Process Paper

Overview:

This paper will give a breakdown on the elements of the 3D physics engine I built for CS550.

It utilizes GJK for collision detection, EPA for contact extrapolation and Sequential Impulse solver implemented using a PGS scheme.

The engine uses an Entity-Component system that communicates using an Observer pattern.

Limitations of this engine include a lack of stacking and persistent manifolds.

Integration:

By default, the engine uses a semi-implicit Euler integration method. RK-4 was implemented, but never required to be used. Same goes for a Verlet integrator that was also implemented.

There are some caveats to integration, such as where colliders that are marked as ‘Static’ are not integrated. This is an issue of poor component design where I require objects that don’t really need physics (namely static objects) to still own one as they contain some data that is necessary for collision detection.

An improvement to this would be to decouple the Collider and Physics components dependencies so that a Collider could function without a Physics component also belonging to the owner.

Collision Handling:

The order of events in the physics engine goes as follows:

1. All objects are advanced one timestep
2. Collision detection is run between all pairs of objects (no broadphase)
3. If collision is detected, EPA is run to extrapolate the contact
4. A collision/contact constraint is created to resolve the collision
5. If relative velocity between two bodies in a constraint is separating or the produced impulse is lesser than an order of magnitude, the constraint is deleted
6. Repeat

Collision Detection:

GJK:

The Gilbert-Johnson-Keerthi(GJK) [1] algorithm is used to provide a Boolean check of whether the two convex shapes that are being tested are intersecting. The strength of GJK is that it can be manipulated completely based on the type of support function (or) support map being used.

Originally, I was using the naïve method of looping through all vertices and getting the dot product of each vertex against the search direction and finding the vertex furthest along that direction. However, for a shape with the spatial symmetry about all three axes (i.e. a cube or a sphere) it is possible to use another method involving an ‘Extents’ approach[2], which just multiplies the sign of the search direction against the half size of the cube (or the radius of the sphere). In this engine I have only implemented cubes, so this method proved sufficient.

Some tradeoffs I have noticed with this method versus the naïve method are:

* Greater number of false positives (false detections of simplex being within origin)
* Causes degenerate simplexes in some configurations

But the upside is a decoupling from the actual vertices of the cube shape, which I greatly prefer as a solution and eliminates the need to loop through all vertices, though that could be outweighed by the extra EPA[3] calculations due to a higher number of false positives. Will need to profile it to make any serious determinations on this front.

There is an iteration limit of 75 to prevent the GJK from cycling infinitely.

As an interesting tidbit, I stumbled across a forum post[4] where Dirk Gregorious states that him and Erin Catto both agree that Casey’s implementation of GJK aggressively overoptimizes and skips some Voronoi regions that need checking. It worked for Casey in his use-case because he would clip away any false positives later on, but his method is prone to error and should not be used as a reference implementation, according to Dirk at the very least.

EPA:

If/when GJK finds a simplex that encloses the origin, it passes the simplex onto EPA which attempts to locate the closest point possible to the origin in the Minkowski space, which should give the contact point, in theory.

In practice, due to the limitations of 32-bit floating point precision and math, EPA is very prone to numerical error, and as Dirk Gregorious as stated on various forums, is a method he does not personally like. I began to see why when I encountered some situations whose blame falls squarely on the imprecision of EPA and its inability to return a consistent contact point or one that can be guaranteed to be close to the center of mass.

It’s my belief that persistent manifolds in fact are a way to make up for the imprecision introduced by EPA into the resolution process, though also the issue of error in impulse solvers exacerbates this.

When EPA returns a point that is within a tolerable limit of the origin, the contact point is extrapolated using barycentric projection. There is an iteration limit of 50 to prevent the EPA from cycling infinitely.

Collision Resolution:

The contact that is generated by EPA is registered using a contact/collision constraint.

This constraint is solved using a method given by Erin Catto in his paper “Iterative Dynamics using Temporal Coherence”[5]. It involves a sequential impulse method that comes down to a linear complementarity problem that must be solved using a numerical solver of some type.

The type of solver used in the paper and here is a projected Gauss-Siedel solver, which is shown to converge faster for this use-case than the Jacobi-Hamilton solver.

The solver performs velocity correction to ensure the velocity-level constraint is satisfied and objects will no longer move into each other, and the penetration is resolved using a Baumgarte stabilization method, which massages the constraint forces to get them to do “virtual work”.

There is also a slop term that is provided on the Baumgarte stabilization to allow objects to penetrate a bit before we apply the Baumgarte term.

Initially there was slop provided for the restitution term too but that caused stability problems and was removed for the submission.

Debugging Tools:

Options in the ‘Debug Settings’ widget allow for these debug tools to be enabled:

* Render simplex
* Render colliders and normal
* Render wireframe mode
* Render Minkowski difference (always between the first cube and the ground object)
* Change thickness of wireframe and line loops
* Enable/Disable simulation (stops integration and impulse solver from running)
* Enable/Disable contact debug mode (stops simulation every time a contact is run)

There are also some debug hotkeys like holding down the SPACE key to stop simulation when a collision is detected and LEFT\_ALT to continue when a contact is detected and the simulation is stopped.

And the debug visual tools implemented were:

* Line loops
* Billboarded quads
* Wireframes
* Normals

Issues:

Lack of persistent manifolds and numerical issues with GJK and EPA mean that there are still some penetrations that can happen in some situations.

Additionally the issue of jitter can cause penetrations to happen. I plan on continuing with the implementation of this engine at least until I can obtain stable stacking, just to teach myself as well as to have that on my portfolio.

Conclusion:

I am pretty satisfied with the final state of my engine. It’s far from perfect, but considering that last year I spent upwards of three months trying to implement OBBs using SAT and I failed utterly, the fact that I managed to implement a working engine with detection and response is honestly enough for me.

The one thing I would have liked to get done was to implement stacking and persistent manifolds, but time got away from me.

Special Thanks:

Erik Mohrmann, Steven Barnett.

References:

1. “The Gilbert-Johnson-Keerthi Distance Algorithm”, Gilbert, Johnson and Keerthi - <https://www.medien.ifi.lmu.de/lehre/ss10/ps/Ausarbeitung_Beispiel.pdf>
2. The Lattice3D engine, from which the ‘Extents’ method is taken, Jacob Tyndall- [https://bitbucket.org/Hacktank/lattice3d/src/adfb28ffe5b51dbd1a173cbd43c6e387f1b4c12d/Lattice3D/src/physics/collider/BoxColliderComponent.cpp?](https://bitbucket.org/Hacktank/lattice3d/src/adfb28ffe5b51dbd1a173cbd43c6e387f1b4c12d/Lattice3D/src/physics/collider/BoxColliderComponent.cpp?at=master&fileviewer=file-view-default)
3. “Proximity Queries and Penetration Depth Computation on 3D Game Objects”, Gino van den Bergen - <http://movement.stanford.edu/courses/cs468-01-fall/Papers/van-den-bergen.pdf>
4. “Dirk doesn’t like EPA/Casey’s method is prone to error” - <https://www.gamedev.net/topic/613054-gjk-caseys-way-closest-pt-illustrations/>
5. “Iterative Dynamic using Temporal Coherence”, Erin Catto – <http://www.bulletphysics.com/ftp/pub/test/physics/papers/IterativeDynamics.pdf>