University of British Columbia
CPSC 314 Computer Graphics
Jan-Apr 2016

Tamara Munzner

Procedural, Collision

Procedural Approaches
Procedural Textures

• generate “image” on the fly, instead of loading from disk
  • often saves space
  • allows arbitrary level of detail
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

- $D = \frac{\log(N)}{\log(r)}$
  - $N =$ measure, $r =$ subdivision scale
- Hausdorff dimension: noninteger
  - $D = \frac{\log(4)}{\log(3)} = 1.26$

http://www.vanderbilt.edu/AnS/psychology/cogsci/chaos/workshop/Fractals.html
Language-Based Generation

• L-Systems: after Lindenmayer
  • Koch snowflake: F :- FLFRRLF
  • 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/

• many shaders
  • http://www.shadertoy.com
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
  - covered in previous texturing lectures
- fractals
- L-systems
- particle systems

- not at all a complete list!
  - big subject: entire classes on this alone
Collision/Acceleration
Collision Detection

• do objects collide/intersect?
  • static, dynamic

• picking is simple special case of general collision detection problem (covered next)
  • 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

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

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
  - covered in upcoming hidden surfaces lectures
Bounding Volume Hierarchies

• covered in previous raytracing lecture
OBB Trees

- oriented bounding boxes
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