CS174 - Construction of a Game

David Warrick
Vy-Luan Huynh
Goal

- To create a physics engine that supports and simulates object collisions
- Given enough time, build a basic beat-'em-up game on top of the engine
Physics Engine

- Has to be able to simulate object interaction
- What happens when two objects collide?
- How to handle gravity, air resistance, and/or friction
- Must be able to do all of this in real time
Building a Game

- Need a player character
- Need a way to control the player such that they properly interact with the environment
  - E.g. can’t just shift the player’s position if the physics engine relies on velocity for collision math
- If enemy characters exist, they need to be distinguishable from the player
- Need a way to distinguish between enemy-on-player collisions and player-on-enemy
  - E.g. player punching an enemy shouldn’t deal damage to the player, and vice versa
Collision Detection - Used the GJK (Gilbert-Johnson-Keerthi) algorithm

Basic idea: use the Minowski Sum of two objects to determine if they have collided

Minowski Sum of two shapes is all of the points in shape 1 added to all of the points in shape 2

\[ A + B = \{a + b | a \in A, b \in B\} \]

GJK works with the difference instead of the sum:

\[ A - B = \{a - b | a \in A, b \in B\} \]

If two shapes are overlapping, this difference will contain the origin.
GJK cont. - Minowski Sum

Intersecting convex shapes

The Minowski “Difference”

Computing entire difference is expensive - only need to determine if it contains the origin

Use Simplexes to try and build a polygon inside the Minowski Difference that contains the origin

This is enclosed in a support function:

- Vector3d support(Shape s1, Shape s2, Vector3d d)
  - Vector3d p1 = s1.getFarthestPointInDirection(d)
  - Vector3d p2 = s2.getFarthestPointInDirection(-d)
  - Vector3d p3 = p1 - p2
  - return p3

Point p3 is added to the simplex
GJK cont. - Simplex Building

- We choose the farthest point in a direction to create a simplex with maximum area, which helps the algorithm exit quickly.

- Also means that all points produced by the support function are on the edge of the Minowski sum.

- This means that if the support function does not return a point on the opposite side of the origin from the previous point, it is impossible for the sum to contain the origin.
  - This allows the algorithm to immediately exit and return false.
GJK cont. - Choosing the Next Simplex Point

- If simplex has only two points (A and B):
  - Return direction perpendicular to the line in the direction of the origin
  - Achieved with a “triple product”: $O \times (AB \cdot AB) - AB(AB \cdot O)$ where $O$ is the direction from point A to the origin and AB is the line between points A and B

- If simplex has three points (A, B, and C):
  - Return direction normal to the triangle in the direction of the origin
  - Take cross product of AB and AC
  - Take the dot product of this with the direction to the origin from A, to see if the normal is in the direction of the origin (if it isn’t, reverse it)
If simplex has four points (A, B, C, and D):

- Origin must lie within the simplex or in a region of space normal to one of the simplex’s faces
  - Know this due to how we choose points
- Check each exterior region; if origin is not in any of them then the simplex contains the origin
- Can do this by finding the outward-facing normal to every simplex face and then taking the dot product of the normal and the vector AO, which is the direction to the origin from point A
  - If the dot product is positive, the origin is in that direction. Return the normal as the new direction and remove the point that is not a member of the triangle
  - If the dot product is negative, check the other faces
  - If every dot product is negative, the simplex contains the origin
Using directions produced by the simplex, can continually choose new points to create new simplexes.

Algorithm exits if one of two conditions are met:
- The next point produced by support() is not past the origin
  - This means the Minowski sum does not contain the origin. Return false
- The constructed simplex contains the origin. Return true

(Implementation of the GJK calls for a third condition: to prevent infinite loops, the algorithm exits after some number of iterations, at which point a collision is assumed to have happened)
Collision!

- If the GJK algorithm returns true, then a collision has occurred between the two objects.
- Need to calculate new velocity vectors.
- Fairly simple kinematics problem:
  - Pick a reference frame where one object is at rest, then solve:
    \[ m_1 v_1 = m_1 v_1' + m_2 v_2' \quad \text{and} \quad \frac{1}{2} m_1 v_1^2 = \frac{1}{2} m_1 v_1'^2 + \frac{1}{2} m_2 v_2'^2 \]
  - Result: \( v_1' = \frac{m_1 - m_2}{m_1 + m_2} v_1 \) and \( v_2' = \frac{2m_1}{m_1 + m_2} v_1 \)
  - Accounting for the choice of reference frame, the final velocities are:
    \[ v_1' = \frac{m_1 - m_2}{m_1 + m_2} (v_1 - v_2) + v_2 \quad \text{and} \quad v_2' = \frac{2m_1}{m_1 + m_2} (v_1 - v_2) + v_2 \]
Collision! - With World Boundary

- World Boundaries (i.e. walls, ground, ceiling) handled differently
- Boundaries don’t move and don’t have mass
- On collision, the object’s corresponding component of velocity is reversed
  - Collision with the ground or ceiling will reverse the object’s y-velocity
  - Collision with a “WALL_RIGHT” or “WALL_LEFT” will reverse the object’s x-velocity
  - Collision with a “WALL_FRONT” or “WALL_BACK” will reverse the object’s z-velocity
- Boundaries also apply a dampening multiplier to simulate the loss of kinetic energy
All physics calculations occur in a physics() function that is called once per frame.

Physics() first finds all of the collisions between objects and saves the new net velocities produced from each collision.

Then applies a “slowdown” factor to simulate air resistance/friction.

Then checks each object against the world boundaries for collisions and reverses object velocities as necessary.

Applies gravity to the object’s y-velocity.

Finally, calculates the object’s new position by adding its velocity vector to its previous position.
For the Future

- More sophisticated models and controls
- Enemy AI
- Animations