Picking II, Collision and Acceleration

Week 10, Fri Mar 23

News

• showing up for your project grading slot is **not** optional
  • 2% penalty for noshow

• signing up for your project grading slot is **not** optional
  • 2% penalty for nosignups 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
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

- 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
Midterm 2: Wed Mar 26

- closed book
- allowed to have
  - calculator
  - one side of 8.5”x11” paper, handwritten
    - write your name on it
    - turn it in with exam, you’ll get it back
- have ID out and face up
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

[Image of L-Systems and Koch Snowflake]

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

• manual ray intersection

• bounding extents

• backbuffer coding
Picking II
Select/Hit

• use small region around cursor for viewport
• assign per-object integer keys (names)
• redraw in special mode
• store hit list of objects in region
• examine hit list

• OpenGL support
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
Viewport

• nontrivial to compute
  • invert viewport matrix, set up new orthogonal projection
• simple utility command
  • gluPickMatrix(x,y,w,h,viewport)
    • x,y: cursor point
    • w,h: sensitivity/slop (in pixels)
  • push old setup first, so can pop it later
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
  - `glInitNames()`
- flat list
  - `glLoadName(name)`
- or hierarchy supported by stack
  - `glPushName(name), glPopName`
  - can have multiple names per object
Hierarchical Names Example

```c
for(int i = 0; i < 2; i++) {
    glPushName(i);
    for(int j = 0; j < 2; j++) {
        glPushMatrix();
        glPushName(j);
        glTranslatef(i*10.0, 0, j*10.0);
        glPushName(HEAD);
        glCallList(snowManHeadDL);
        glLoadName(BODY);
        glCallList(snowManBodyDL);
        glPopName();
        glPopName();
        glPopMatrix();
    }
    glPopName();
}
```

http://www.lighthouse3d.com/opengl/picking/
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
Integrated vs. Separate Pick Function

• integrate: use same function to draw and pick
  • simpler to code
  • name stack commands ignored in render mode

• separate: customize functions for each
  • potentially more efficient
  • can avoid drawing unpickable objects
Select/Hit

• advantages
  • faster
    • OpenGL support means hardware acceleration
    • avoid shading overhead
  • flexible precision
    • size of region controllable
  • flexible architecture
    • custom code possible, e.g. guaranteed frame rate

• disadvantages
  • more complex
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 into scene from cursor location
  - call gluUnProject twice with same (x,y) mouse location
    - \( z = \text{near}: (x,y,0) \)
    - \( z = \text{far}: (x,y,1) \)
    - subtract near result from far result to get direction vector for ray
  - use this ray for line/polygon intersection
Picking and P4

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

• do objects collide/intersect?
  • static, dynamic

• picking is simple special case of general collision detection problem
  • check if ray cast from cursor position collides with any object in scene
  • simple shooting
    • projectile arrives instantly, zero travel time

• better: projectile and target move over time
  • see if collides with object during trajectory
Collision Detection Applications

• determining if player hit wall/floor/obstacle
  • terrain following (floor), maze games (walls)
  • stop them walking through it
• determining if projectile has hit target
• determining if player has hit target
  • punch/kick (desired), car crash (not desired)
• detecting points at which behavior should change
  • car in the air returning to the ground
• cleaning up animation
  • making sure a motion-captured character’s feet do not pass through the floor
• simulating motion
  • physics, or cloth, or something else
From Simple to Complex

- boundary check
  - perimeter of world vs. viewpoint or objects
    - 2D/3D absolute coordinates for bounds
    - simple point in space for viewpoint/objects
- set of fixed barriers
  - walls in maze game
    - 2D/3D absolute coordinate system
- set of moveable objects
  - one object against set of items
    - missile vs. several tanks
  - multiple objects against each other
    - punching game: arms and legs of players
    - room of bouncing balls
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 \( i \) straddle polygon \( q \) of \( j \)
    • if straddle, then test intersection of polygon \( q \) with polygon \( p \) of object \( i \)
• very expensive! \( O(n^2) \)
Fundamental Design Principles

- *fast simple tests first*, eliminate many potential collisions
  - test bounding volumes before testing individual triangles
- exploit *locality*, eliminate many potential collisions
  - use cell structures to avoid considering distant objects
- use as much *information* as possible about geometry
  - spheres have special properties that speed collision testing
- exploit *coherence* between successive tests
  - things don’t typically change much between two frames
Example: Player-Wall Collisions

- first person games must prevent the player from walking through walls and other obstacles
- most general case: player and walls are polygonal meshes
- each frame, player moves along path not known in advance
  - assume piecewise linear: straight steps on each frame
  - assume player’s motion could be fast
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 Response

• frustrating to just stop
  • for player motions, often best thing to do is move player tangentially to obstacle
• do recursively to ensure all collisions caught
  • find time and place of collision
  • adjust velocity of player
  • repeat with new velocity, start time, start position (reduced time interval)
• handling multiple contacts at same time
  • find a direction that is tangential to all contacts
Accelerating Collision Detection

- two kinds of approaches (many others also)
  - collision proxies / bounding volumes
  - spatial data structures to localize
- used for both 2D and 3D
- used to accelerate many things, not just collision detection
  - raytracing
  - culling geometry before using standard rendering pipeline
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

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

Increasing complexity & tightness of fit:
- Sphere
- AABB
- OBB
- 6-dops
- Convex Hull

Decreasing cost of (overlap tests + proxy update)
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