News
- showing up for your project grading slot is **not** optional
- 5 people have missed their slot, without notifying the TA in advance of the need to change
- 2% penalty for no-shows for P3 and P4

Review: 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
- 3D function \( \rho(x,y,z) \)

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

Review: Perlin Noise
- coherency: smooth not abrupt changes
- turbulence: multiple feature sizes

Review: 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

Review: Environment 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

Review: Cube Environment Mapping

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

Self-Similarity
- infinite nesting of structure on all scales

Procedural Approaches II
- textures, geometry
  - nonprocedural: explicitly stored in memory
  - procedural approach
    - compute something on the fly
    - not load from disk
    - often less memory cost
    - visual richness
    - adaptable precision
- noise, fractals, particle systems

Fractal Dimension
- \( D = \log(N)/\log(r) \)
- Hausdorff dimension: noninteger
- coastline of Britain
- Koch snowflake

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

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

2D: Diamond-Square
- fractal terrain with diamond-square approach
  - generate a new value at midpoint
  - average corner values at midpoint
  - 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, explosion
• 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
  • flocking/swarming behavior
  • procedural motion
  • 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

Particle System Examples

• generation
  • randomly within “fuzzy” location
• dynamics
  • attributes of each particle may vary over time
  • 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 Life Cycle

• generation
  • randomly within “fuzzy” location
  • initial attribute values: random or fixed
• dynamics
  • attributes of each particle may vary over time
  • position: previous particle position + velocity + time
  • color darker as particle cools off after explosion
  • can also depend on other attributes
• 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

Particle System Examples

• generation
  • randomly within “fuzzy” location
• dynamics
  • attributes of each particle may vary over time
  • 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 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
• 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

Particle System Examples

• generation
  • randomly within “fuzzy” location
• dynamics
  • attributes of each particle may vary over time
  • 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 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
• 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
### Viewport
- small rectangle around cursor
  - change coord sys so fills viewport
  ![Viewport Diagram]
- why rectangle instead of point?
  - people aren’t great at positioning mouse
  - fits’ Law: time to acquire a target is function of the distance to and size of the target
  - allow several pixels of slop

### Hybrid Picking
- select/hit approach: fast, coarse
  - object-level granularity
  - exact intersection point
- manual: slow, precise
  - use select/hit to find object
  - then intersect ray with that object
- hybrid: both speed and precision
  - use select/hit to find object
  - then intersect ray with that object

### OpenGL Precision Picking Hints
- gluUnproject
  - transform window coordinates to object coordinates given current projection and modelview matrices
  - use to create ray from cursor location
  - call gluUnProject twice with same (x,y) mouse location
    - z = near: (x,y,0)
    - z = far: (x,y,1)
  - subtract near result from far result to get direction vector for ray
  - use this ray for line/polygon intersection

### Name Stack
- again, "names" are just integers
  - glInitNames()
- flat list
  - glLoadName(name)
- or hierarchy supported by stack
  - glPushName(name), glPopName
- can have multiple names per object

### Render Modes
- glRenderMode(mode)
  - GL_RENDER: normal color buffer
    - default
  - GL_SELECT: selection mode for picking
    - (GL_FEEDBACK: report objects drawn)

### Hit List
- glSelectBuffer(bufferSize, *buffer)
  - where to store hit list data
  - on hit, copy entire contents of name stack to output buffer.
  - hit record
    - number of names on stack
    - minimum and minimum depth of object vertices
      - depth lies in the z-buffer range [0,1]
      - multiplied by 2^32 -1 then rounded to nearest int

### Hit List Command
- glVertex3f(x,y,z)
- glSelectBuffer(bufferSize, *buffer)

### Name Stack
- again, "names" are just integers
- flat list
- or hierarchy supported by stack
- can have multiple names per object

### Hit List
- glSelectBuffer(bufferSize, *buffer)
  - where to store hit list data
  - on hit, copy entire contents of name stack to output buffer.
  - hit record
    - number of names on stack
    - minimum and minimum depth of object vertices
      - depth lies in the z-buffer range [0,1]
      - multiplied by 2^32 -1 then rounded to nearest int

### Hit List Command
- glVertex3f(x,y,z)
- glSelectBuffer(bufferSize, *buffer)

### Hybrid Picking
- select/hit approach: fast, coarse
  - object-level granularity
- manual ray intersection: slow, precise
  - exact intersection point
- hybrid: both speed and precision
  - use select/hit to find object
  - then intersect ray with that object

### OpenGL Precision Picking Hints
- gluUnproject
  - transform window coordinates to object coordinates given current projection and modelview matrices
  - use to create ray from cursor location
  - call gluUnProject twice with same (x,y) mouse location
    - z = near: (x,y,0)
    - z = far: (x,y,1)
  - subtract near result from far result to get direction vector for ray
  - use this ray for line/polygon intersection