Objective of the project

This project is going to implement Post-Stabilization Algorithm, which is an algorithm that correct the interpenetration created during the simulation process. The algorithm should address the three issues of the correction algorithm widely used these day, which are adding extra energy into the current system, allowing bodies to be in an inadmissible configuration, and requiring tuning on constant before using. The algorithm will make physical simulation much more reliable.

Fields that get benefits

Rigid body simulation is widely used in many different fields. Some major fields among those are Robotics, Mechanical Engineering, and Game Development. In this section, we will talk about how these fields uses Rigid body simulation and how they will be affected by this algorithm.

In Robotics, rigid body simulation is used to test programs before running them on a real robot, so that the robot will not be broken by the program due to unappropriated command in the program. With the algorithm, the simulation is more reliable, and the robot is more likely to perform how it performs in the simulation. This can help speed up the efficiency of robot programming, since running program in simulation is much easier and faster in simulations that on real robots.

In Mechanical Engineering, rigid body simulation is often used during virtual prototyping. In virtual prototyping, prototypes of designs are constructed and validated in a rigid body simulator. With the algorithm, the simulation is more reliable, which means the physical designs are less likely to malfunction when they are produce. Both the economic cost and the time cost of product development can be lowered.

In Game Development, rigid body simulation is one of the main component by a physical engine which is widely used nowadays. With the algorithm, there will less likely be a glitch in the engine due to extra energy being added to the system. A downside in this case though, is since a glitch in the game system usually has minor consequence, game engines are design to focus more on speed than the accuracy of the simulation.

The flow of the project

The algorithm will be implement and tested under Moby, a time-stepping rigid body dynamic simulator software written in C++. The basic idea of this algorithm is to add a constraint stabilization step after every simulation step. Figure 1 and Figure 2 are graphical illustrations of the simulator before and after constraint algorithm is added repectively.

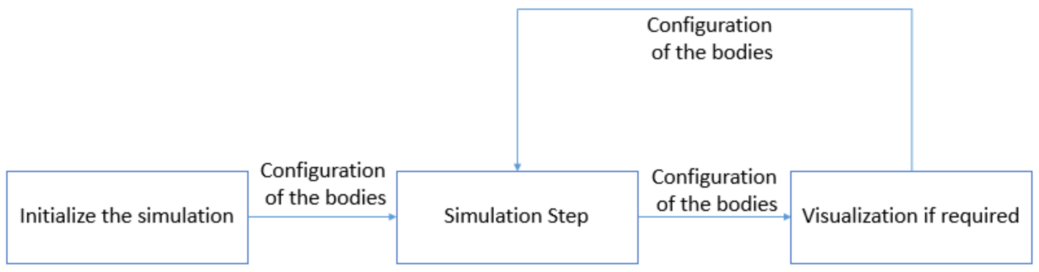
Configuration of the bodies
Initialize the simulation
Configuration of the bodies
Simulation Step
Configuration of the bodies
Visualization if required

Figure 1 Program flow before Constraint stabilization is added

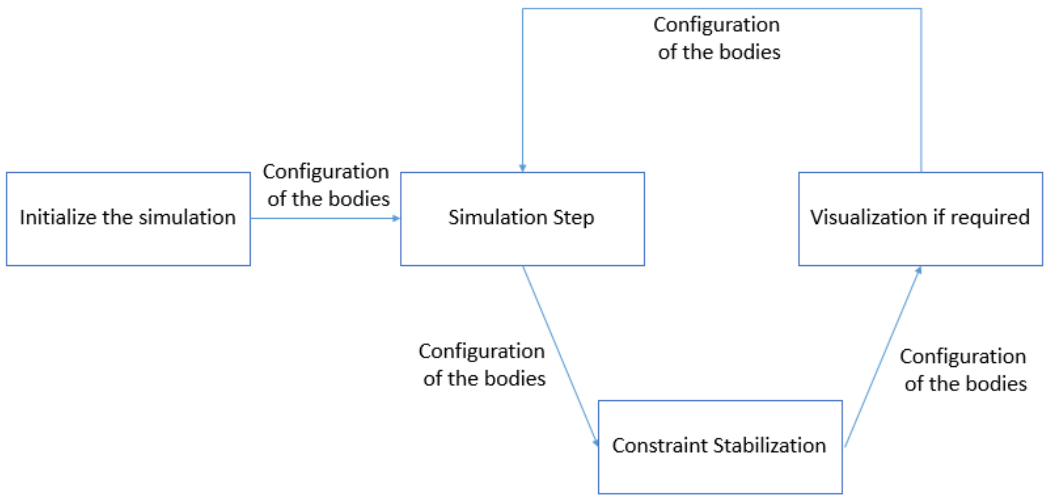

Figure 2 Program flow after Constraint stabilization is added

From the graph we can see the constraint stabilization is going to take the configuration of the bodies as input correct them and return the corrected configuration as output.

Figure 3 is a figure of the program flow of constraint stabilization.

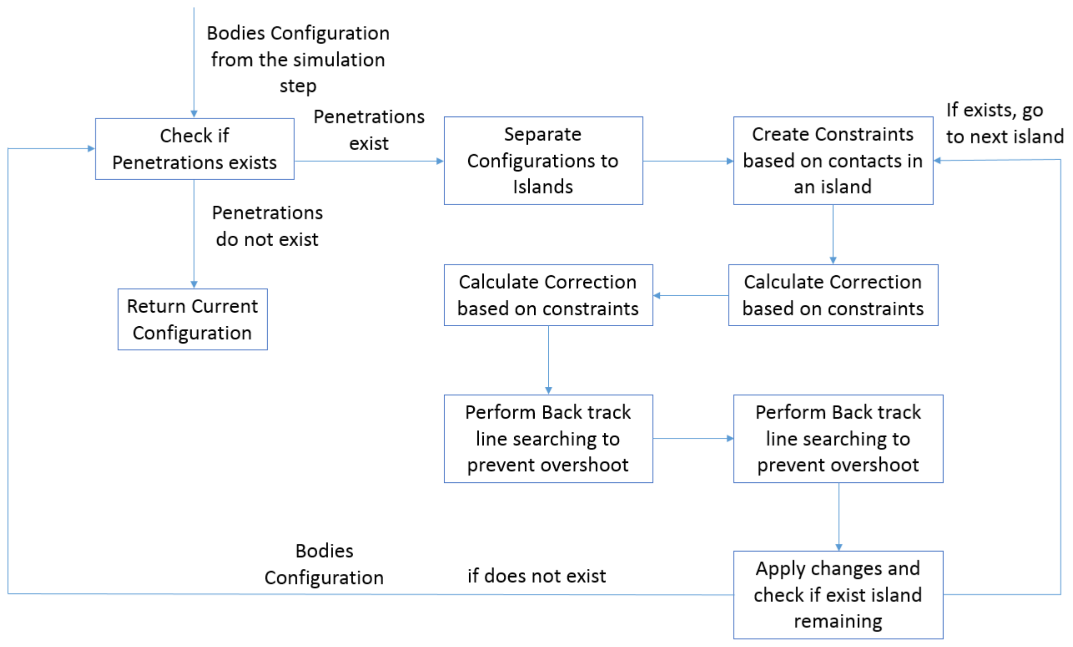
Bodies Configuration from the simulation step
Check if Penetrations exists
Penetrations exist
Separate Configurations to Islands
Create Constraints based on contacts in an island
If exists, go to next island
Penetrations do not exist
Return Current Configuration
Calculate Correction based on constraints
Calculate Correction based on constraints
Perform Back track line searching to prevent overshoot
Perform Back track line searching to prevent overshoot
Bodies Configuration
if does not exist
Apply changes and check if exist island remaining

Figure 3 Program flow of constraint stabilization

Following is the program flow of constraint stabilization in a ordered list form:

1. Gets configuration from the end of the simulation step
2. Check if penetrations exist
    a. If exist
        i. Separate the configurations into islands where each island consists of rigid bodies that are touching or interpenetrating
        ii. Create the constraint based on the contacts in the island
        iii. Calculate the correction needed based on the constraints
        iv. Perform Backtrack line searching to prevent overshoot
        v. Apply changes and check if there are islands remaining, if yes, return to step 2.a.i with the next island, if not return to step 2
    b. If do not exist

i. Return the current configuration

Functional and non-functional requirements

The functional requirement of this project is, the algorithm should correct the bodies configuration into an admissible state.

The non-functional requirements will be, the algorithm should run fast, and the algorithm should be robust so that edge cases can be handled.