Particle Hitting Glass Simulation

By Joseph Orchard
Games Academy, Falmouth University
Repo link: https://github.falmouth.ac.uk/JO278870/Comp305

Objectives:

The objective when making this artifact was to create a graphic simulation of particles hitting glass and then disappearing a few seconds after to simulate dust or sand particles hitting a glass pane.

Software and development approach:

The artifact aims to simulate collision of dust/sand particles with a glass pane, this is simulated using boxes to represent the particles and a transparent cuboid to represent the glass.

The particles spawn in random positions that is bound so each will hit the glass in different places. They approach the glass at a constant speed. An AABB collision test is run in the update function that tests the bounds of each particle against the bounds of the glass. This allows detection of when each particle collides with the glass, when this happens the particle is stopped, a delay function is triggered which then stops the particle rendering after a couple seconds of hitting the glass.

After deciding on what to create, research was done into OpenGL mainly using learnopenGL [1] to try and teach myself how to use it. I also looked at a video on transparency and blending in OpenGL [2] which I used to make the glass pane transparent.

I modified the template code that came with this module to get two models loading with one being the transparent glass and one being a box to represent a particle, I then made this approach the glass, this was then modified again for worksheet 3 to allow a larger number of boxes to be loaded.

For worksheet 4 I added AABB collision to the boxes. This took some more research from learnopenGL and an explanation that I found here [3]. I then added a delay function as well as a check when rendering which allowed each cube to stop being rendered a few seconds after hitting the glass, essentially deleting them.

Profiling and optimisation:

Throughout the development of the artifact profiling was carried out using the performance profiler that is built into visual studio. This was used to measure CPU usage. I also tried measuring the GPU usage however this never changed no matter how many models were added, I assume this is because the models were not detailed enough to require much of the GPU's power.

The CPU usage varied 5 percent at 1 model loaded to 13 percent at 10.000 models loaded during worksheet 3. Interestingly this rose to 13 percent at just 1 model loaded for worksheet 4 and stayed consistently 13 percent average ranging from 1 to 10,000 models loaded.

The added collision test added a considerable hit to performance however, during worksheet 3 which did not have the AABB collision test running it took until 10,000 models for the frame rate to decrease. However, in worksheet 4 which did have the collision test it only took 1,000 models for the frames to drop.

I also measured the hot path when there were few models loaded in both worksheet 3 and 4 nvoglv64.dll used the most resources. This is a link library that is used to handle OpenGL working with NVIDIA GPUs. [4]

In worksheet 3 when 10,000 models were loaded "glm::operator* <float,0>" took over, this is different to worksheet 4 when it only required 100 models to be loaded before glm::operator* <float,0> took over nvoglv64.dll as the most resource heavy call. At 10,000 models in worksheet 4 however LoadModel took over glm::operator* <float,0> as the most resource heavy call, I believe this was because it was called for each model, while in worksheet 3 it was only called twice no matter how many models were used.

This was because in worksheet 3 I did not need to store the vertices of each box, instead I could use one set each for the glass and the boxes and translate each box's model matrix. I needed to store the vertices in worksheet 4 to do the maximum and minimum bounds calculation for each box to handle the AABB collision test.

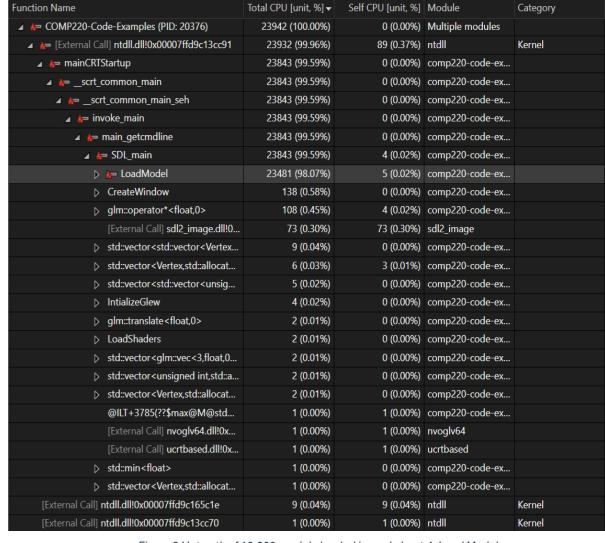
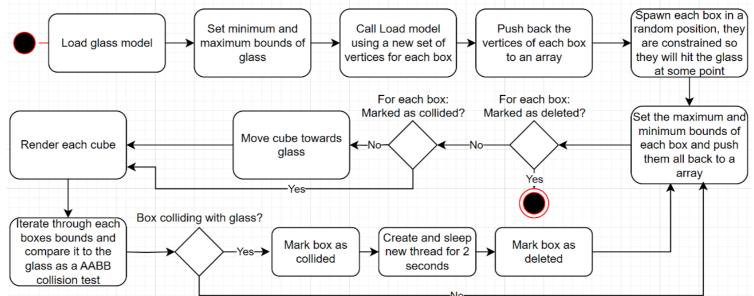


Figure 2 Hot path of 10,000 models loaded in worksheet 4, Load Model overtook glm::operator* <float,0> as the most resource heavy call

Figure 1 Image of artifact with 100 models loaded

Artifact UML Activity Diagram:



References:

- [1] Learn OpenGL, Extensive Tutorial Resource for Learning Modern OpenGL. https://learnopengl.com/.
- [2] Gordan, Victor. "OpenGL Tutorial 17 Transparency & Blending" YouTube, 28 May 2021
- https://www.youtube.com/watch?v=crOfyWiWxmc.
- [3] 3D Collision Detection Game Development | MDN. 1 Aug. 2024, https://developer.mozilla.org/en-US/docs/Games/Techniques/3D collision detection.
- [4] Pilici, Stelian https://malwaretips.com/blogs/nvoglv64-dll-what-it-is-how-to-fix-errors/ 29 June 2023