Author: Callan Moore

POST MORTEM

2D Physics Project

Known Issues with Release Build

The only issue that my application has in the release build is a small physics issue when two objects collide and one has an extremely low density. Box2D used small impulses to move the objects apart to not overlap yet the object with basically no weight won't move and the heavy object being heavy also won't move so the objects stay overlapping.

Problem Solving Strategies

Every time I came across an issue that wasn't immediately obvious I would take a moment to sit back and have a solid think about the problem and see if I could solve the problem on my own.

During my physics programming I would often use pen and paper to draw or write down the calculations so I may better figure them out. If no clear cut solution was calculated by me I would try a series of various calculations in a bit of a trial and error to see if I could narrow down the problem and get closer and closer to the solution.

If I was unable to solve the issue myself I would turn to google and the internet to see if someone else had come across a similar problem and had a solution.

If google failed me as well then I would ask a fellow classmate if they had encountered the problem and solved it or if they could in fact just come up with a solution that I could try.

Occasionally no one would be able to help me and so I would try to think of a way around the problem or see if the particular scenario could be done in a different way and usually I would find another way to complete my task.

Software Engineering Approaches for Validation and Verification

In all cases where I create or initialise an object I wrap a macro called VALIDATE around it to ensure that the function returns true. Returning true only happens if the entire function was completed with no errors generally using more VALIDATE checks. The lowest form of my validation was to check that a pointer was not NULL after the creation.

Any VALIDATE that fails stops the creation process and returns false up the chain of VALIDATE function right into the winMain where the program terminates. Every VALIDATE function that fails prints the file, function and line number to the debug stream for easily readable error debugging that gives you the exact location of what failed so it can be remedied.

What Issues I Encountered

My first issue of concern was that I had never used the API Box2D that I was expected to use nor any other physics API that I could use instead. Learning how Box2D could be utilised in C++ programming took a little bit to learn as many tutorials that offered good insight were written in other language which created some confusion when I tried to translate into C++.

After I was able to figure out the basics which was mainly creating the World object and how to then create fixtures and bodies it came down to figuring out which combination of joints and bodies were required to create the necessary gameplay features.

The first real issue within the physics environment that I had was that the physics world uses different units that the graphics screen space and so I had to convert unit values every time I transitioned from screen space to physics world or vice versa.

One of the main things that slowed me down was overlooking some properties in the joints and bodies I was creating such as angular damping on circular objects. Figuring out why my objects continued to spin regardless of how much friction was be acted upon it took more time than I would like to admit. This was more just failing of fully understanding the Box2D API that I was using.

The hardest was creating the spring as there was no direct spring joint like there was with the rope and pulley joints and required a combination of distance and prismatic to create. There was also little documentation online about how to create a spring object in Box2D. I first tried using a motor joint combined with a prismatic joint which created a rather soft spring that was weak yet did in my opinion create a spring object. Speaking to other classmates someone mentioned using a distance joint with a prismatic joint to create a proper spring and so I researched into that and was able to create a better version of a spring now that I knew what to look for online.

Even after I learnt more about how Box2D physics worked I would still spend a long time tweaking each body or joint to get them to act in the exact way I wanted them to. Changing values like density and friction could change the way an object acted quite a lot.

Personal Reflection

I spent a decent amount of time slowly building my framework incrementally to create and hold all my physics objects and render them to the screen. By doing this it allowed me to create more complex level design that was extremely enjoyable and easy to create more and more objects that allowed for a more fulfilling play experience.

One thing I did make the mistake on is putting too much time into a few aesthetics like trying to make the rope act as separate pieces so that it could wrap around which would make very little different to the code but give more realistic view of the rope. This cost me time that could have been better utilised increasing my knowledge of the physics environment which was the main purpose of this assignment.

One regret I have is that I created my graphics renderer using GDI rather than DX10. I thought it would be quickest to implement so that I could work on the physics being the focus of the assignment, and did not have time return at the end and convert to DX10 renderer.