Ross Adelman
15467 A

Collision Detection Using SDFs

For project 3, I decided to implement a rigid body simulator. For the most part, I based my implementation on the paper by David Baraff entitled "Physically Based Modeling: Rigid Body Simulation." However, I implemented collision detection using a completely different method.

Each of the bodies in the simulation is defined by a signed distance function (SDF). Basically, an SDF is a way to implicitly define a region in space -- the SDF is negative for points inside the region, positive for points outside the region, and zero for points on the boundary. A lot of cool things can be done with SDFs. For example, let's say we have two SDFs, d_1 and d_2, which define two regions, A and B. We can create an SDF, $d_3 = \min(d_1, d_2)$, which defines a region equal to the union of A and B. We can also create an SDF, $d_4 = \max(d_1, d_2)$. The region defined by d_4 is the intersection of A and B.

There are three versions of the algorithm. They are all extremely similar -- they only differ by a couple steps near the end.

Slow Method

(1) Construct the smallest box possible that contains both bodies.
(2) Calculate $\max(d_1, d_2)$ at the center of the box, where d_1 and d_2 are the SDFs of the bodies.
(3) Is the value greater than the size of the box?
    a. Yes? There's no way the bodies are colliding in the box. Stop.
    b. No? Keep going.
(4) Divide the box into eight smaller, equally sized ones.
(5) Repeat (2) - (4) for each of those boxes.

This can't go on forever, so some maximum depth is chosen beforehand. If, at the maximum depth, (4) is reached, then the box's center is saved to an array. After the algorithm has completed, the array, if not empty, will contain a collection of points that lie in the region occupied by both bodies. Taking the average of these points will yield the point of the collision. The normal of the collision can be calculated using the gradient of the SDFs. These values can be used to resolve the collision like usual. In the best case, the run time of this algorithm is $O(k)$, where k is the maximum depth. This assumes only one of the eight children boxes reaches (4) -- the other seven are thrown out. This rarely happens. In the worst case, the run time is $O(8^k)$.

Despite being slow, this version has some benefits. For example, suppose two cubes are colliding in such a way that the collision is occurring uniformly over some surface. Then, this

version will yield a point of collision somewhere in the middle of that surface. This is typically the best place to resolve the collision.

Fast Method

(1) Construct the smallest box possible that contains both bodies.
(2) Calculate max(d_1, d_2) at the center of the box, where d_1 and d_2 are the SDFs of the bodies.
(3) Is the value greater than the size of the box?
    a. Yes? There's no way the bodies are colliding in the box. Stop.
    b. No? Keep going.
(4) Divide the box into eight smaller, equally sized ones.
(5) Calculate max(d_1, d_2) at the center of each of these boxes.
(6) Pick the one that has the smallest value.
(7) Repeat (2) - (6) for this box.

The major change here lies in (6). Instead of repeating all the steps blindly for all eight boxes, this version only does them for one box, and throws all seven of the other boxes out. As a result, this version no longer has an exponential run time -- the worse case run time is now O(k). However, in exchange for being faster, this algorithm lacks accuracy in some scenarios. There is a chance that more than one of the eight boxes will have the same (or very close) value. However, the algorithm chooses only one box. In the example above with the two cubes, this version might yield a point of collision nowhere near the center of the collision. Instead, it could be near the edge. This could lead to a weird looking collision response -- for example, the collision might impart torque on the cubes, which might be incorrect.

Hybrid Method

(1) Construct the smallest box possible that contains both bodies.
(2) Calculate max(d_1, d_2) at the center of the box, where d_1 and d_2 are the SDFs of the bodies.
(3) Is the value greater than the size of the box?
    a. Yes? There's no way the bodies are colliding in the box. Stop.
    b. No? Keep going.
(4) Divide the box into eight smaller, equally sized ones.
(5) Calculate max(d_1, d_2) at the center of each of these boxes.
(6) Pick the one that has the smallest value.
(7) Pick any other of the seven boxes that has a value of max(d_1, d_2) equal or close to that of the box chosen in (6).
(8) Repeat (2) - (6) for this box.

The hybrid method corrects the problem described above by sacrificing some speed. Instead of choosing only one of the eight boxes, after selecting the box with the smallest

value, the hybrid method checks whether any of the other seven boxes have the same (or a very similar) value. If so, the algorithm recurses on these as well.

In the end, I decided to use the fast method. This was mostly because it was guaranteed to run in O(k). The hybrid method, while faster than the slow method, occasionally ran in exponential time.