Physics Simulation of Rigid Bodies

Evan Collier

Goals for this project

Support for concave and convex bodies

Robust and performant real-time simulation

Stackable objects

Apply friction and drag forces

Libraries being used

glm - math library

Imgui - gui library

OpenGL 3.3 - graphics API

stb - image loading

Assimp - model loading

glfw 3.2.1 - window management

glad - OpenGL function bindings

Engine Sequence

1.) Apply forces and torques
2.) Update linear and angular momentums with fixed time step
3.) Update linear and angular positions within fixed time step
4.) Update collider object if necessary
5.) Update broad-phase and check for collision pairs
6.) Pass collision pairs to narrow phase to generate contact manifold (in progress)
7.) Resolve contacts (impulse application or resting contact)
8.) Repeat

Rigid Body

Contains the following state information for the object:

```
glm::mat3 Ibody;

glm::vec3 x; // position
glm::quat q; // orientation
glm::vec3 P; // Linear Momentum
glm::vec3 L; // Angular Momentum
glm::vec3 force;
glm::vec3 torque;

float mass;
```

Rigid Body Update

Integrator is currently forward euler

```
 P = P + force * dt; // implicitly has acceleration data
 v = P / mass;
 x = x + v * dt;
```

Using quaternions for keeping track of orientation/angles

```
L = L + torque * dt;
w = Iinv * L;
glm::quat qdot = 0.5f * glm::cross(glm::quat(0, w), q);
q = q + qdot * dt;
```

Using a fixed time step for the update

```
dt = (1.0f / 144.0f);
```

Collider

Contains the following shape information for the object:

```
AABB mAABB;
```

Planning on adding an actual collision shape that is accurate to the mesh.

Collider Update

Currently, the collider only contains the bounding box of the object so we only update that

```
mAABB.Transform(mObject->mRigidBody->GetModelMatrix());
```

The collider's bounding box is computed from the model's points as follows

```
mAABB.mMin = points[0];
mAABB.mMax = points[0];
for (auto& pt : points) {
  for (unsigned i = 0; i < 3; ++i) {
    mAABB.mMin[i] = std::min(mAABB.mMin[i], pt[i]);
    mAABB.mMax[i] = std::max(mAABB.mMax[i], pt[i]);
  }
}
```

Dynamic Aabb Tree Implementation

Contains the following set of data to function:

```
const float mPad;
unsigned mRoot;
unsigned mKeyCounter;
std::vector<Node> mNodes;
std::stack<int> mFreeIndex;
```

Insert function uses the minimum change in surface area to determine which branch to take on insert

Erase function properly removes and reshapes the tree

Self Query returns a list of query results which contain pairs of colliding objects

Broad-Phase

Using a dynamic aabb tree to cull objects that aren’t colliding

Dynamic aabb tree is currently unbalanced. Will be doing a perf test to see if balancing is beneficial

Currently storing nodes in a vector as opposed to dynamically allocating each node, improved performance.

Collision list has no duplicates and early outs when it can by checking node aabbs for overlap before descending.

https://twitter.com/etc597/status/971618031959879680

Narrow Phase

Currently in the process of implementing GJK with EPA.

Decided on GJK as it is known for its robustness and its performance over other narrow-phase options

Performance is important as I plan on supporting complex convex shapes in my simulation

Will produce a contact manifold

Contact Manifold

A contact will contain the following information:

```
glm::vec3 contactPoint;
glm::vec3 faceNormal;
float penetrationDistance;
```

This contact manifold will be passed to the contact resolver so that impulses can be applied or resting contacts can be resolved.

Contact Resolver

From the contact manifold generated by GJK, a contact response will be generated

The contact response will depend on the relative velocities

If the relative velocity is small and the penetration distance is small, resting contact will be modeled

Otherwise an impulse will be applied to the objects as the following formula shows:

```
j = (-1 + r) * vRel / (maInv + mbInv + crossp(dotp(n, Iainv * crossp(rax, n)), ra) +
crossp(dotp(n, Ibinv * crossp(rb, n)), rb)
```

Current problems and issues

Narrow phase has not yet been implemented

Actual collision shape and convex decomposition has not yet been implemented. Still just using the graphics model.

Friction application has not yet been implemented

GUI could use more debugging features

Forward Euler won’t be good enough when applying friction, presumably

Conclusion

Goal is to make a robust 3D rigid body simulation that supports friction, resting contact, and concave objects.

Making good progress towards this goal and should be there by the end of the semester.

Going to be taking spring break to make up for the lost time working on other classes/homework.

References

[1] Baraff, David. (1997). An Introduction to Physically Based Modeling: Rigid Body Simulation I—Unconstrained Rigid Body Dynamics. Carnegie Mellon University.
[2] Coutsias, Evangelos A. and Romero, Louis. (1999). The Quaternions with an application to Rigid Body Dynamics. University of New Mexico.
[3] Mandre, Indrek. (2008). Rigid body drnamics using Euler’s equations, Runge-Kutta and quaternions.
[4] (2018, February 8). Quaternions. https://en.wikipedia.org
[5] (2017, September 20). Exterior Algebra. Retrieved from https://en.wikipedia.org
[6] Mamou, Khaled. (2011, October 2). HACD: Hierarchical Approximate Convex Decomposition. Retrieved from http://kmamou.blogspot.com

Questions?