CMPS 20: Game Design Experience

Collision Detection and Resolution
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

Chu Chu Rocket, Dreamcast
Naïve Collision Detection: n x 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
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
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.
Common Bounding Volumes

- Circle/Sphere
- Axis-Aligned Bounding Box (AABB)
- Oriented Bounding Box (OBB)
- Convex Hull

Better bounds, better culling
Faster test, less memory

• 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;
    }
}

Compare Euclidean distance between circle centers against sum of circle radii.
Axis-Aligned Bounding Boxes (AABBS)

- Three common representations
  - Min-max
    - Min-widths
      - Center-radius

Can easily be extended to 3D
Axis Aligned Bounding Box Intersection (min-max)

- 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;
}
```
# Axis Aligned Bounding Box Intersection (min-width)

- 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

\[
\begin{align*}
\text{b.c.x} - \text{a.c.x} &> \text{a.rx+b.ry} \\
\text{a.c.x} - \text{b.c.x} &> \text{a.rx+b.rx} \\
\text{a.c.x} &= \text{b.c.x} \\
\text{b.c.y} - \text{a.c.y} &> \text{a.ry+b.ry} \\
\text{a.c.y} - \text{b.c.y} &> \text{a.ry+b.ry} \\
\text{a.c.y} &= \text{b.c.y}
\end{align*}
\]

```cpp
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
There’s more ...

- Per-pixel collision
- Per-pixel with transformed objects