Project Hollow Point

May 1741

Advisor: Mani Mina
Travis Evers
Tevers@iastate.edu

Revised: 21 April 2017

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 Project Hollow Point is to incorporate the both rigid body physics support as well as dynamic objects that can be used for inverse kinematic motion from the Bullet Physics Library into Unity 3D. This integration must improve Unity 3D's performance for physics based calculations while not sacrificing stability or portability.

2 Deliverables

The main deliverables of this project are the integration of the Bullet Physics Library rigid body system, collision system, and dynamic object system into Unity 3D. The added functionality must improve Unity 3D's ability to perform physics based calculations as well as maintaining stability and portability of Unity 3D. To make development easier a customized user interface is also being provided so that implementation of Bullet Physics objects is quicker and simpler to implement.

3 Design

Due to this being an adaptor to Unity 3D the general design principle of this project is to implement a component based system to fit in line with current Unity 3D systems.

3.1 SYSTEM SPECIFICATIONS

System specifics for executing this project are that the platform has the capability to run Unity 3D version 5.5 applications. Project Hollow Point's performance increase is primarily seen when the application is being run a device with a GPU.

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 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 DESIGN/METHOD

This project is divided into a few sections: rigid 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. See the attached appendix for diagrams pertaining to design and software architecture.

When discussing project Hollow Point it is important to understand the general architecture being used. The Unity 3D systems reside at the top level with direct interaction with project Hollow Point that interacts with a C# wrapper for Bullet Physics that then interacts with the Bullet Physics DLL for the given platform. The wrapper and DLLs are not part of project Hollow Point. The following platforms are currently supported with the provided DLLs: x86, x64, OSX, iOs, Android, UWP, and ARM. All the following are designed to include based on whatever platform the final project is being built for, as an example, if the developer chooses to build the Unity 3D project for the Play Station 4 the x86 DLL will automatically be included while the other DLLs will not.

3.2.1 UNIT CONVERSION AND UTILITY EXTENSIONS

Since it would be more difficult to constantly have developers familiar with Unity 3D to enter in values compatible with Bullet Physics it was imperative to write static conversion methods for the following mathematical structures: vectors, matrices, and quaternions. These conversions also work both directions so Bullet Physics mathematical structures can be converted back to Unity 3D structures just in case the developer wants to implement logic that directly effects or is dependent on Unity based mathematical functions, this was implemented to help make it easier to integrate Project Hollow Point into already existing Unity 3D projects.

To make dealing with collision flags easier extra enumerator extensions were made. These extensions allow for generic comparisons as well being able to treat the enumerations as bit mask to maximize efficiency when checking for possible collisions. The extensions also make the custom user interface in editor easier to implement

3.2.2 COLLISION DETECTION

Collison detection is divided into two separate parts: creating the 3-dimensional representation of areas that are used for detecting collisions and detecting the collisions themselves. This section will discuss the creation of collision shapes, since detecting collisions is more relevant when discussing updating.

The following shapes are supported by Bullet Physics: boxes, spheres, capsules, cylinders, and cones. The following are complex shapes that are supported by the Bullet Physics Library: triangular meshes, height field terrains, and compound shapes from any of the listed supported shapes. Since these shapes are already supported by Bullet Physics the implementation is designed to convert Unity 3D based values into Bullet Physics values and pass them through to Bullet Physics for construction.

The collision shape is stored in a collision object. The collision object as a manager for collisions on the given game object. Whenever the collision objects manifold is visited or is finished being visited an event is triggered, these events ensure that timing is maintained between Unity 3D, Bullet Physics, and every game object in the scene. Identification information is also stored in the collision object to identify collision groups that the object can interact with.

3.2.3 RIGID BODY

Rigid body objects are objects that have physics based operations performed on them. Since this is an adaptor to Bullet Physics the first step in creating a rigid body object is to convert Unity values into values useable with Bullet Physics. The values being converted include the following: local inertia, friction, rolling friction, linear dampening, angular dampening, mass, linear velocity, angular velocity, and thresh holds for how much force is needed for performing calculations. Manipulating Bullet Physics objects is done through force manipulation to the object rather than simply changing velocities. The forces can be applied both in a linear fashion or applied as torque for rotations. The forces applied can either be applied from internally in the object or applied from a position relative to the origin of the rigid body.

Construction of rigid body objects is handled by the physics world from a user given configuration. All rigid body objects are registered to the physics world in scene. Adding a rigid body is simply done by invoking the add rigid body method in the object., this add method passes the object down to the Bullet Physics. Disposing of rigid body objects involves making sure that all dynamic objects, collision objects, and motion state objects are also destroyed.

3.2.4 MESH CREATION

Since primitive objects can be created by project Hollow Point that are compatible with Bullet physics the mesh structure that is visible must also by created. A mesh for a 3D object contains the following: vertices, normal vectors, UV mapping, and triangle construction. This is the most math intensive portion of the project where the mesh structures are created using geometric algorithms.

3.2.5 UPDATE MANAGMENT

Update management is the most critical portion of this project. The update management is handled by the physics world object and the physics world helper object. The general idea is to use the physics world as a container for all Bullet physics objects, the physics world is then used as a hub for distribution of messaging to these objects. To prevent conflictions between the possibility of the user creating multiple physics world object it is also designed to be a statically accessible singleton object.

The helper object is designed to handle the interactions between Unity 3D and Bullet Physics. Unity 3D as two main types of updating that need to be addressed: fixed updates and updates. Fixed Updates sub divide the time between updates which happen once per frame just before the rendering process occurs. Bullet Physics as two types of updates as well physics steps and partial physics steps. Partial physics steps are equidistantly spaced update calls designed to lessen the work load of the actual physics step. For the Bullet physics update to perform correctly the frame rate should be fixed at run time as well as the total number of partial steps that will be taken before the final physics update is made, in project Hollow Point both values can be set by the user.

Collison handling is also handled by the physics world. Collision detection is divided into two separate phases: broad phase and narrow phase detection. Broad phase detection is used to find which objects should be checked in the narrow phase: project Hollow Point allows the user to select one of the following broad phase collision detection methods: dynamic axis aligned bounding boxes, axis sweeping (both 64 bit and 32 bit variants), and simply ignoring the step. Narrow phase collision is where collision events are actual determined and resolved based on the results from the broad phase detection. Both phases of the collision detection occur on every call of fixed update from Unity. Collision objects are notified through a collision handling event that they are registered to. The collision detection check is also called after any movement is applied to an object as well ensuring accuracy of detection.

The physics world does not remove Unity 3D's physics functions but instead makes it so that there are no objects to be processed for it. Though the option is there for the developer to mix and match methodologies which is not recommended.

3.2.6 USER INTERFACE

Unity 3D has the ability for developers to customize menus and interfaces to optimize workflow. Project Hollow Point utilizes the Unity Editor interface to add functionality in four separate areas: drop down menus, right click menus, the inspector window, and in-editor view.

For implementation, most the functionality has been constructed into a factory design pattern so that there is maximum code reusability and allows for easier extension. The basic layout utilities are stored separately and designed to be used to create both menu layouts and inspector windows.

The in-editor gizmos which have a similar look and control options to that of Unity 3D's in-editor gizmos are constructed per object. This allows users to quickly manipulate position/ rotation/ scale of Bullet Physics objects visually in a scene.

3.3 DESIGN CHANGES

Through the development process of project Hollow Point a few major design changes were made. The most notable is the dropping of soft body physics integration. This change in scope was done due to time constraints onto the project making it unrealistic to have them finished, tested, and polished enough to be included. The messaging system applied within the physics world helper was simplified from having separate messaging for visiting manifolds on partial and final physics steps. This was done because the functionality of both is identical for each phase with the difference being the difference in time potentially being longer for the final physics step in comparison to the partial steps, if the coroutine functions during the partial step take longer to process than the allotted time for them. Offline simulation was also stopped due to time constraints.

3.3 DESIGN ANALYSIS

The current design has been configured so extension is easily implemented so that the remaining features of the Bullet Physics Library can be implemented in the future as well as new features. The overall design is using a component based system that implements a message bus system to regulate timing between Unity 3D and Bullet Physics. The component based design pattern implement also makes the plugin very modular in nature making it easy to adjust it when Unity 3D is updated which may result in currently used methods being removed or functionality being changed, with one exception for this. The exception of a removed method happened when importing a user created 3D model, one of the Unity methods being called was removed from Unity 5.4 to Unity 5.5 (the current version of Unity that project Hollow Point is compatible with, project Hollow Point has not been tested for the current iteration of Unity 5.6 which released on March 31, 2017) making it so functionality for optimization of 3D models is not being performed in the current build.

4 Testing/Development

4.1 INTERFACE SPECIFICATIONS

This project requires use of Unity 3D version 5.5.

4.2 HARDWARE/SOFTWARE

Testing for this project has been developed and tested in the following software environment, 64-bit Windows 10 system that is using Unity 5.5.1; as for hardware, the system being tested with utilizes dual AMD Radeon HD 7800 in both crossfire configuration and running as standalone cards, since project Hollow Point is more dependent on the GPU than CPU it is less important to note any other pieces of hardware. Project Hollow Point as also been successfully deployed to both an OS X device as well as a Play Station 4.

4.2 PROCESS

Testing for project Hollow Point isn't as simple as being able to write assertion test and automatically test if the code is working properly or not due to how important timing is in the project itself. By inserting an assertion test it skews the timing results making the test relatively unreliable for the given results. This forced the testing to be a bit more creative with performing the test with a combination of manual input and automatically generated input and verifying changes manually and or post operation. Editor resting is also done through manual testing due to complexities associated with handling dynamic user input.

5 Implementation and Demo

5.1 CREATING A PROJECT WITH PROJECT HOLLOW POINT

Creating a new project with project hollow point is simple. To add project Hollow Point to a new or existing project the user has two main options. The first option is to simply drag the plugin directory from project Hollow Point into the new project and add a text file named smcs to the Assets directory. The smcs file only needs one line to be in it, "-unsafe" , this is done so that Unity 3D can utilize C++ pointers that are present in the DLLs. The second method is to import a custom package from the Assets dropdown menu and select project Hollow Point.

Either method will then allow the user to use project Hollow Point within their Unity project. To prevent confusion to the developer most of the functionality that is project Hollow Point is contained in its own names space of BulletUnity.

It is important to note that project Hollow point can be imported to other versions of Unity 3D version 5, but functionality is not guaranteed for any version other than Unity 3D version 5.5.1 in its current state.

5.2 DEPLOYING THE DEMO

Deploying the Demo is done by simply executing the Unity 3D application for the demo. Controls use the keyboard and mouse and onscreen prompts provide the user with controls for the given scene that they are in.

6 Results

The current results from project Hollow Point are the following: collision detection through Bullet Physics, rigid Body Physics handled through Bullet Physics, the ability to create Bullet Physics based objects and easily manipulate them in the Unity 3D editor, creating 3D models for an assortment of primitive shapes, the ability to import user generated 3D models in a way that they are compatible with Bullet Physics, the ability to create constrained joint objects, and a custom user interface that is quick and easy to use for developers. The features listed work properly while not sacrificing Unity 3D's portability and improving performance of Unity 3D's capability process more physics based calculations as well as improving the accuracy of those calculations. This is also designed to allow for further expansion as well to include soft body physics, fluid dynamics, and an over haul to the animation system in Unity 3D so that Bullet Physics and Unity 3D's Mecanim system can be integrated together as well.

Testing as shown that There is always a minimum increase in object capacity by at least 25% depending on the system and complexity of the geometry of the objects being used. When using simple spheres and boxes this increase can exceed a 500% increase in number of objects that can be simultaneously handled.

7 Conclusions

In conclusion, testing has shown that project Hollow Point has successfully integrated the collision detection and rigid body physics into Unity 3D. This integration improves Unity 3D's ability to handle physics based calculations making it more viable for real world simulation testing as well as improving the performance of Unity projects designed for entertainment purposes.

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/