objects, reproduce coloring & normal variations = **texture**
- can often replace complex geometric details

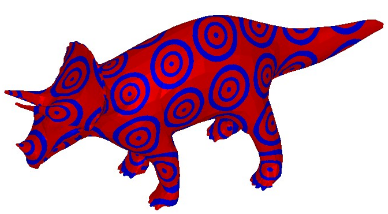


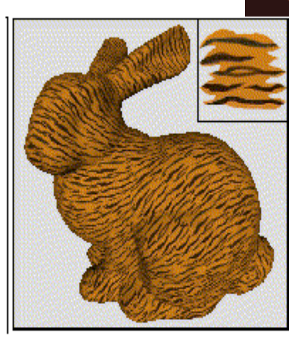
Texture Mapping

- introduced to increase realism
  - lighting/shading models not enough
- hide geometric simplicity
  - images convey illusion of geometry
  - map a brick wall texture on a flat polygon
  - create bumpy effect on surface
- associate 2D information with 3D surface
  - point on surface corresponds to a point in texture
  - “paint” image onto polygon

Color Texture Mapping

- define color (RGB) for each point on object surface
- two approaches
  - surface texture map
  - volumetric texture

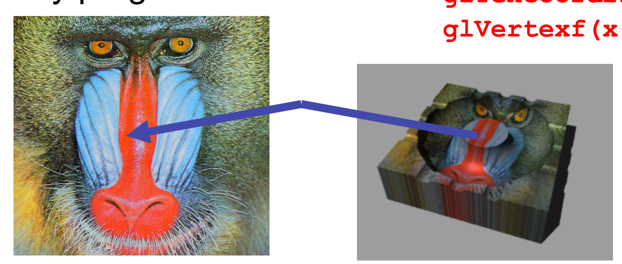




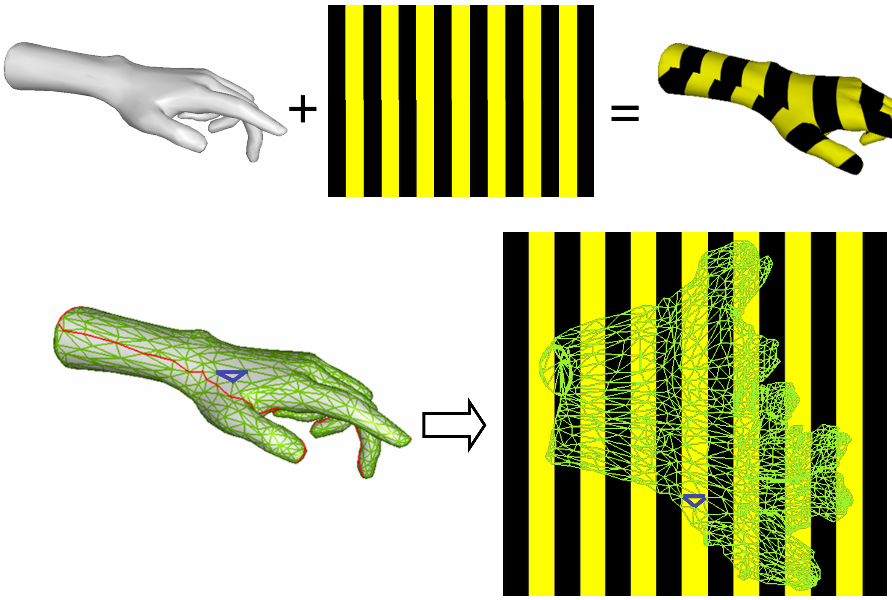
Texture Coordinates

- 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

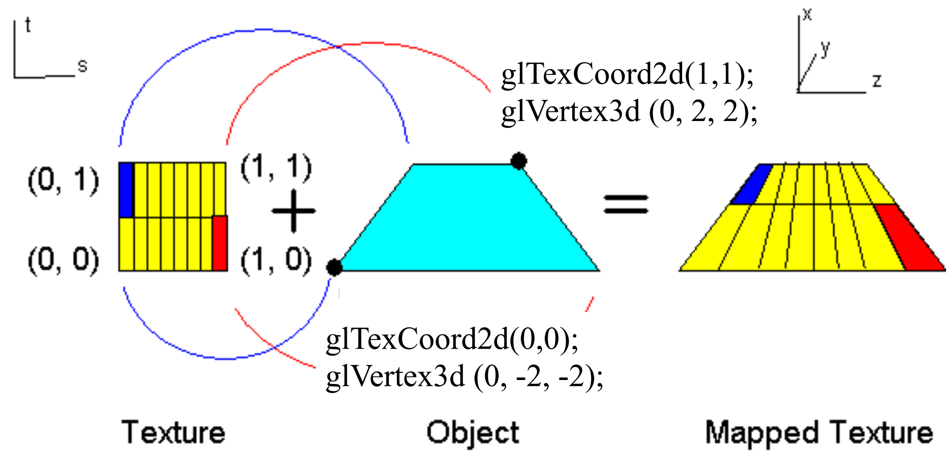
```
glTexCoord2f(s,t)
glVertexf(x,y,z,w)
```



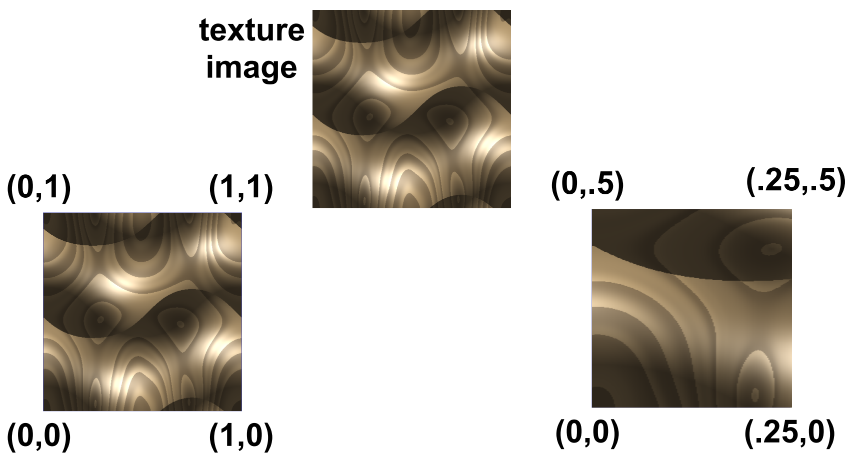
Texture Mapping Example



Example Texture Map



Fractional Texture Coordinates

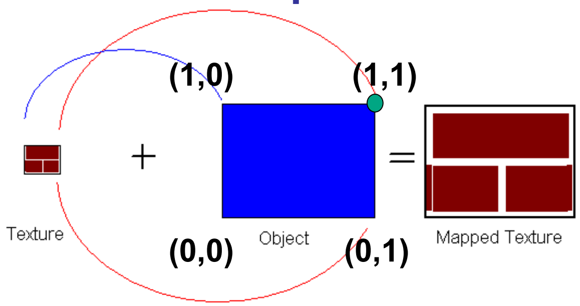


Texture Lookup: Tiling and Clamping

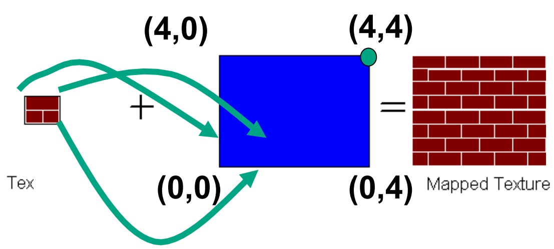
- what if s or t is outside the interval [0…1]?
- multiple choices
  - use fractional part of texture coordinates
    - cyclic repetition of texture to tile whole surface
      glTexParameteri(…, GL_TEXTURE_WRAP_S, GL_REPEAT, GL_TEXTURE_WRAP_T, GL_REPEAT, ...)
  - clamp every component to range [0…1]
    - re-use color values from texture image border
      glTexParameteri(…, GL_TEXTURE_WRAP_S, GL_CLAMP, GL_TEXTURE_WRAP_T, GL_CLAMP, ...)

Tiled Texture Map

glTexCoord2d(1, 1);
glVertex3d (x, y, z);



glTexCoord2d(4, 4);
glVertex3d (x, y, z);



Demo

- Nate Robbins tutors
  - texture

Texture Coordinate Transformation

- motivation
  - change scale, orientation of texture on an object
- approach
  - *texture matrix stack*
  - transforms specified (or generated) tex coords

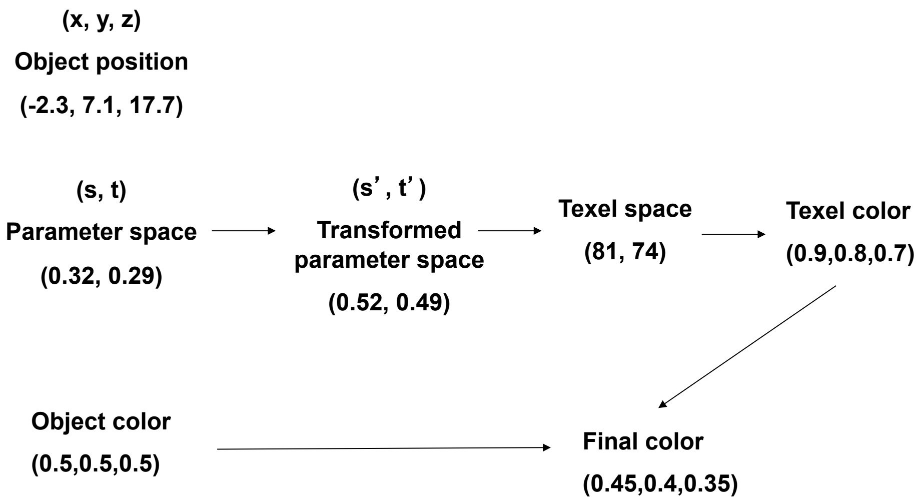
```
glMatrixMode( GL_TEXTURE );
glLoadIdentity();
glRotate();
   ...
```

  - more flexible than changing (s,t) coordinates
- [demo]

Texture Functions

- once have value from the texture map, can:
  - directly use as surface color: `GL_REPLACE`
    - throw away old color, lose lighting effects
  - modulate surface color: `GL_MODULATE`
    - multiply old color by new value, keep lighting info
    - texturing happens **after** lighting, not relit
  - use as surface color, modulate alpha: `GL_DECAL`
    - like replace, but supports texture transparency
  - blend surface color with another: `GL_BLEND`
    - new value controls which of 2 colors to use
    - indirection, new value not used directly for coloring
- specify with `glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, <mode>)`
- [demo]

Texture Pipeline



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

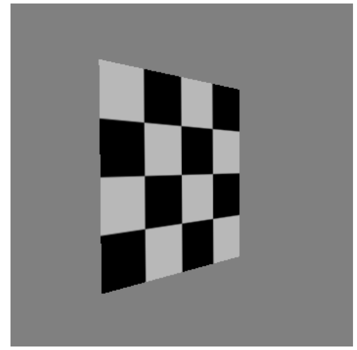
- texture coordinates
  - specified at vertices

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

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

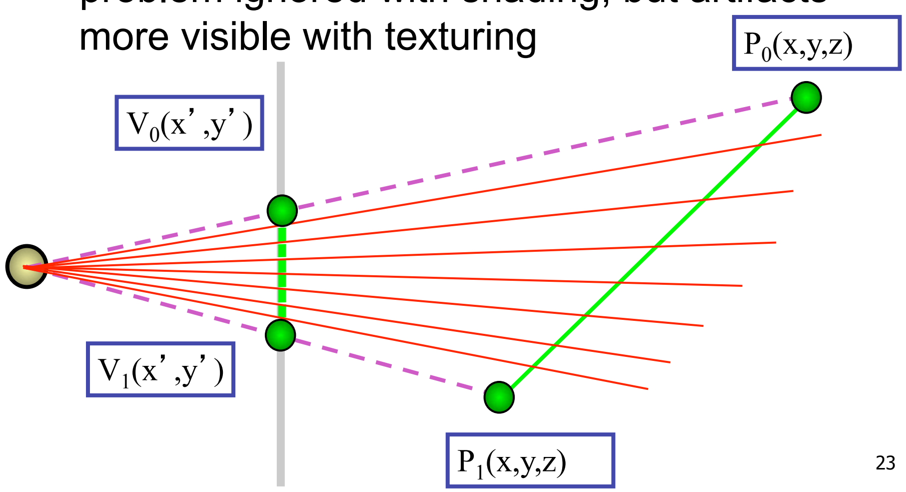
Texture Mapping

- texture coordinate interpolation
  - perspective foreshortening problem



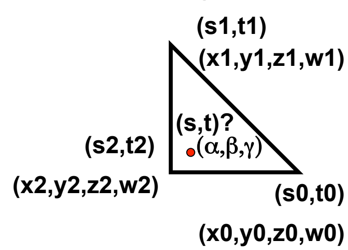

Interpolation: Screen vs. World Space

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



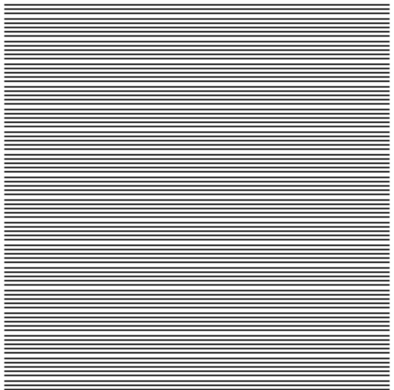
Texture Coordinate Interpolation

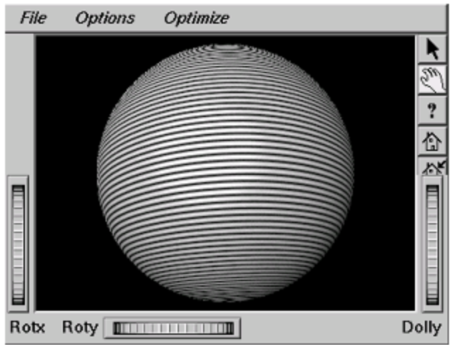
- perspective correct interpolation
  - α, β, γ :
    - barycentric coordinates of a point **P** in a triangle
  - *s0*, *s1*, *s2* :
    - texture coordinates of vertices
  - *w0*, *w1*,*w2* :
    - homogeneous coordinates of vertices



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

Reconstruction

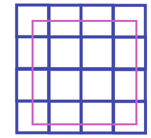




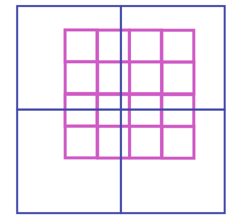
(image courtesy of Kiriakos Kutulakos, U Rochester)

Reconstruction

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

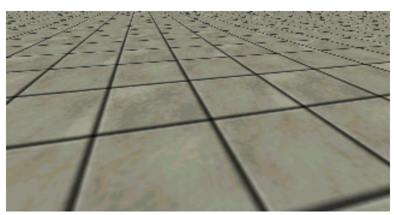


  - pixels that are much smaller than texels ?
    - interpolate

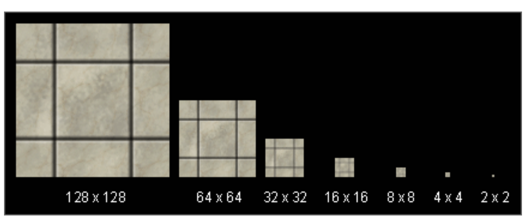


MIPmapping

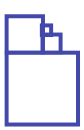
use “image pyramid” to precompute averaged versions of the texture

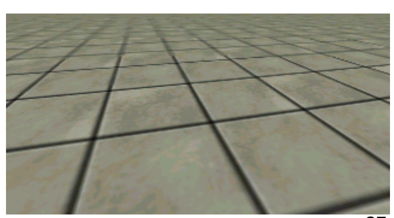


Without MIP-mapping



store whole pyramid in single block of memory



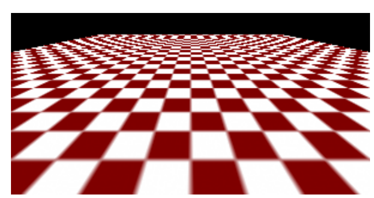


With MIP-mapping

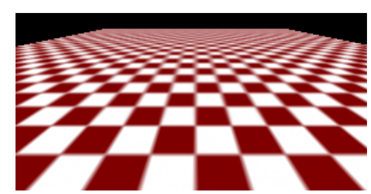
MIPmaps

- **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

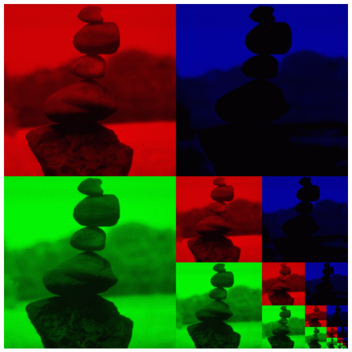


with



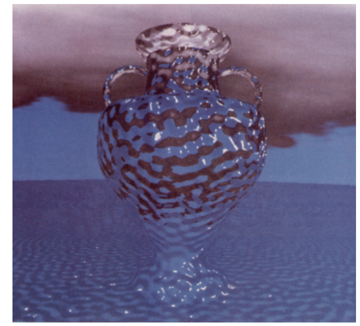
MIPmap storage

- only 1/3 more space required



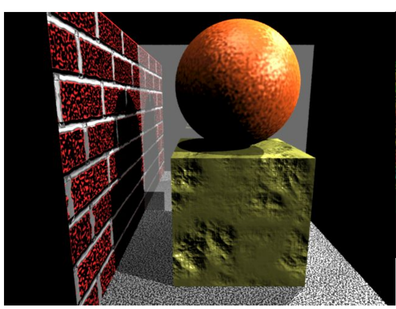
Texture Parameters

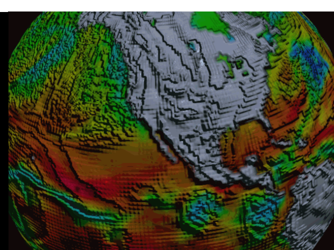
- in addition to color can control other material/ object properties
  - surface normal (bump mapping)
  - reflected color (environment mapping)

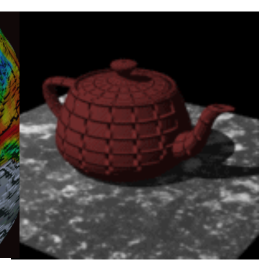


Bump Mapping: Normals As Texture

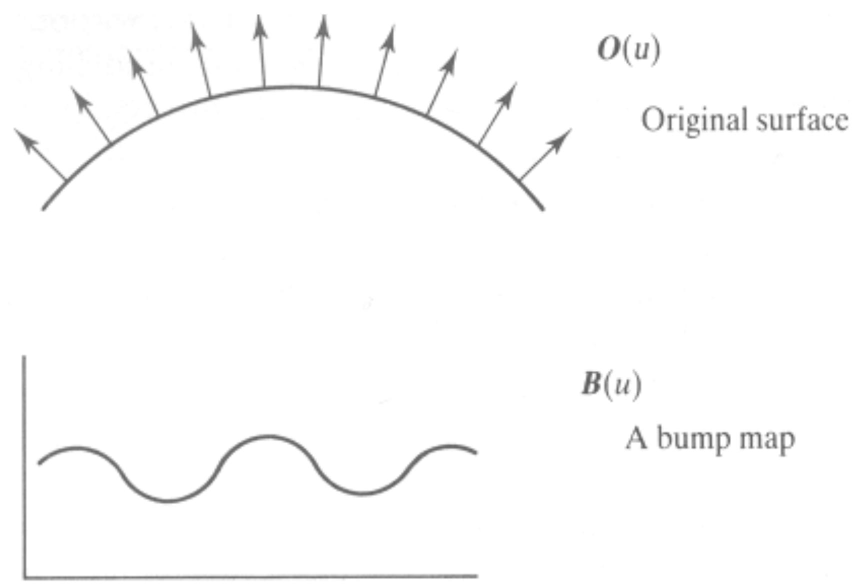
- object surface often not smooth – to recreate correctly need complex geometry model
- can control shape "effect" by locally perturbing surface normal
  - random perturbation
  - directional change over region



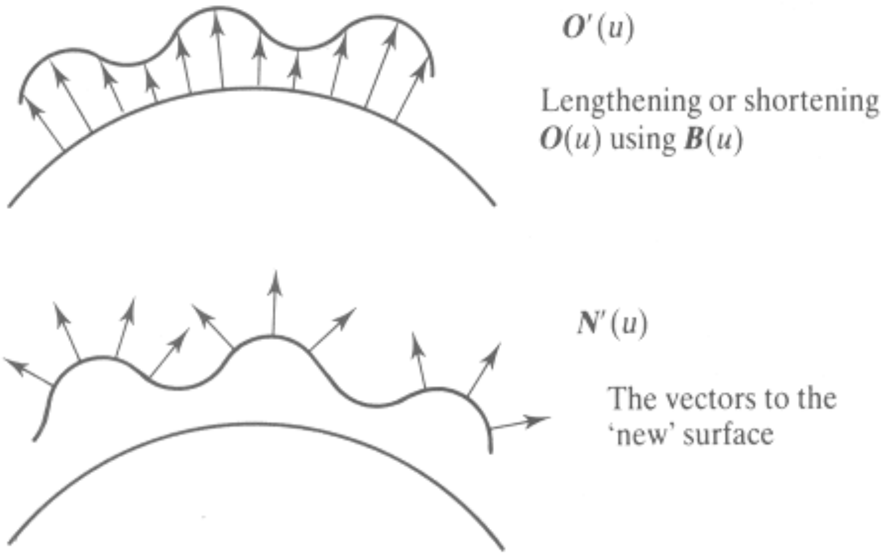




Bump Mapping

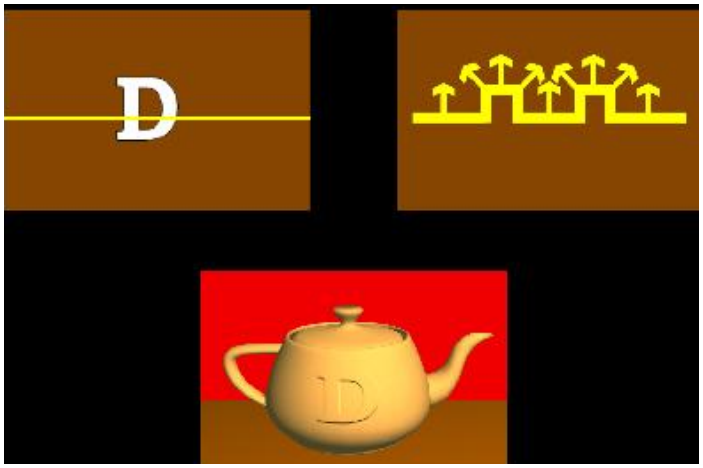


Bump Mapping



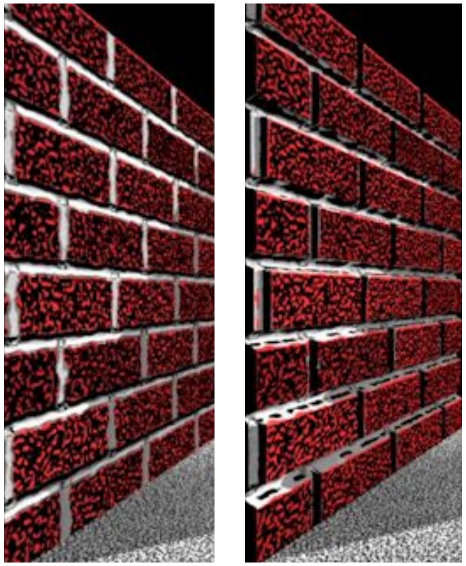
Embossing

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



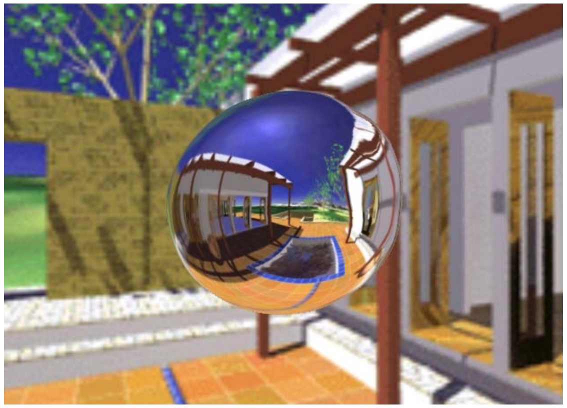
Displacement Mapping

- 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

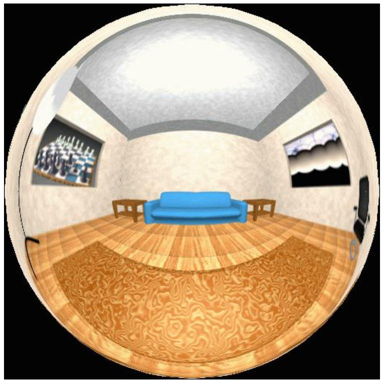


Environment Mapping

- 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

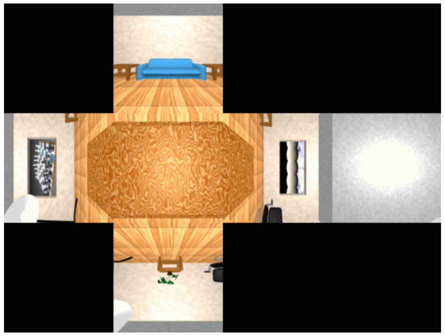
Sphere Mapping

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

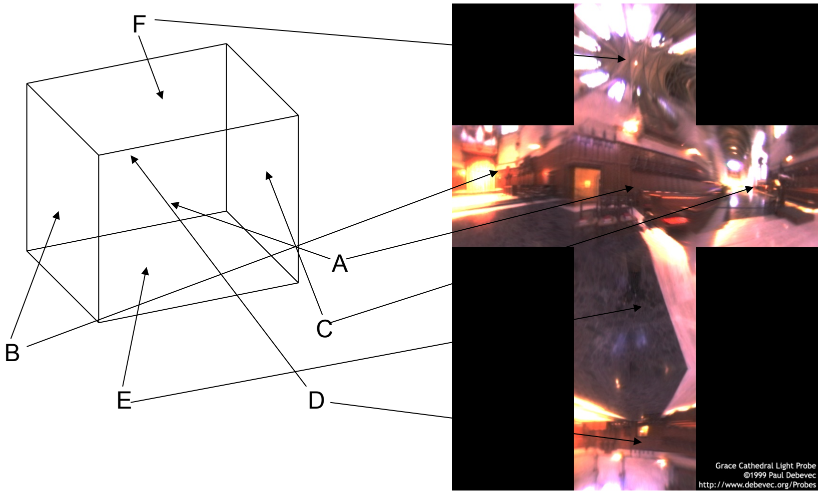


Cube Mapping

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



Cube Mapping

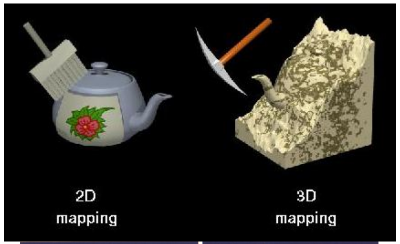


Cube Mapping

- 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

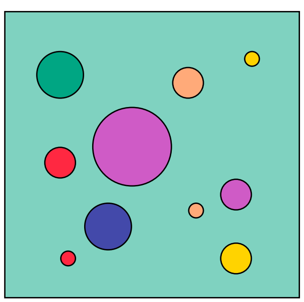
Procedural Approaches

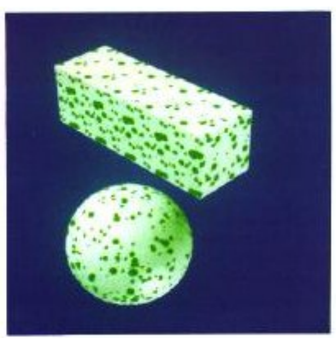
Procedural Textures

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

Procedural Texture Effects: Bombing

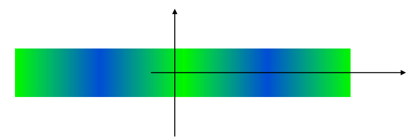
- randomly drop bombs of various shapes, sizes and orientation into texture space (store data in table)
  - for point P search table and determine if inside shape
    - if so, color by shape
    - otherwise, color by objects color





Procedural Texture Effects

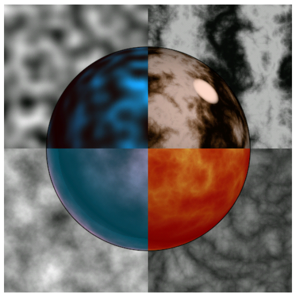
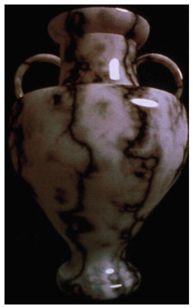
- simple marble



```
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

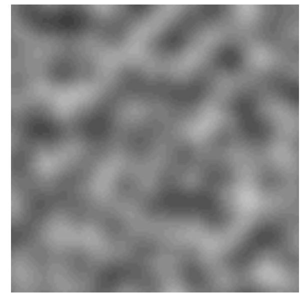




http://mrl.nyu.edu/~perlin/planet/

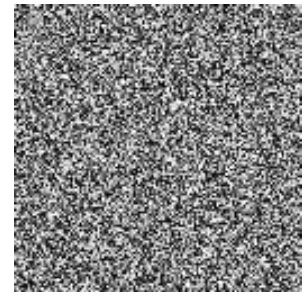
Perlin Noise: Coherency

- smooth not abrupt changes

coherent

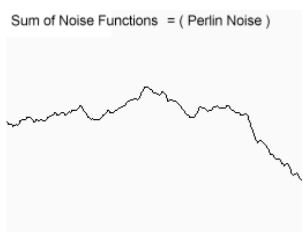


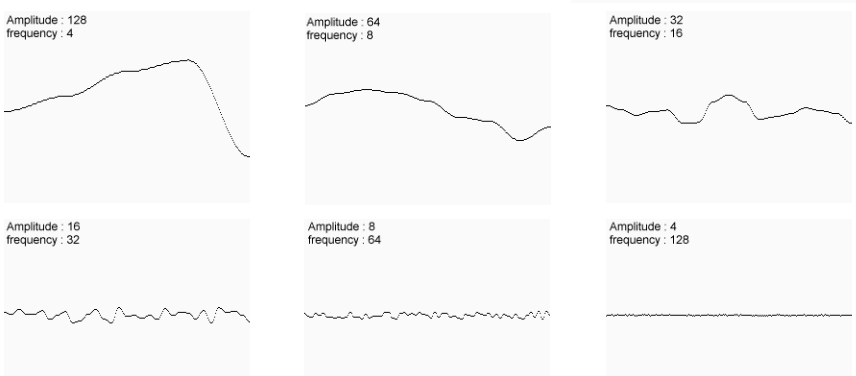
white noise



Perlin Noise: Turbulence

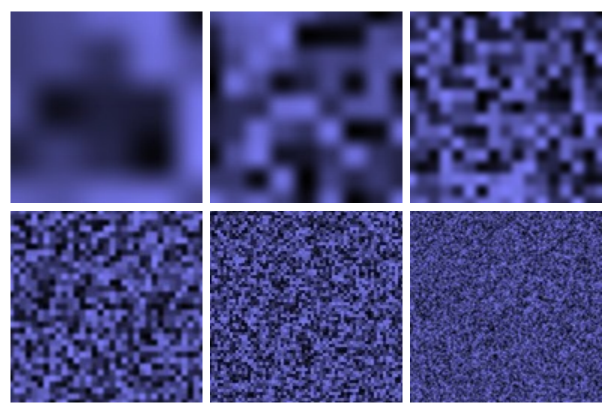
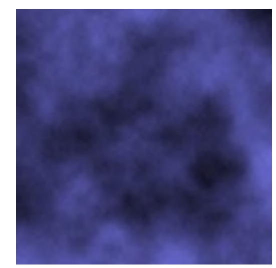
- multiple feature sizes
  - add scaled copies of noise





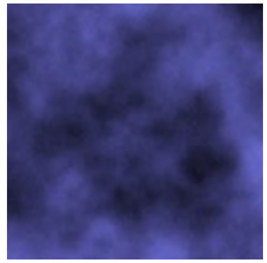
Perlin Noise: Turbulence

- multiple feature sizes
  - add scaled copies of noise

Perlin Noise: Turbulence

- multiple feature sizes
  - add scaled copies of noise



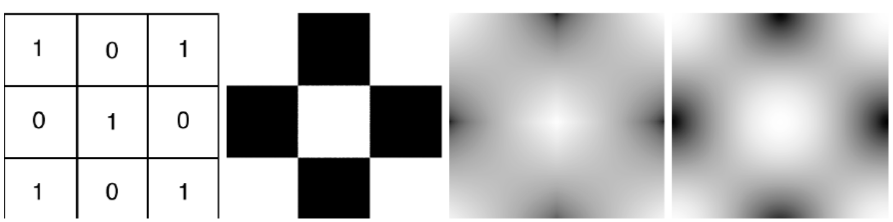
```
function turbulence(p)
     t = 0; scale = 1;
     while (scale > pixelsize) {
          t += abs(Noise(p/
scale)*scale);
          scale/=2;
     } return t;
```


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

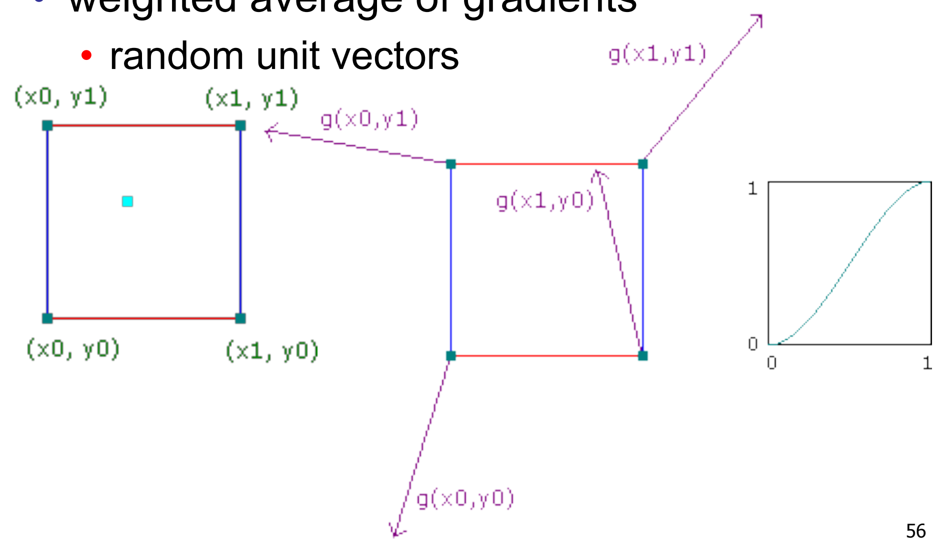
Interpolating Textures

- nearest neighbor
- bilinear
- hermite



Vector Offsets From Grid

- weighted average of gradients
  - random unit vectors



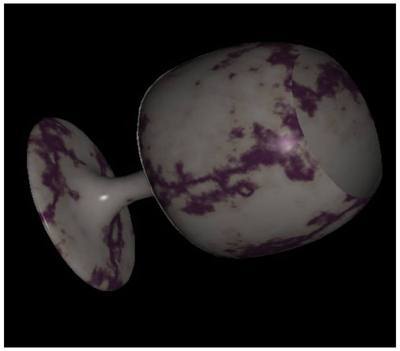
Optimization

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

Perlin Marble

- use turbulence, which in turn uses noise:

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

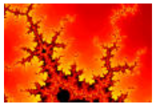



Procedural Modeling

- 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

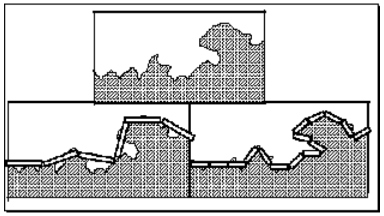
- fractals: not just for "showing math"
  - triangle subdivision
  - vertex displacement
  - recursive until termination condition



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

Self-Similarity

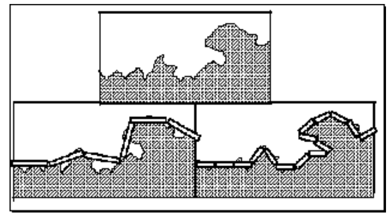
- infinite nesting of structure on all scales



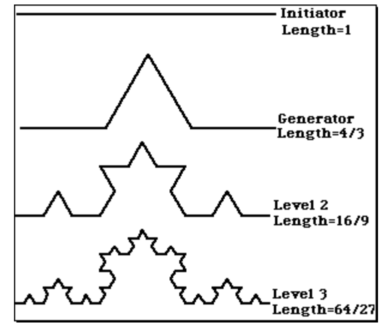
Fractal Dimension

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

coastline of Britain



Koch snowflake

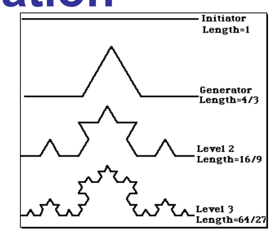


D = log(N)/log(r) D = log(4)/log(3) = 1.26

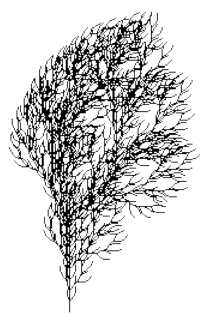
http://www.vanderbilt.edu/AnS/psychology/cogsci/chaos/workshop/Fractals.html

Language-Based Generation

- L-Systems: after Lindenmayer
  - Koch snowflake: F :- FLFRRFLF
    - F: forward, R: right, L: left



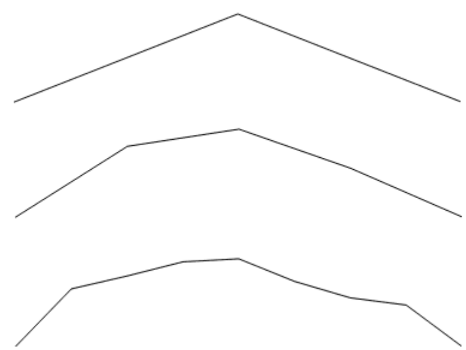
  - Mariano's Bush:
    F=FF-[-F+F+F]+[+F-F-F] }
    - angle 16



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

1D: Midpoint Displacement

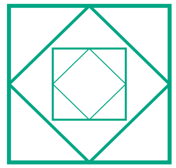
- divide in half
- randomly displace
- scale variance by half

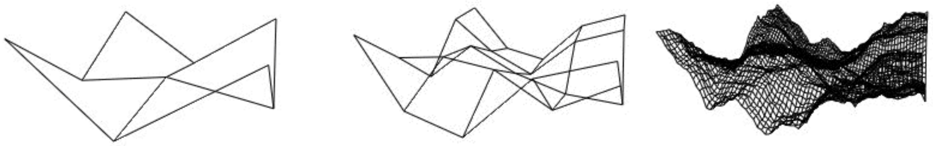


http://www.gameprogrammer.com/fractal.html

2D: Diamond-Square

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



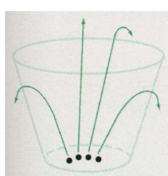


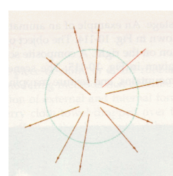
Particle Systems

- 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