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

Tamara Munzner

Collision/Acceleration

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

Reading for This Module

- FCG Sect 12.3 Spatial Data Structures

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

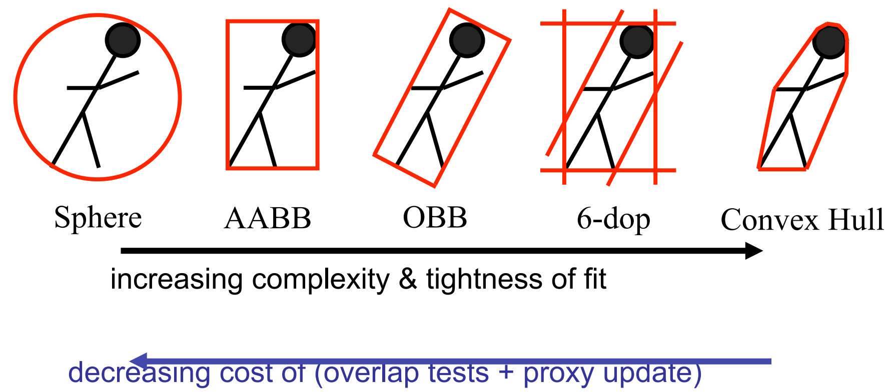
Accelerating Collision Detection

- 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
- k-dops – shapes bounded by planes at fixed orientations
  - discrete orientation polytope

Pair Reduction

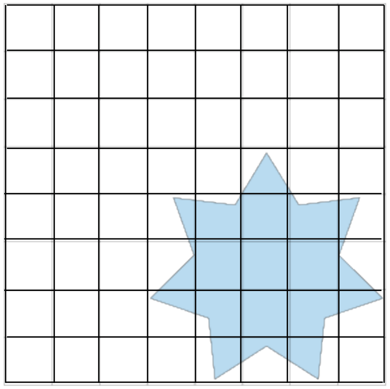
- want proxy for any moving object requiring collision detection
- before pair of objects tested in any detail, quickly test if proxies intersect
- when lots of moving objects, even this quick bounding sphere test can take too long: N^2 times if there are N objects
- reducing this N^2 problem is called *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

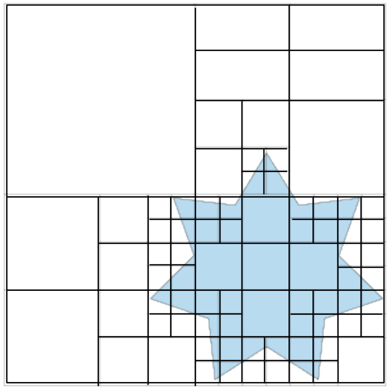
Uniform Grids

- axis-aligned
- divide space uniformly



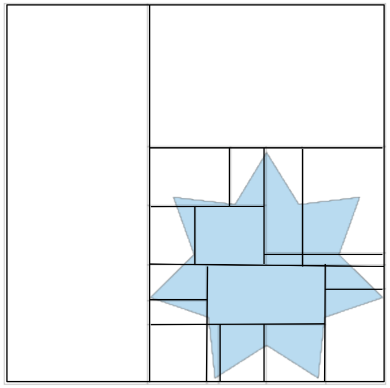
Quadtrees/Octrees

- axis-aligned
- subdivide until no points in cell



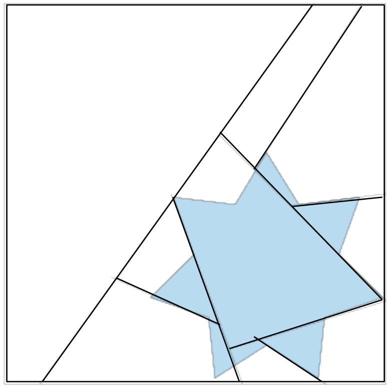
KD Trees

- axis-aligned
- subdivide in alternating dimensions

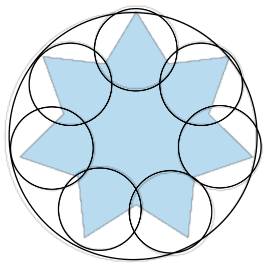


BSP Trees

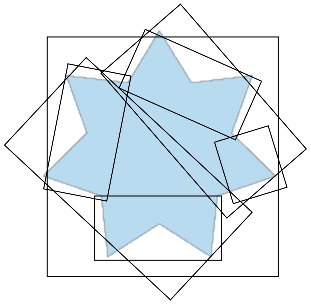
- planes at arbitrary orientation



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)
  - http://www.cs.wisc.edu/~schenney/courses/cs679-f2003/lectures/cs679-22.ppt
- slides borrow lightly from
  - Steve Rotenberg, (UCSD CSE169)
  - http://graphics.ucsd.edu/courses/cse169_w05/CSE169_17.ppt