An Investigation into Ragdoll Physics and Bullet Physics library on rendering performance

Madeleine Johnston  
B00268803

Table of Contents

1. Abstract
2. Introduction
3. Literature Review
4. Methodology
5. Implementation  
   5.1 Testing Rendering Times  
   5.2 Falling Cubes  
   5.3 Throw Cubes  
   5.4 Throw Ragdoll  
   5.5 Other Code
6. Discussion and Conclusions
7. References
8. Abstract

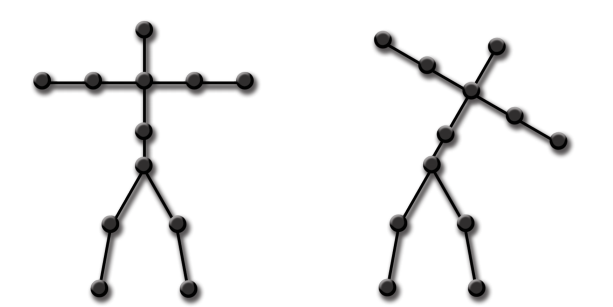
The aim of this project was to research how the standard physics implementation and ragdoll physics interact with each other when in the same environment. The level was designed to have different types of collidables to see how the interaction varies. This included varying quantities of cubes being spawned to show how light objects can be moved by the ragdoll. A solid object was included to see how the ragdoll collides with a heavier object that cannot be moved. The walls were sloped to see how objects slide on a smooth surface when the standard force of (Earth) gravity is applied. The cubes can be thrown separately from the ragdoll to display how multiple objects react when one large force is applied. The time of execution for different rendering sections were investigated to see if varying rigid body counts and the forces being applied would have an impact on performance.

1. Introduction

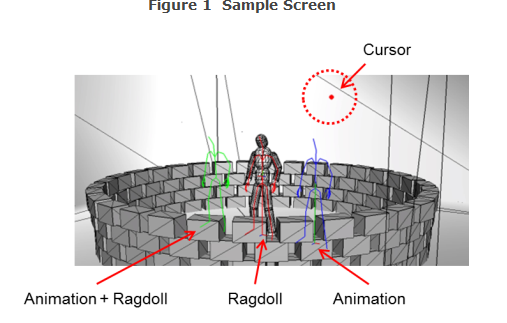
This project investigates how ragdoll physics and a standard physics implementation has an effect on how a character and solid cubes collide and the effect it has on performance. The main use of ragdoll physics in the games industry is to provide a realistic death animation of a character as it causes the model to become limp and look lifeless.

1. Literature Review

Most game engines have a ragdoll physics implementation specific for their engine and the effect they are looking for therefore there is not much documentation on a generic approach (Watkinson, 2009). The biggest difficulty for ragdoll physics is solving the constrained dynamics problem which is described as “to be able to define relationships between particles in a system, and to ensure that those relationships are never violated.” (Mulley, 2007). Mulley et al found during their investigation into producing a ragdoll simulation that defining the constraints was one of the most difficult for them to implement in their project.

  
Figure 1. Source: Mark Watkinson – Real Time Character Animation: A Generic Approach To Ragdoll Physics

Realtime animation works by creating a 3D model and animation data is stored by either defining the animations within the modelling package or using motion capture to be stored within the models files. Both the model and animation are exported to a file format and the game then imports the model to be held within the programmes memory for when needed during runtime. Typically animated bodies have points for the joints to represent where movement can happen and specify for each type of joint the constraints in place for it [Figure 1]. A rigidbody is a usually a single node in a skeleton and multiple rigidbodies make up the complete skeletal joints. This makes sure that the body does not bend into unnatural positions when a strong force or trauma is enforced on the model. The use of ragdoll physics requires a lot of conditions called constraints on the object model to make sure it does not react in an unrealistic way when under different types of forces. Standard physics libraries aim to replicate Newtonian physics and how the movement of a body reacts when a force is applied to it. This would reflect the motion of ragdoll physics which allows the bodies to be free to move within their constraints however this movement applies force upon its connecting joints and the other bodies also move around as a result. This project renders a skeleton for the character model then renders an animated skeleton with the movements incorporated. A skeleton is then rendered from the ragdoll and all three make up the combined movement and animations for the model that appears on screen. [Figure 2]

  
Figure 2 Source: %SCE\_PHYRE%/Docs/PhyreEngine\_Library\_Reference\_e.chm/Tutorials

As ragdoll movement relies on receiving force from connecting bodies, collision detection is an essential component. For collision detection to run with the least amount of processing power, it can be divided into two main steps. Following Baraffs (2001) method; first, any overlapping bounding boxes on objects are detected and all others without overlaps are no longer considered. A separating plane is used to determine if two objects are colliding. If all the vertices of object A and B are on opposite sides of the separating plane, then the two do not touch and therefore there is no collision. This separating plane is seen as a witness to two objects not colliding. This means a test can be run to see if the witness is present which in turn demines if a collision in happening or not. Once its determined that two bounding boxes are overlapping then more complex collision events can be set up to happen between the objects. In this project, the Bullet library was used for physics implementation. Bullets main author, Erwin Coumans, used to work for Sony Interactive Entertainment and offers a free open source software which is available via Github.

Mirtich (2000) explains that traditional high level algorithms that are used for rigid body simulations cannot maintain good performance rates when a programmes runs hundreds of moving and interacting bodies which is a standard for modern games. This investigation times how long the rendering process is for different sections of the programme and compares the results when increasing numbers of rigid bodies are incorporated. The rendering times are also compared when a large number of rigid bodies interact and collide at the same time.

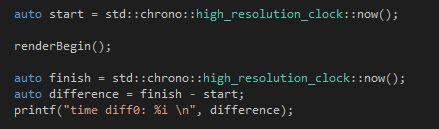
4. Methodology

Visual Studio was used to write and edit the code for the program. The project files are then run through Playstation Neighbourhood which connects it to the Playstation development kits. This application also displays the debuggers, console screen output and can give remote controls on screen to mimic the use of a Playstation DuelShock controller on the software. The PhyreTextEditor was also looked into for displaying text on screen and the PhyreLevelEditor was looked into for level design and editing. As starting to create ragdoll physics would require a larger time constraint that this project would allow, a sample code of basic ragdoll physics was used as a base from the Sony’s sample code files. The sample code also provided a simple environment which could display how the ragdoll and other objects in the environment will react to a range of terrain including a slope, unmoveable objects and moveable objects. The tutorial helped understand how the ragdoll method works and then allowed for testing to be carried out through modification of the code. During development, since the project was completed using unfamiliar software, a trial and error method was used during prototyping. This allowed for unforeseen issues to be discovered early on before a lot of time was spent developing code for a section that cannot work. An example of this was when using the PhyreTextEditor as a lot of time was spent setting up the correct files however in the end it was unable to compile the external dependencies. After it was thought that there may be an issue using the PhyreTextEditor with the type of project that was used from the tutorial. Known as incremental design, by designing the system on a daily basis instead of working from an upfront design allows discovery of design faults on an immediate basis rather than towards the later end of development.

1. Implementation

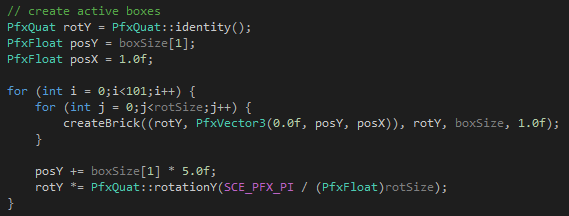
5.1 Testing Render Times

For testing purposes, a method was set up to get the time of execution for different steps in the rendering process. To utilise the clock function, the Chrono library was used. Chrono is a useful library to use as it eliminates the fact that timers and clocks may be different on separate systems. The library provides a precision-neutral concept by separating duration and point of time from specific clocks. A start and a finish parameter was created to capture the current time and a difference parameter holds the information of finish time minus start time. This difference parameter now holds the total time it takes that function to execute and it is printed to the debug line for testing purposes.

  
Code Snippet: Testing Render Times

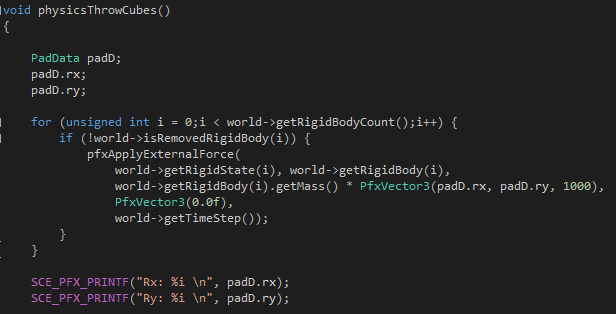
5.2 Falling Cubes

Here the number of cubes can be altered through the use of a for loop which checks if there are less than the desired number of cubes and if so, produces more cubes. The positioning of the cubes allows them to fall from the top of the screen to show the effect standard gravity has on them.

  
Code Snippet: Falling Cubes

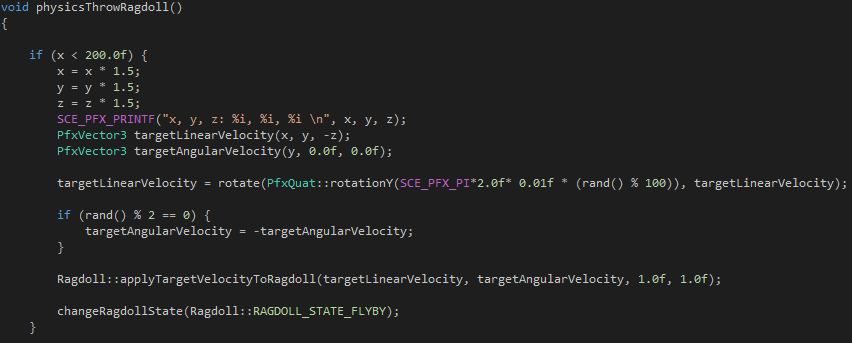
5.3 Throw Cubes

The cubes are thrown using the values from the right analogue sticks x and y value. This feature was implemented to vary the force the objects have exerted on them depending on the users movements on the controller. To obtain the values for the right sticks x and y values, the PadData structure was used. The values for the x and y co-ordinates can be between 0 and 225. The function will only apply force to active rigid bodys therefore when running the application, if the cubes have remained untouched and become inactive, then no force will be applied. SCE\_PFX\_PRINTF was used to output what the x and y values were for that loop.

  
Code Snippet: Throw Cubes

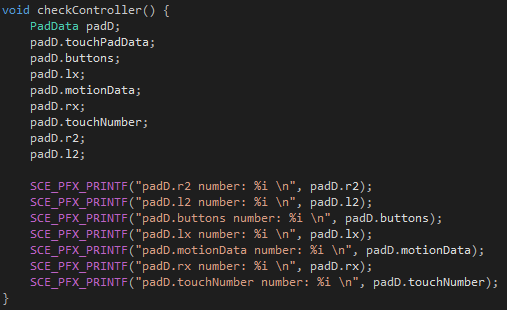
5.4 Throw Ragdoll

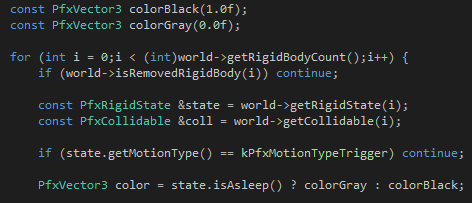
For testing purposes, the calculation for applying force on the ragdoll can be increased each time force is applied. This was to see if there was any effect on the applications ability to handle large amounts of force being applied to the ragdoll without it breaking any constraints. While the x force is less than 200, each x, y and z value are increased by 1 and a half times. With each loop the target linear velocity is increased which causes stronger forces being applied to the ragdoll. The ragdoll state is then changed to reflect the force being exerted on it.

  
Code Snippet: Throw Ragdoll

5.5 Other Code

To see which values were returned for each controller input, a checkController() function was created. This helped in choosing which control buttons would be best to offer a constantly changing value for the code for throwing the cubes. When the cubes become inactive, they change colour. The base code changed from white to grey however this was not apart enough to see when cubes were active or not. Therefore the colour was changed to black when the cubes became inactive to display clearer the changes of states.

  
Code Snippet: Other Code – checkController()

  
Code Snippet: Other Code – Colour change for state

1. Discussions and Conclusions

The report obtained results about the rendering performance times with different variants applied. The first test found the average performance times for 10, 100 and 200 cubes. For consistency, the times were recorded after the first 400 frames which gave enough time for all the cubes to spawn on screen. The time of execution was recorded 3 times for each test to produce an average. The test was then carried out 3 times and the average from all the tests were recorded. The total bytes of all the rigid bodies being rendered was also recorded however this only changed when the number of cubes were increased therefore it was only recorded 1 time.

6.1 Changing cube numbers

These results show that as the total rigid body bytes increases, the processing time for RenderBegin() and creating the rigid bodies increases. With 10 cubes the average render time for RenderBegin() was 30.43ms whereas when 200 cubes were implemented, the RenderBegin() average time was 65.16ms. The rigid bodies went from an average of 5033.4ms for 10 cubes to 9244.43ms. The simulation average time also increased with the rigid body count from 0.67ms for 10 cubes to 6.91ms for 200 cubes. However, surprisingly the time it takes to finish rendering decreases as the rigid body count increase. For 10 cubes the average was 14606.3ms whereas for 200 cubes the average was only 455.53ms. [Table 3.1, 3.2, 3.3]

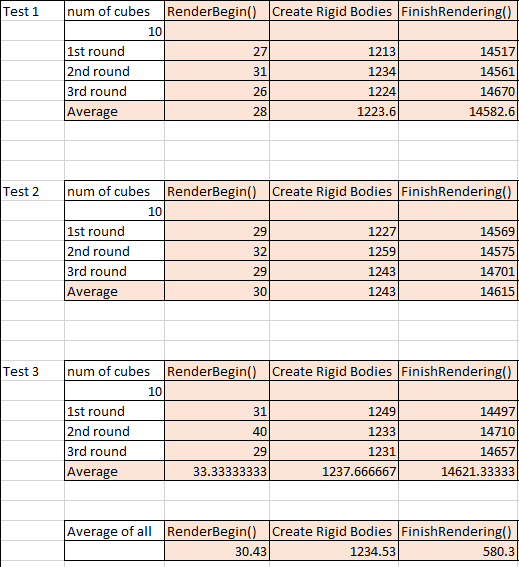






Table 3.1

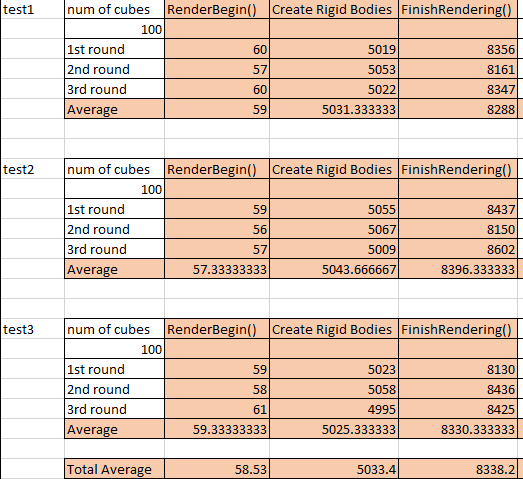




Table 3.2

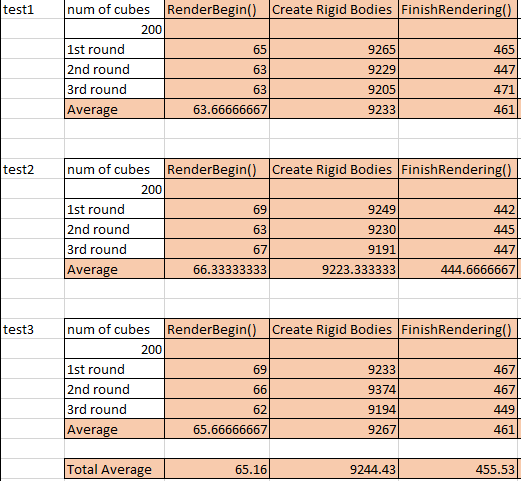




Table 3.3

6.2 Applying force to cubes

When testing the performance of applying force to the cubes, 100 cubes were rendered and the rendering times were recorded immediately after the force has been applied. To make this happen, a printf() was used once the button for applying the force had been pressed. The right analogue sticks x and y values were displayed to show the change of force being applied. The results show that the average times were roughly the same as when 100 cubes are rendered and no action is taken upon them. There is a slight increase in the RenderBegin() time which may be due to having to render multiple cubes changing movement however the increase was not substantial. [Table 4]

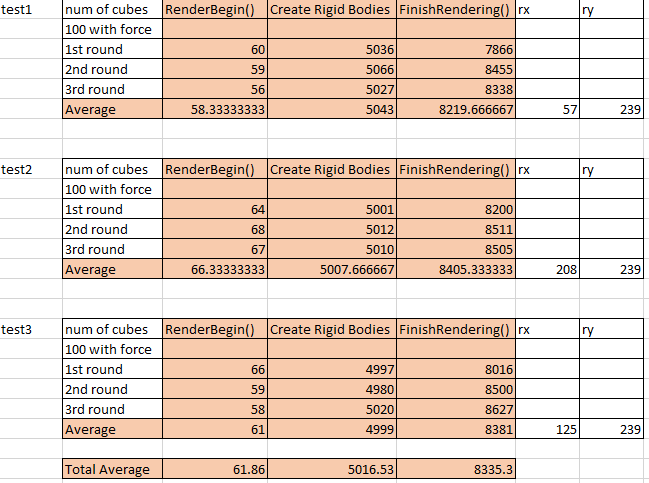




Table 4

6.3 Ragdoll Force Increase

When changing the force of the ragdoll, the smallest 2 y and z values were compared against the highest y and z values allowed inside the loop which allows the Z count to reach 200. There were small increments in the rendering time for RenderBegin() and for creating rigid bodies as the force being applied was increased. There was no notable change to the finishRendering() times. [Table 5]

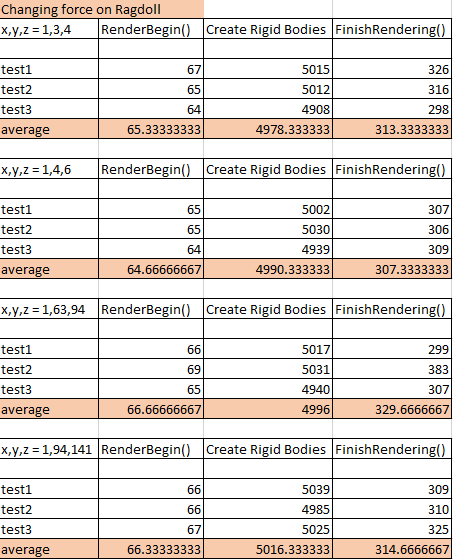


Table 5

As the tests ran during this project were small and designed to be specific for this project, they have little strength in determining the change of rendering times as the rigid body count increases. During the project, several issues arose which slowed down development time or were unsuccessful. Initially the PhyreLevelEditor was going to be used to change the environment from the sample project however as it was using the sample files from the SCE documentation, it was unable to integrate the level editor tool. The same issue appear for using the PhyreFontEditor as this was unable to be integrated into the project. The files were set up and were able to locate the external dependencies however it was unable to compile them properly for it to work.

There are many possibilities for the future development of the project. If the font editor was able to be integrated then the state of the ragdoll would be useful to display as well as the frame rate. This would allow the user to understand better the stages the ragdoll is going through during interactions with forces or other objects. The framerate would help understand when the program reaches it limit on rigid bodies being used which can lead to being able to optimise your code at earlier stages of development. Another future development would be to make the cubes in a constant active state and see how this effects performance. By making the cubes inactive after not being used for a set amount of time can help enhance the performance as their rigid bodies are no longer considered until they become active again which frees up processing power for the application.

References

Baraff, D., 2001. Physically based modeling: Rigid body simulation. *SIGGRAPH Course Notes, ACM SIGGRAPH*, *2*(1), pp.2-1.

GeeksforGeeks. 2018. Chrono in C++ - GeeksforGeeks. [ONLINE] Available at: https://www.geeksforgeeks.org/chrono-in-c/. [Accessed 06 April 2018].

Ghodsi, A. and Wilson, D., Physics Based Ragdoll Animation.

Mulley, G. and Bittarelli, M., 2007. Ragdoll Physics. *New York*.

Watkinson, M., 2009. Real Time Character Animation: A Generic Approach to Ragdoll Physics. *MSc. diss., Coventry University*.