CSI 3200 Micro-Computer Graphics Collision Detection/Response

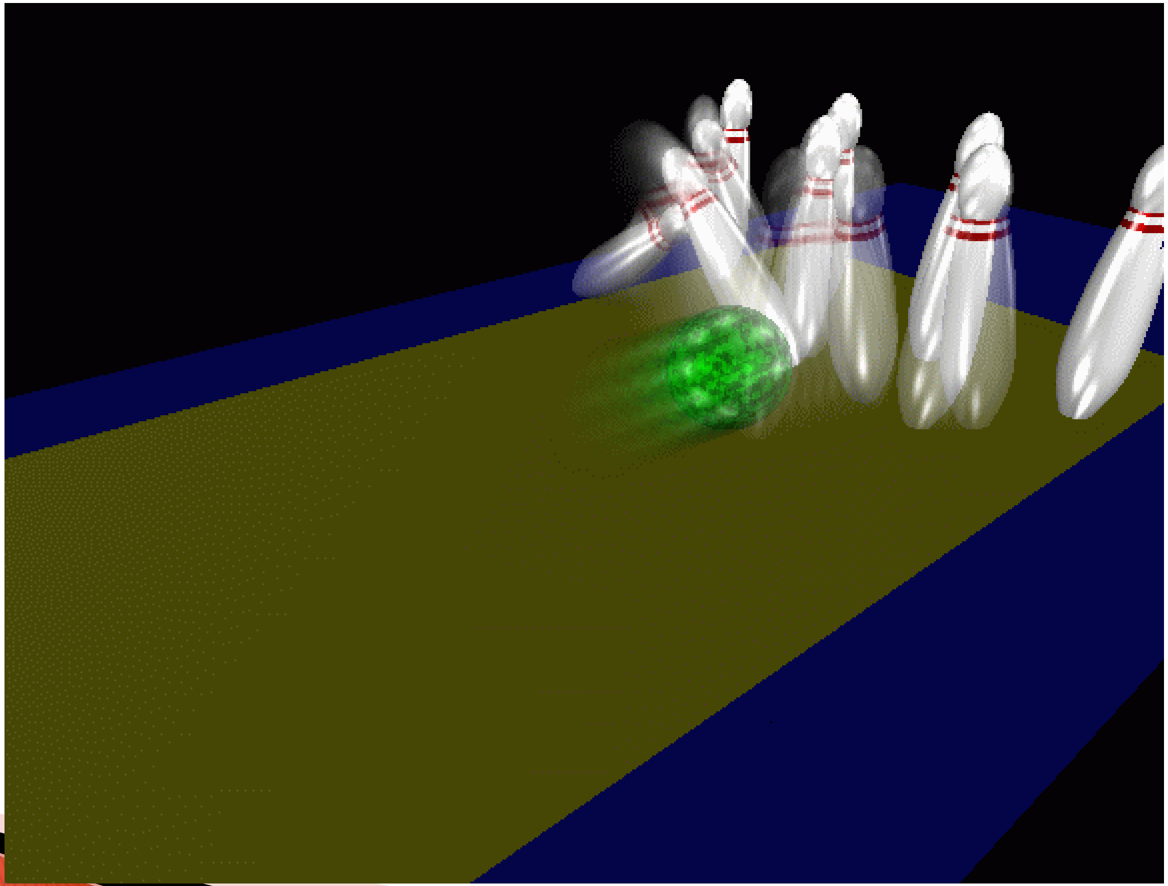
Presenter: Girendra Persaud
University of Guyana

Outline

- What is collision detection?
- Collision Response
- Classical Situation
- Implementation
  - Approach
  - Implementation 1….2
  - Portal Engines and Object-Object Collisions
- Questions
- Resources

What is collision detection?

- The observation of intersection (close-to) of two or more object in (2D/3D) space



Collision Response

- The programmed reaction for a collision
- Heavily dependant on
  - Desired natural reaction of the simulated objects, or
  - The desired effect

Consider the following areas

- Video games
- Simulations/Training
- Robotics

Classical Situations

- Boundaries created by walls/floors
- Interaction between two or more objects in space
- Consider normal physical phenomena and the effects
  - Gravity
  - Elasticity
  - Limits
- General Areas
  - Polygonal Collision Detection
  - Curved Surfaces

*Approach

- Start planning and creating its basic framework at the same time that we're developing a game's graphics pipeline
- Building a quick collision detection hack near the end of a development cycle will probably ruin the whole game because it'll be impossible to make it efficient.
- In a perfect game engine, collision detection should be precise, efficient, and very fast.
- These requirements mean that collision detection has to be tied closely to the scene geometry management pipeline.
- Brute force methods won't work — the amount of data that today's 3D games handle per frame can be mind-boggling. Gone are the times when you could check each polygon of an object against every other polygon in the scene.

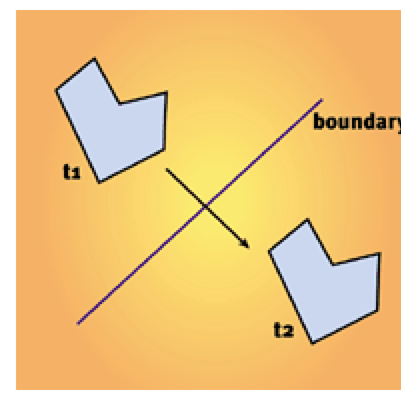
*http://www.gamasutra.com/view/feature/131598/advanced_collision_detection_.php

Implementation 1

- while(1){
```
while(1){
      process_input();
      update_objects();
      render_world();
}
```
- update_objects(){
```
update_objects(){
      for (each_object)
      save_old_position();
      calc new_object_position
      {based on velocity accel. etc.}
      if (collide_with_other_objects())
      new_object_position = old_position();
      {or if destroyed object remove it etc.}
```
- }

Problems with Implementation 1

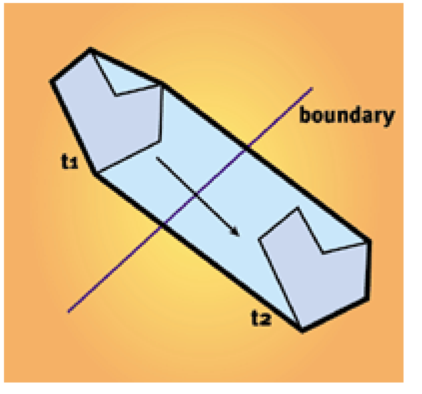
- No consideration for time in our equation
- If an object doesn't collide at time t1 or t2, it may cross the boundary at time t where $t1 < t < t2$
- This is especially true when we have large jumps between successive frames



http://www.gamasutra.com/view/feature/131598/advanced_collision_detection_.php

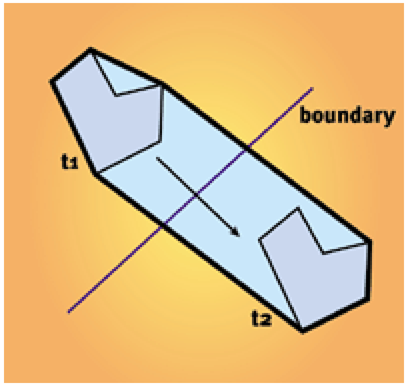
Alternatively

- We could also create a solid out of the space that the original object occupies between time t1 and t2 and then test the resulting solid against the collision boundary
- This approach is very inefficient and will definitely slow down your game



Implementation 2

- Subdivide the given time interval in half and test for intersection at the midpoint
- This approach will be faster than the previous method, but it's not guaranteed to catch all of the collisions



Consider

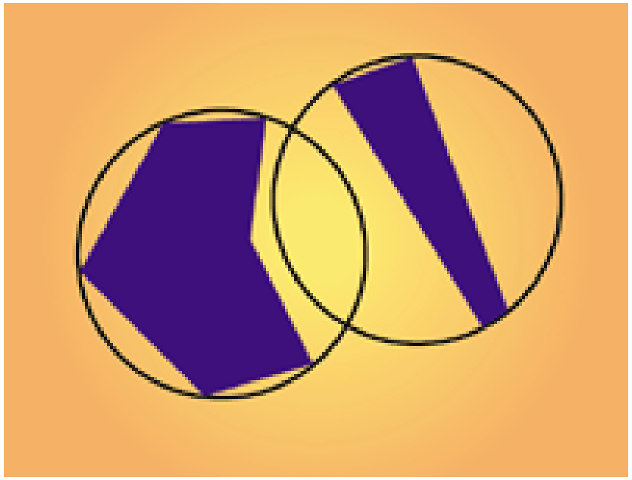
- collide_with_other_objects() routine
- If we have a lot of objects in the scene, this routine can get very costly (using the previous methods)

Portal Engines and Object-Object Collisions

- Portal-based engines divide a scene or world into smaller convex polyhedral sections
- Convex polyhedra are well-suited for the graphics pipeline because they eliminate overdraw
- Determining whether an object's polygons penetrate the world polygons can be computationally expensive
- One of the most primitive ways of doing collision detection is to approximate each object or a part of the object with a sphere, and then check whether spheres intersect each other
- This method is widely used even today because it's computationally inexpensive

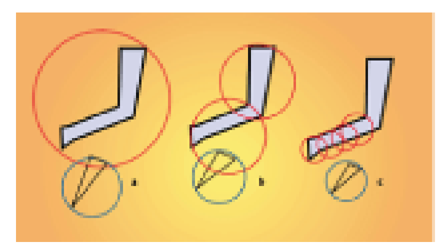
Portal Engines and Object-Object Collisions

- We check whether the distance between the centers of two spheres is less than the sum of the two radii (which indicates that a collision has occurred)



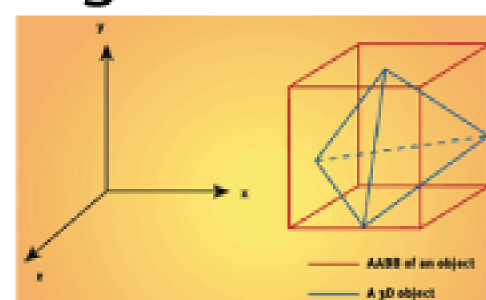
Portal Engines and Object-Object Collisions

- But what if we use this imprecise method as simply a first step.
- We represent a whole character as one big sphere, and then check whether that sphere intersects with any other object in the scene.
- If we detect a collision and would like to increase the precision, we can subdivide the big sphere into a set of smaller spheres and check each one for collision
- We continue to subdivide and check until we are satisfied with the approximation
- This basic idea of hierarchy and subdivision is what we'll try to perfect to suit our needs



Portal Engines and Object-Object Collisions

- Using spheres to approximate objects is computationally inexpensive, but because most geometry in games is square, we should try to use rectangular boxes to approximate objects
- Developers have long used **bounding boxes** and this recursive splitting to speed up various ray-tracing routine
- In practice, these methods have manifested as octrees and axis-aligned bounding boxes (AABBs)



Implementation Guidelines

- “Simple” algorithm for interaction (pools)?
  - Consider
    - Position of the objects
    - Force
    - Mass
    - Acceleration
    - Momentum
    - Velocity

Collision Detection

- A huge area of study for design of graphics simulations and games
- Further area of study
  - Building AABB trees
  - Detecting Collisions Using Hierarchy Trees
  - Collision Techniques Based on BSP Trees
  - Curved Objects and Collision Detection

Questions?

Collision Detection Further Reading

- H. Samet. *Spatial Data Structures: Quadtree, Octrees and Other Hierarchical Methods*. Addison Wesley, 1989.
- • For more information about AABBs take a look at J. Arvo and D. Kirk. "A survey of ray tracing acceleration techniques," *An Introduction to Ray Tracing*. Academic Press, 1989.
- • For a transformation speedup, check out James Arvo's paper in Andrew S. Glassner, ed. *Graphics Gems*. Academic Press, 1990.
- • S. Gottschalk, M. Lin, and D. Manocha. "OBBTree: A hierarchical Structure for rapid interference detection," Proc. Siggraph 96. ACM Press, 1996. has contributed a great deal to the discussion of OBBs in terms of accuracy and speed of execution.
- • S. Gottschalk. *Separating Axis Theorem*, TR96-024, UNC Chapel Hill, 1990.
- • N. Greene. "Detecting intersection of a rectangular solid and a convex polyhedron," Graphics Gems IV. Academic Press, 1994. introduces several techniques that speed up the overlap computation of a box and a convex polyhedron.

Resources

- http://www.gamasutra.com/view/feature/131598/advanced_collision_detection_.php
- http://en.wikipedia.org/wiki/Collision_detection
- Collision Maths
  - http://www.edenwaith.com/products/pige/tutorials/collision.php