Midterm 2: Wed Mar 26
• covering through Homework 3 material
  • MT1: transformations, some viewing
  • MT2 emphasis
    • some viewing
    • projections
    • color
    • rasterization
    • lighting/shading
    • advanced rendering (incl raytracing)
• graded H3 + solutions out Monday

News
• project 4 proposals due today 3pm
  • handin cs314 proj3.prop
  • or on paper in box
• proposal: your chance to get feedback from me
  • don’t wait to hear back from me to get started
  • you’ll hear from me soon if I see something dubious
  • not a contract, can change as you go

Midterm 2: Wed Mar 26
• closed book
• allowed to have
  • calculator
• one side of 8.5"x11" paper, handwritten
• write your name on it
• return it in with exam, you'll get it back
• have ID out and face up

News
• showing up for your project grading slot is not optional
• 2% penalty for no-shows
• signing up for your project grading slot is not optional
• 2% penalty for no-shows within two days of due date
• your responsibility to sign up for slot
• not ours to hunt you down if you chose to skip class on
  signup day
• we do make best effort to accommodate change requests via email to grader for that project

Review: Language-Based Generation
• L-Systems
  • F: forward, R: right, L: left
  • Koch snowflake:
    • F = FLFRRFLF
  • Mariano’s Bush:
    • F=FF-[F+F+F]+[+F-F-F]
    • angle 16
  • midpoint
  • glPushName (names)
  • for(int i = 0; i < 2; i++) {
    glPushName(i);
    for(int j = 0; j < 2; j++) {
      glPushMatrix();
      glPushName(j);
      glCallList();
      glPopMatrix();
    }
    glPopName();
  }
  • hierarchical names example
  • http://www.lighthouse3d.com/opengl/picking/

Review: Fractal Terrain
• 1D: midpoint displacement
  • divide in half, randomly displace
  • scale variance by half
• 2D: diamond-square
  • generate new value at midpoint
  • average corner values + random displacement
  • scale variance by half each time
• http://www.gameprogrammer.com/fractal.html

Review: Particle Systems
• changeable/fluid stuff
  • fire, steam, smoke, water, grass, hair, dust,
    waterfalls, fireworks, explosions, flocks
• life cycle
  • generation, dynamics, death
• rendering tricks
  • avoid hidden surface computations

Viewport
• small rectangle around cursor
  • change coords sys so fills viewport
• why rectangle instead of point?
  • people aren’t great at positioning mouse
  • Fitts’ Law: time to acquire a target is a function of the distance to and size of the target
  • allow several pixels of slop

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

Name Stack
• again, “names” are just integers
  • glListName() (names)
  • flat list
  • or hierarchy supported by stack
  • glPushName(), glPopName()
  • can have multiple names per object

Viewport
• nontrivial to compute
  • invert viewport matrix, set up new orthogonal projection
• simple utility command
  • glPushMatrix(x,y,w,h,viewport)
    • x,y: cursor point
    • w,h: sensitivity/slop (in pixels)
    • push old setup first, so can pop it later

Review: Picking Methods
• manual ray intersection
• bounding extents
• backbuffer coding

Review: Picking II

Review: Particle Systems
• changeable/fluid stuff
  • fire, steam, smoke, water, grass, hair, dust,
    waterfalls, fireworks, explosions, flocks
• life cycle
  • generation, dynamics, death
• rendering tricks
  • avoid hidden surface computations

Viewport
• small rectangle around cursor
  • change coord sys so fills viewport
• why rectangle instead of point?
  • people aren’t great at positioning mouse
  • Fitts’ Law: time to acquire a target is function of the distance to and size of the target
  • allow several pixels of slop
Collision Detection Applications

- determining if player hit wall/obstacle
- terrain following (floor), maze games
- separate: customize functions for each
- potential more efficient
- can avoid drawing unpickable objects

OpenGL Precision Picking Hints

- you must implement true 3D picking!
- you will not get credit if you just use 2D information

Hybrid Picking

- select/hit approach: fast, coarse
- object-level granularity
- check if ray cast from cursor position collides with any object in scene
- use select/hit to find object
- then intersect ray with that object

Collision Detection

- select/hit approach: fast, coarse
- object-level granularity
- manual ray intersection: slow, precise
- exact intersection point
- hybrid: both speed and precision

Accelerating Collision Detection

- two kinds of approaches (many others also)
- spatial data structures to localize
- used for both 2D and 3D
- used to accelerate many things, not just collision detection

Naive General Collision Detection

- for each object i containing polygons p
  - test for intersection with object j containing polygons q
  - for polyhedral objects, test if object i penetrates surface of j
  - test if vertices of j straddle polygon q of j
    - if straddle, then test intersection of polygon q with polygon p of object i
  - very expensive! O(n^2)

Collision/Acceleration

- two kinds of approaches (many others also)
- spatial data structures to localize
- used for both 2D and 3D
- used to accelerate many things, not just collision detection

Stupid Algorithm

- on each step, do a general mesh-to-mesh intersection test to find out if the player intersects the wall
- if they do, refuse to allow the player to move
- problems with this approach? how can we improve:
  - in response?
  - in speed?
**Collision Proxies**
- **proxy**: something that takes place of real object
- cheaper than general mesh-mesh intersections
- collision proxy (bounding volume) is piece of geometry used to represent complex object for purposes of finding collision
  - if proxy collides, object is said to collide
  - collision points mapped back onto original object
- good proxy: cheap to compute collisions for, tight fit to the real geometry
- common proxies: sphere, cylinder, box, ellipsoid
- consider: fat player, thin player, rocket, car ...

**Trade-off in Choosing Proxies**
- increasing complexity & tightness of fit
- decreasing cost of (overlap tests + proxy update)

- AABB: axis aligned bounding box
- OBB: oriented bounding box, arbitrary alignment
- k-dops – shapes bounded by planes at fixed orientations
  - discrete orientation polytope

**Pair Reduction**
- want proxy for any moving object requiring collision detection
- before pair of objects tested in any detail, quickly test if proxies intersect
- when lots of moving objects, even this quick bounding sphere test can take too long: $N^2$ times if there are $N$ objects
- reducing this $N^2$ problem is called **pair reduction**
- pair testing isn’t a big issue until $N > 50$ or so...

**Spatial Data Structures**
- can only hit something that is close
- spatial data structures tell you what is close to object
- uniform grid, octrees, kd-trees, BSP trees
- bounding volume hierarchies
- OBB trees
- for player-wall problem, typically use same spatial data structure as for rendering
  - BSP trees most common

**Uniform Grids**
- axis-aligned
- divide space uniformly

**Quadtrees/Octrees**
- axis-aligned
- subdivide until no points in cell

**KD Trees**
- axis-aligned
- subdivide in alternating dimensions

**BSP Trees**
- planes at arbitrary orientation

**Bounding Volume Hierarchies**

**OBB Trees**

**Related Reading**
- Real-Time Rendering
  - Tomas Moller and Eric Haines
  - on reserve in CICSR reading room

**Acknowledgement**
- slides borrow heavily from
  - Stephen Chenney, (UWisc CS679)
- slides borrow lightly from
  - Steve Rotenberg, (UCSD CSE169)
    - http://graphics.ucsd.edu/courses/cse169_w05/CSE169_17.ppt