CSE4GPP Assignment Part 1 Technical Report

APIs used

OpenGL 3.1

OpenGL 3.1 was used for the *glDrawElementsInstanced* function, as it allows for many spheres to be rendered with a single render call.

GLee

GLee (OpenGL Easy Extension library), while not directly performance related it does provide easy access to OpenGLs extensions which gives access to more advanced functionality such as instantiated rendering.

DevIL

DevIL (Developer's Image Library, formerly OpenIL) is a cross platform image processing library, developed in the style of OpenGL. The main purpose of using DevIL however is simply for loading and processing the texture used for the font.

Boost

The boost library is used solely for implementing the multi-threading aspect.

Physics Algorithm

The physics algorithm implemented was the Velocity Verlet algorithm along with Hooke's Law for treating the surface of each sphere and the walls are being springy and simple Kinematic equations.

The algorithm was implemented where every sphere was compared against ever other sphere to find any overlaps as well as overlaps with the walls. If any overlaps were found, force was applied using Hooke's Law and Kinematic equations; the force was then applied to the spheres using the Velocity Verlet algorithm.

Optimizations

SSE Instructions

The optimized version of the program makes use of SSE4.1 intrinsics which takes advantage of the SSE instructions added on Intel's Penryn Core 2 CPUs in 2007. This allows for a single instruction to work on 4 pieces of data and while this has a theoretical maximum increase in performance of 4, the actual increase is much lower. The lower increase is mostly due to certain algorithms not being as efficient to implement in SSE and switching between SSE and the usual floating point model is expensive yet often required. This is one of the reasons in the physics algorithm for detecting overlap between a sphere and the walls is done using a dot product with the walls normal and the sphere's position. This keeps most of the work required in SSE and only the comparison is done on floating point. This does have the disadvantage that the same code running not in SSE is slower than simply checking a float value on a Vector3.

Multi-threading

Making use of the boost library, the physics function is broken up such that multiple threads can execute it to calculate the physics being applied to each sphere in the scene. This has the massive advantage that as CPU core numbers increase, so does the performance increase; and it was found that the performance increases scaled very well the more cores that were used. Due to the nature of how physics does not have to be 100% accurate, there is little to no synchronization between the individual physics threads and the rendering thread. The only synchronization that occurs is when changing the number of spheres in the scene.

Striding

When moving through the array of spheres, the physics function will move by n-thread indices at a time until it reaches the end. While the sphere is treated a single entity, in memory it is actually represented as two arrays, one for the positions of the spheres and the other for the remaining sphere data (velocity and acceleration). This has two main advantages, firstly when using instanced rendering, being able to not have to manipulate the sphere position data before sending it to the graphics card saves a lot of time and makes the whole process much easier. And secondly as most of the physics function spends its time check for overlaps which is primarily based on position, this keeps more of the sphere's positions in cache as they are all together in one place and can be easily shared between threads (in cache).

Instanced rendering

A specific vertex shader that supported instancing was required; this meant that an array of sphere positions needed to be sent to the shader. Unfortunately the maximum size of the array is generally less than the number of spheres required to be rendered. So the rendering function needs to be done in batches of instanced calls. In the end however even if all the spheres could not be rendered at once, generally the number of render calls is reduced hundreds of times compared to rendering each individual sphere.

As the radius and colour of each sphere was not sent to the shader, the shader has to work out which colour and radius each sphere is solely based on its instanced id, which always starts from 0 each render call. This has the side effect that the number of spheres that can be rendered has to be a multiple of 3 except in the case of the final render call to render the remainder of the spheres.

Textured font

When text needs to be drawn to the screen, it is first turned into a texture by bliting together the appropriate letters from the font image into a single image which is then turned into a texture and used to texture map a quad which displays the text. This has the advantage that rendering subsequent text that remains unchanged is very fast, however if the text is changing rapidly then the texture needs to be remade every time using many blits, which is quite slow. However overall not a lot of rapidly changing text is used in the program and as such is not an important performance factor.