Project Hollow Point: Bullet Physics Integration into Unity 3D

DESIGN DOCUMENT

May 1741

Advisor: Mani Mina
Travis Evers
Tevers@iastate.edu

Revised: 1 Dec 2016

Contents

1 Introduction

1.1 PROJECT STATEMENT

This project is about integrating the Bullet Physics Library into Unity 3D.

1.2 PURPOSE

Unity 3D use a light version of the NVidia Phys X system which does not utilize the GPU for physics calculations whereas the Bullet Physics Library uses the GPU for all physics based calculations. By performing physics calculations on the GPU instead of the CPU accuracy and speed is vastly improved. Other benefits for including the Bullet Physics Library include having full access to soft body physics calculations, such as, cloth movement, deformable surfaces, and fluids.

1.3 GOALS

The goal of this project is to integrate the three major components of the Bullet Physics Library into Unity 3D without sacrificing performance or stability. The three major portions of the Bullet Physics Library are as follows: rigid body, soft body, and inverse kinematic.

2 Deliverables

Full integration of all components in the Bullet Physics Library, a set update cycle that works with both Unity 3D and the physics space created by Bullet Physics, updated user interface for creating/editing/manipulating Bullet Physics objects in the Unity Editor, integration with both the legacy animation system and Mecanim animation system. If time permits I would like to modify the assertion testing module of Unity to also support Bullet Physics as well.

3 Design

Include any/all possible methods of approach to solving the problem. Discuss what you have done so far. What have you tried/implemented/tested etc. We want to know what you have done.

3.1 SYSTEM SPECIFICATIONS

System specifications for this project are simply any system capable of running the Unity 3D version 5 editor and Unity 3D version applications.

3.1.1 Non-functional

This project has a few non-functional requirements: maintain Unity 3D's portability, maintainability, making the plugin editor so it feels familiar to the base Unity 3D editor, and making it so the plugin is expandable/modifiable.

3.1.2 Functional

The functional requirements for this project include the following: integrate the rigid body physics component of Bullet Physics, integrate the soft body physics component of Bullet Physics, integrate the inverse kinematic physics component of Bullet Physics, all physics calculations will be done on the GPU instead of the CPU, no timing complications between Bullet Physics' update cycle and Unity 3D's update cycle, and a fully functional editor interface.

3.2 PROPOSED DESIGN/METHOD

This project is divided into a few sections: rigid body physics, soft body physics, inverse kinematic physics, user interface, and update management. The idea for this project is to subdivide these portions into the main steps, each of the physics components. Each of the components are being developed using a combination of agile development practices combined with extreme programing principals.

The update management system is on the most critical parts of this project and are being designed to use a message board system that is integrated with the Unity 3D update system. Within Unity's update cycle partial physics updates will be performed and during the fixed update cycle (occurs at a fixed interval of time default timing will be locked to 1 / 60 of a second) where a complete physics will occur. Before the Bullet Physics update happens there will be a message broadcast to all Bullet Physics objects for a before Physics update, this update will return in a concurrent method that can be interrupted for the physics update. After the physics update is completed a message will be broadcasted for an after physics update. This system will allow any object that does not need physics updates to remain within Unity's update cycle. This method also can prevent update conflicts as well, such as, if a game objects state can change due to a change in velocity or acceleration.

Bullet Physics objects are subdivided into rigid body, soft body, and inverse kinematic. Each of these objects will constructed using a component wise design pattern. This separation is also being done so that the physics update system can be controllably restricted to conserve system resources; this is being done mainly due to the complexity of performing soft body physics calculations.

3.3 DESIGN ANALYSIS

So far this project there have been many designs that have been tried. The first idea was to use a message bus system that would have sent messages during Unity’s update and fixed update phase based on how much time had passed; this method did not work due to increase overhead needed to process all of the messages being passed. The next method was simply just assuming that the Bullet Physics update occurred at the same time as the fixed update, both actually do trigger at 1/60 of a second when the frame rate is locked to 60 frames per second; this design produced some unexpected conflicts where behaviors such as state changes that were reliant on movement based attributes could become skewed.

Currently the system is designed using a message board design where a physics manager broadcast a message to passive listening physics objects that then trigger a physics update. This system also allows for partial physics step processing to minimize potential issues of ting up too many resources that would also be needed for rendering. This method also as a few other methods such as being able to separate collision methodology and which types of physics can be processed. The ability to separate these functions is incredibly valuable due to the fact that a developer may not need to spend resources on a more exact collision detection system or have access to hardware capable of performing the more mathematically intense calculations needed for soft body physics.

4 Testing/Development

4.1 INTERFACE SPECIFICATIONS

This project requires use of the paid version of Unity 3D version 5.4 or higher. Due to the fact that the Bullet Physics Library is written in C++ the free version of Unity 3D and all version prior to version 5 cannot support using C++ DLLs. Another specific system requirement for this project to run properly is that the system that is running the plugin must have a GPU, even though the plugin will work on a system without a GPU it will not show the enhanced ability to calculate physics.

4.2 HARDWARE/SOFTWARE

Testing and development for this project is divided into two phase: the plugin phase and the deployment phase. For the development phase the plugin is being developed on a Windows machine that meets Unity 3D version 5's. The deployment phase is a bit more involved requiring the following platforms for testing: a Windows PC, a Mac, an Android device, an IOS device, an Xbox One, and a PS4. I would also like to make sure that the plugin works with both the Xbox 360 and the Ps3, but it is not a priority. Testing on Nintendo platforms has an added complication due to required licensing needed to deploy applications to them, the current plan is not to have this plugin compatible with Nintendo's current hardware; this doesn't mean that there won't be an attempt to test this but that will be done at the last moment in March of 2017.

4.2 PROCESS

Testing for this project is being done using a combination of Unity 3D's automated testing and user testing. This project cannot be 100% tested by using automated testing methods due to how automated testing systems can interfere with the precise timing needed for this project, hence why some user testing is actually needed. Debugging tools are also being developed to log information such as time, frame rate, input, collision flags, and so on. These debugging tools are needed so that manual testing will yield results that can actually be analyzed and constantly repeated.

5 Results

The main result so far is to use a message bus system that allows for a component based design method for the update system. This system allows for maximum flexibility with the most control over when physics updates actually occur. This also allows for the expansion of different broad phase collision detections to be implemented as well which the developer would have the choice for which one they would want to use based on hardware available to the system of deployment.

Currently the test setup is working correctly and is being reworked to include the features needed to access all of the Bullet Physics Library rigid body functionality.

6 Conclusions

At the current moment the collision shapes and rigid body data structure are nearly complete as well as a working prototype for the update management system. The editor interface as well as the update should be completed before the end of 2016. Hopefully there will be a full fledge demo for the rigid body integration within the next 4 weeks.

The other portions of the project include integration of the soft body and inverse kinematic support which will start short after completion of the rigid body integration. With the current design of the input management there should be no conflict in timing between Unity 3D and Bullet Physics and the DLLs for each of the native platforms have already been created and tested.

7 References

Bullet Physics Library Documentation: http://bulletphysics.org/Bullet/BulletFull/

Unity 3D API: https://docs.unity3d.com/ScriptReference/

C# and C++ APIs: https://msdn.microsoft.com/en-us/library/hh846503.aspx

Unity 3D forums: https://forum.unity3d.com/

Stack exchange: http://stackexchange.com/