Bullet Physics Unity Integration

PROJECT PLAN

May 1741

Advisor: Prof. Mani Mina
Travis Evers
Tevers@iastate.edu

1 Dec 2016

Contents

1 Introduction

1.1 PROJECT STATEMENT

This project is to integrate the Bullet Physics Library into Unity 3D. The integration will allow developers to use the GPU to do physics based calculation on the GPU instead of the CPU.

1.2 PURPOSE

Unity 3D is a powerful development platform that has incredible portability as well as expandability, but has a few areas where it is inefficient. The main inefficiency in Unity 3D is how physics calculations are processed; the current method is that physics based calculations are sent to the CPU in multiple threads. By sending the physics calculations to the CPU it bottlenecks performance and prevents the platform from calculating large volumes of calculations in a realistic way. By integrating the Bullet Physics Library into Unity 3D physics calculations will be handled on the GPU instead, meaning the bottle neck for physics processing is now removed as well as being able to do more calculations per frame as well. This improvement doesn't just improve the quality of video games that can be produced using Unity 3D but can also be used for improving virtual reality training simulations as well.

1.3 GOALS

The goals of this project are to integrate the Bullet Physics Library into Unity 3D so there is no ambiguity on when the updating of physics objects happens for all of the following types of physics based objects: rigid body, inverse kinematic, and soft body. This integration will also have an overhaul to the Unity 3D user interface so developers familiar with Unity 3D will have minimum issues using the expanded toolset. This project will also be deployable onto all platforms that are supported by the pro version of Unity 3D with the exceptions of specific platforms that require individual licensing, such as, the Nintendo 3DS and Nintendo Wii U.

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

Due to the complexity of this project a general architectural view of the project is included in the appendix. I'm planning on using a combination of agile development mixed with some extreme programming practices to finish this project.

3.1 PREVIOUS WORK/LITERATURE

There is only one GPU based physics plugin for the current build of Unity 3D, Bullet Sharp. Bullet Sharp integrates Bullet Physics into Unity 3D but has a major issue with conflicting update cycles between both Unity 3D and Bullet Physics. I have also worked on a project that incorporated a smaller less complex GPU based physics engine into Unity 3D as well.

3.2 PROPOSED SYSTEM BLOCK DIAGRAM

See Appendix for the architectural description of the project

3.3 ASSESSMENT OF PROPOSED METHODS

There are two main ways to implement this system. The first method is to create an event handler that acts as an extension to the Unity 3D update cycle. This method is the harder option to keep synchronization stable but would have the least amount of overhead for processing. The second method is to create an update manager that circumvents the entire Unity 3D update cycle and replaces it with a custom update cycle. The second method is a more brute force approach that has increased overhead but it would be easily synchronized.

3.4 VALIDATION

To test to see if the Bullet Physics engine is fully integrated there are a couple test that can be performed. The first test is using the system analysis tools included with the pro version of Unity to make sure that the physics commands are going into the graphics pipeline instead of the CPU. The second test is to modify the assertion testing plugin available for Unity 3D to deal with the Bullet Physics Library. One of the most difficult things to test is making sure that there is no timing conflicts between the Unity 3D and Bullet Physics, to do this it will involve doing manual testing versus automated testing. The reason for this is the inclusion of an assertion check can actually modify the timing of the update cycle. Granted each of the individual modules of the Bullet Physics Library need to be integrated and have unique test create for them as well.

4 Project Requirements/Specifications

4.1 FUNCTIONAL

All Physics calculations to include: ray casting, collision detection, collision resolution, motion, and inverse kinematic support.

A set update cycle that does not fluctuate.

Direct integration with Unity 3D's animation systems.

An updated user interface for Unity 3D for creating/editing/manipulating Bullet Physics Objects.

Examples and demos for each of the added components from the Bullet Physics Library.

4.2 NON-FUNCTIONAL

Maintain Unity 3D's current portability.

Have the physics calculations run at a minimum of 60 FPS (frames per second).

Make the extension developer friendly, a non-programmer should be able to create and manipulate assets for Bullet Physics.

5 Challenges

This project is rife with challenges and pitfalls. The biggest challenge is the synching the update cycles this is the major hurdle of this project. Since Unity 3D's update cycle is divided into 2 separate parts: update (continually runs) and fixed update (happens at end of frame) the best approach is most likely to create an event handling system that adds two new update cycles: before physics update and after physics update. By creating two separate update cycles it will allow developers to control when specific game objects will update and wither the game object is actually reliant on the physics update process at all, for example, reading user input does not need to be linked with the physics engine but performing a state change for an object may be linked to the physics engine.

6 Timeline

BPL = Bullet Physics Library

U3D = Unity 3D

UED = Unity Editor

IK = Inverse kinematic

Design documentation is being created going into each task

Presentation are highlighted in green and will take precedence over tasks list below them. This is an optimistic timeline with built in time for design refinement and debugging.

Task	Estimated Completion Time
Optimize Bullet Sharp Wrapper	(completed)
Bullet Conversion Methods	(completed)
Collision Shapes	(completed)
Collision Shape Management	(completed)
Collision Objects	(completed)
Physics World (Collision)	(completed)
Primitive Shape Generation	(completed)
Rigid Body	(completed)
Update Manager	12 Jan 2017
Physics World (Rigid Body)	12 Jan 2017
Broad Phase Collision Detection Modules	(completed)
Fall presentation	7 Dec 2016
Drawing Utility Functions	20 Jan 2017
Inverse Kinematic Constraints	27 Jan 2017
Character Controllers	7 Feb 2017
Soft Body Physics (Ropes)	14 Feb 2017
Soft Body Physics (Fluids)	15 Mar 2017
Physics World (soft body)	30 Mar 2017
Soft Body Physics (voxel objects)	15 Apr 2017
Demo	1 May 2017
Final presentation	1 May 2017

If all tasks are finished early I would like to develop a cloth physics simulator and fluid physics simulator as well. Granted I don't believe I will have the time to get to these additional features before the end of the spring semester. As stated before this is a very optimistic timeline assuming minimal issues, but there is enough over estimated time built into the end of the project to give ample time to fix even the most sever issues.

6.1 FIRST SEMESTER

The first semester is primarily documentation and integrating the rigid body physics systems. Timing conflicts must be resolved during this time period as well. Also a general framework for the entire editor should also be established during this time period as well.

6.2 SECOND SEMESTER

The second semester is about integrating the soft body physics and inverse kinematic systems. The inverse kinematic systems have an added complication of how to integrate them into the Unity 3D animation system as well. Also demos and user documentation need to be finished during this time period as well.

7 Conclusions

This project is designed to integrate all of the functionality of the Bullet Physics Library into Unity 3D without losing the main non-functional pillars that Unity 3D stands on: portability and ease of use. All physics calculations, collision detection, and other transform operations must be performed on the GPU through the Bullet Physics Library. This integration will increase the accuracy, speed, and volume of calculations that Unity 3D can perform.

8 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/