What are some of the possible ways to implement collision detection systems for video games?

COMP130 - Software Engineering

1706966

March 23, 2018

This paper looks into some of the different ways to implement a collision detection system designed for use in video games, how they can interact with different components of the programme and the best practice for setting up these interactions. Insight into how this can affect the end user will be given, then the paper will take a look at how some of the industry standard game engines handle collision detection system cohesion. After this it will look at the design patterns that can be used to give a base when implementing a system that isn't pre-built. Then going on to summarise what practices should be implemented to get the best optimisation between the collision system and the other components.

1 Introduction

Forms of collision detection systems have played a part in video games since the beginning, being used to simulate the world which they are creating. The basic premise of a collision system is to detect when two objects come into contact with each other in the game world. The collision system will then send information to other components informing them of the collision with many possibilities for additional data to be sent, such as location of collision, what the objects are etc. For example in a game of pong, as the ball hits a paddle, this is detected and its information is gathered and sent to another component that changes how the ball now moves based on the information given. This is a very rudimentary example, most games being released at present have much more complex systems, making it more important to optimise how these systems work. Having only the data that is needed sent to the other component saves on memory and time.

If a system isn't working as intended or is poorly designed it can cause dissatisfaction for the end user. Things might occur such as a character being able to walk through walls, or being stuck on something that they can not see because the detection thinks the object is bigger than it looks to the user. Take "Big Rigs: Over the Road Racing"[1] as an example, the collision for the bridges in this game didn't work as would be expected, so when driving over a bridge the player's truck would fall through. There are other reasons a bad collision system could cause dissatisfaction that might be less obvious to the player. If a system is poorly optimised it can slow down the game causing lag issues and poor frames per second.

Some game engines come with ways of handling collision detection built into them, such as Unity and Unreal[2], we will be taking a look at how these systems work. When writing a programme without using a game

engine, there are a few different options available for implementing a collision detection system and how it interacts with components. Some of which being design patterns that can be taken as a base and worked into your code to fit your needs[3]. There are also some open source collision detection libraries that can be downloaded and used[4].

2 Game Engines

Game engines are pieces of software developed to facilitate video game creation, offering a wide variety of software components that are commonly used for game development[5].

2.1 Unity

Within Unity you can give objects either colliders or rigidbodies. The colliders and rigidbodies have to be set on the object and given measurements. Both of these give the object the ability to call "OnCollisionEnter" when they collide with other objects that have one of the aforementioned properties[6]. A collision class is passed when OnCollisionEnter is used, the class contains information about contact points, impact velocity and more. If you are not planning to use this information you can leave out a parameter to save unnecessary calculation[6]. Alternatively you are able to use "OnTriggerEnter" which does a similar thing but is called when a collider enters a set trigger area. Only the fact that the event has been triggered is sent as data, and that information is sent only to the collider which has hit the trigger and the trigger itself[7]. Therefore a lot less data is passed in comparison to OnCollisionEnter, and it is being sent to very specific locations. Unity has supplied us with some very easy to implement ways of checking for collisions, but it does have its caveats. Both objects need to be either colliders or

rigidbodies. One of them must be a rigidbody when using the trigger event[7]. Data that these events can supply is restricted in its customisability, having basically an all or nothing approach.

2.2 Unreal

In Unreal Engine the set-up of how the colliders interact with each other is similar to unity; all objects you want to have collision need to be given a static mesh first. A static mesh can be wrapped around an object very finely. A more precise wrap will require more resources[8]. Once an object has been given a static mesh and been set to 'collision' you are then able to designate the type of collision. There are many different types including custom grouping systems, but a few main ones will be listed here[9]. First is blocking, both objects block the other from passing, but no event is generated from the collision. This is used for when you want to stop an object but no event needs to be activated from the collision. Next we have Hit Events, similar to blocking neither object can pass through the other but this time we have an event activate. An event can be triggered on either object or both, and the event will tell any code attached to the object that it has hit another object. Information about the other object can be passed to other components. Lastly there is Overlap Events, these activate when the two objects overlap one another, so they don't block like before but instead pass through. The objects can either ignore this or send data similar to the Hit Event data[10]. Unreal seems to show more easily manageable customisation options when it comes to setting up collision on objects and how they send data to other areas. However if you want to inform an component that isn't one of those colliding the programmer would need to code that inside one of the objects.

3 Design Patterns

"In software engineering, a design pattern is a general repeatable solution to a commonly occurring problem in software design." [3]. These patterns are often used to speed up development but also to prevent issues that can cause problems later in the project. When it comes to working with collisions one specific design pattern stands out; that is the Visitor Pattern[11]. The Visitor Pattern allows the programme to traverse a data structure and operate on each element in the structure. A disadvantage to this is that for every new class type in the data structure a new Visit function must be added to the visitor interface and a new Accept function to the class itself[12]. The Visit functions are called by the visitor for every element in the data structure. The Accept functions are called on the class being visited by the visitor. With a Visitor Pattern implemented there exists the ability to check through an array of all the in game objects and call a check to see if they are colliding with another object. Using this pattern allows you to check for collision but you are not able to control the order that the visitor iterates through the data[12]. Meaning if objects are in the data structure but no where nearby they will still get checked against. With a large amount of objects in game at once this can become rather resource intensive.

4 Libraries

Software libraries contain pre written code for a programmer to add to their programme, giving them access to all the code in the library. By using libraries that have already been rigorously tested it can save time and bug issues.

With programmes having the ability to incorporate libraries smoothly into

them it opens a whole world of different possibilities[13]. With this it's possible to include a library specific to/or containing pre-built code used for collision detection. One such library is RAPID (Rapid and Accurate Polygon Interference Detection)[14] which is designed for use with objects containing oriented bounding boxes. It is difficult to use this library with alternative bounding volumes, but there are libraries available to cover most of the different bounding types. Because of the restrictions to the bounding box types, using libraries might not always seem like the best option when using different bounding boxes in a programme.

However another example library is FCL (Flexible Collision Library)[15]. Although this system was designed with physical robots in mind, it can also be used for virtual interactions as well. FCL is different because it can work with a wide class of models allowing the choices made about bounding types to be unrestricted. Although this might be expected to not be as resource efficient, the FCL library creators claim that "The overall performance of the FCL is comparable to state-of-the-art algorithms."[15]. FCL is able to compute if two objects have collided, where they did so, when they did and most interestingly the distance between objects even without them colliding.

5 Conclusion

To summarise it is possible to implement a collision detection system in many different ways, taking advantage of the system you are using, how your objects are set up and the resources available. Using a game engine for collision allows for a very easy to use built in method, but has limits on its customisability and requires the use of the game engine for the rest of the project. Visitor Patterns can be written to fit any language and is highly customisable but needs to be re-factored for every new class. Libraries can

make collision detection a breeze once the programmer is familiar with how they run, but most of them are quite restrictive only catering for a single type of bounding box detection.

During the research for this paper it became apparent there are a few good programming practices that will help towards having a well optimised collision system, these are the following:

1. Only send information where it is needed if possible - Avoiding sending extra information saves on resources.

2. Limit that information to the minimum required for the component receiving the data - In a similar manner this is another step towards lowering the amount of data being passed.

3. Code the detection system be more customisable - this is the key to letting you be more specific in what information is gathered on collision, in turn letting you dispatch that information more accurately.

References

[1] "Big rigs by stellar stone," accesed on 2018-03-22. [Online]. Available: https://en.wikipedia.org/wiki/Big_Rigs:_Over_the_Road_Racing

[2] P. E. Dickson, J. E. Block, G. N. Echevarria, and K. C. Keenan, "An experience-based comparison of unity and unreal for a stand-alone 3d game development course," in *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education.* ACM, 2017, pp. 70–75.

[3] "Source making - design patterns," accesed on 2018-03-22. [Online]. Available: https://sourcemaking.com/design_patterns

[4] "Awesome collision detection collection," accesed on

2018-03-22. [Online]. Available: https://github.com/jslee02/awesome-collision-detection#libraries

[5] D. Polančeć and I. Mekterović, "Developing moba games using the unity game engine," in *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2017 40th International Convention on*. IEEE, 2017, pp. 1510–1515.

[6] U. Technologies, "Unity - scripting api: Collider.oncollisionenter(collision)," accesed on 2018-03-18. [Online]. Available: https://docs.unity3d.com/ScriptReference/Collider.OnCollisionEnter.html

[7] ——, "Unity - scripting api: Collider.ontriggerenter(collider)," accesed on 2018-03-18. [Online]. Available: https://docs.unity3d.com/ScriptReference/Collider.OnTriggerEnter.html

[8] "Unreal static mesh documentation," accesed on 2018-03-22. [Online]. Available: https://docs.unrealengine.com/en-us/Engine/Content/Types/StaticMeshes

[9] "Unreal collision filtering documentation," accesed on 2018-03-22. [Online]. Available: https://www.unrealengine.com/en-US/blog/collision-filtering

[10] E. Games, "Collision overview," accesed on 2018-03-18. [Online]. Available: https://docs.unrealengine.com/en-US/Engine/Physics/Collision/Overview

[11] E. Gamma, *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.

[12] B. Horsfall, N. Charlton, and B. Reus, "Verifying the reflective visitor

pattern," in *Proceedings of the 14th Workshop on Formal Techniques for Java-like Programs.* ACM, 2012, pp. 27–34.

[13] R. G. Kula, D. M. German, T. Ishio, A. Ouni, and K. Inoue, "An exploratory study on library aging by monitoring client usage in a software ecosystem," in *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on.* IEEE, 2017, pp. 407–411.

[14] S. Gottschalk, M. C. Lin, and D. Manocha, "Obbtree: A hierarchical structure for rapid interference detection," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques.* ACM, 1996, pp. 171–180.

[15] J. Pan, S. Chitta, and D. Manocha, "Fcl: A general purpose library for collision and proximity queries," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on.* IEEE, 2012, pp. 3859–3866.