Introduction to Collision Detection

Lecture based on *Real Time Collision Detection*, Christer Ericson, Morgan Kauffman, 2005

Game Design Experience
Professor Jim Whitehead
February 9, 2009
Announcements

• Homework #1, Revisited (Improved Design)
  ► Due today
  ► May turn it in to the box by my door by the end of the day
  ► Also make sure you do electronic submission
  ► See details on web site

• Midterm exam
  ► Aiming for return of exams on Wednesday
  ► Will post answer key after class today
Technical Design Document

• Create a UML structure diagram for your game
  ► Map out your design on paper, before writing code
  ► Allows you to think about design decisions early
  ► Due this Friday, February 13

• In one or more UML diagrams, describe all of the classes and interfaces you will create for your project
  ► For each class and interface, give
    • Class properties, methods, and variables
  ► Show inheritance and containment relationships
• Items that should be in your design
  ► input handling
  ► representation of game levels, if present
  ► player class
  ► enemy classes, including a container holding active enemies
  ► game object classes, including a container to hold them
  ► collision detection
  ► classes for drawing player, enemies, other game objects
  ► classes for handling audio
  ► menu system classes
• Make sure TDD is consistent with game design
  ► Are you missing anything?
• Strongly recommend using a UML modeling tool
  ► StarUML is free, has good set of features
  ► http://staruml.sourceforge.net/en/ (see link on Tools page)

*Demonstration of use of StarUML*
Collision Detection

- Collision detection
  - Determining if two objects intersect (true or false)
  - Example: did bullet hit opponent?
- Collision resolution (collision determination)
  - Determining when two objects came into contact
    - At what point during their motion did they intersect?
    - Example: Two balls needing to bounce off each other
  - Determining where two objects intersect
    - Which points on the objects touch during a collision?
    - Example: Pong: where ball hits paddle is important
- Complexity of answering these questions increases in order given
  - If < when < where
Key to collision detection: scaling

- Key constraint: only 16.667ms per clock tick
  - Limited time to perform collision detection
- Object shapes in 2D games can be quite complex
  - Naïve: check two objects pixel-by-pixel for collisions
- Many objects can potentially collide
  - Naïve: check every object pair for collisions
- Drawback of naïve approach
  - Takes too long
  - Doesn’t scale to large numbers of objects
- Approaches for scaling
  - Reduce # of collision pairs
  - Reduce cost of each collision check
Naïve Collision Detection: $n \times n$ checking

- Assume
  - $n$ objects in game
  - All $n$ objects can potentially intersect with each other

- Conceptually easiest approach
  - Check each object against every other object for intersections
  - Leads to $(n-1) + (n-2) + \ldots + 1 = n(n-1)/2 = O(n^2)$ checks
  - Done poorly, can be exactly $n^2$ checks
  - Example: 420 objects leads to 87,990 checks, per clock tick

*Demonstration of $n \times n$ collision checking*
Broad vs Narrow Sweep

- With many small objects in large playfield
  - Each object only has the potential to collide with nearby objects
- Broad sweep
  - Perform a quick pass over n objects to determine which pairs have potential to intersect, p
- Narrow sweep
  - Perform p x p check of object pairs that have potential to intersect
- Dramatically reduces # of checks
Broad sweep approaches

- Grid
  - Divide playfield into a grid of squares
  - Place each object in its square
  - Only need to check contents of square against itself and neighboring squares
  - See http://www.harveycartel.org/metanet/tutorials/tutorialB.html for example

- Space partitioning tree
  - Subdivide space as needed into rectangles
  - Use tree data structure to point to objects in space
  - Only need to check tree leaves
  - Quadtree, Binary Space Partition (BSP) tree

- Application-specific
  - 2D-shooter: only need to check for collision against ship
  - Do quick y check for broad sweep

Point Quadtree (Wikipedia)
Reducing Cost of Checking Two Objects for Collision

• General approach is to substitute a **bounding volume** for a more complex object
• Desirable properties of bounding volumes:
  ► Inexpensive intersection tests
  ► Tight fitting
  ► Inexpensive to compute
  ► Easy to rotate and transform
  ► Low memory use

Megaman X1 (Capcom). White boxes represent bounding volumes.
Most introductory game programming texts call AABBs simply “bounding boxes”
Circle Bounding Box

• Simple storage, easy intersection test
• Rotationally invariant

struct Point {
    int x;
    int y;
}

struct circle {
    Point c; // center
    int r;   // radius
}

bool circle_intersect(circle a, circle b) {
    Point d; // d = b.c - a.c
    d.x = a.c.x - b.c.x;
    d.y = a.c.y - b.c.y;

    int dist2 = d.x*d.x + d.y*d.y; // d dot d
    int radiusSum = a.r + b.r;

    if (dist2 <= radiusSum * radiusSum) {
        return true;
    } else {
        return false;
    }
}
Axis-Aligned Bounding Boxes (AABBs)

- **Three common representations**
  - **Min-max**
  
  ```
  struct AABB {
    Point min;
    Point max;
  }
  ```

  - **Min-widths**
  
  ```
  struct AABB {
    Point min;
    int dx; // x width
    int dy; // y width
  }
  ```

  - **Center-radius**
  
  ```
  struct AABB {
    Point c;
    int rx; // x radius
    int ry; // y radius
  }
  ```

Can easily be extended to 3D
• Two AABBs intersect only if they overlap on both axes

```cpp
bool IntersectAABB(AABB a, AABB b) {
    if (a.max.x < b.min.x || a.min.x < b.max.x) return false;
    if (a.max.y < b.min.y || a.min.y < b.max.y) return false;
    return true;
}
```
• Two AABBs intersect only if they overlap on both axes

```cpp
bool IntersectAABB(AABB a, AABB b) {
    int t;

    t = a.min.x - b.min.x;
    if (t > b.dx || -t > a.dx) return false;

    t = a.min.y - b.min.y;
    if (t > b.dy || -t > a.dy) return false;

    return true;
}
```

// Note: requires more operations than
// min-max case (2 more subtractions,
// 2 more negations)
AABB Intersection (center-radius)

- Two AABBs intersect only if they overlap on both axes

```c
bool IntersectAABB(AABB a, AABB b) {
    if (Abs(a.c.x - b.c.x) > (a.r.dx + b.r.dx))
        return false;
    if (Abs(a.c.y - b.c.y) > (a.r.dy + b.r.dy))
        return false;
    return true;
}

// Note: Abs() typically single instruction on modern processors
```
Homework

• Readings from Real Time Collision Detection
  ► See forum/syllabus for details, will scan today