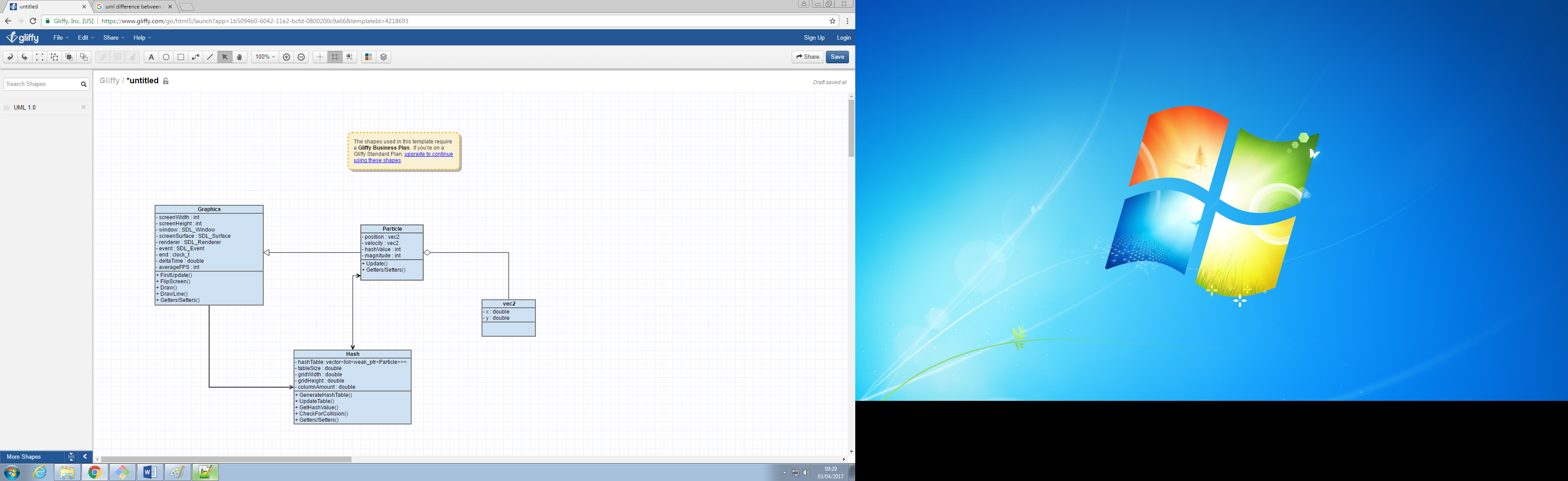
## Class Design Detail

The assignment brief was to make a spatial partitioning program which used hash tables to store the variables in each partition. This is because hash tables have an efficiency of O(1) because you can directly access the element where the particles are stored by using the hash value. Below is my UML diagram for my program.



The reason I did not bother implementing a Bucket class was that I thought it would not be necessary in order to make the code more object orientated. This is because the way that I implemented my hash table was by using a vector of lists of particles. This essentially makes the bucket class a glorified linked list, had I wanted to do more with the potential bucket class, for example changing the bounds of the buckets and have different sized buckets depending on how many particles there were in the bucket, it would be appropriate to go back and create a bucket class.

I also created a Graphics class, this was very useful because it allowed me to package away the SDL library and just directly call functions of the class. For example instead of having SDL\_DrawPixel() inside the particle class, I could call sdl->Draw(). In my opinion this is a much nicer way of incorporating a graphics library into my program.

## Implementation Details

One of a newer area of programming for me is the use of parallelising certain functions to make the program run much quicker. By using openmp I was able to make the particle update for loop parallel. This made the program a bit quicker, below the table shows the different between using serial and parallel functions by average FPS.

|  |  |  |
| --- | --- | --- |
| Type | Particle Amount | Average FPS |
| Parallel | 10k | 151 |
| Serial | 10k | 129 |

Another thing to speed up the process was by compiling and building with cmake and MinGW32 instead of using Visual Studio. By using mingw you can set certain parameters and build the program for a more efficient build. By compiling with VERBOSE=1 it will run the optimisation for the code using O3 which is the most efficient. Visual Studio only compiles with OX, which is the safest. By testing the average FPS when compiled in different modes the efficiency drops dramatically, the results are shown below. It is almost hard to believe debug mode is that bad.

|  |  |  |
| --- | --- | --- |
| Compiler | Particle Amount | Average FPS |
| Visual Studio Debug Mode | 10k | 1 |
| Visual Studio Release Mode | 10k | 107 |
| MinGW32 VERBOSE=1 | 10k | 151 |
| Default MinGW32 | 10k | 142 |

### Critical Evaluations

I think my program could have been improved by adding functionality for a 3D environment. In theory this shouldn’t be much harder than a 2D world because the logic is the same, had I had a good graphics engine I would have been inclined to develop my hash table inside that engine.

However, in my opinion the layout of the code and inclusion of shared pointers, weak pointers and openmp more than makes up for the lack of 3D. Another thing I really liked is the particles have been given different velocities, this means that they are not all moving at a constant speed which makes it look much nicer.

It is arguable that it would be good to include a bucket class for the linked list, however as justified earlier on in the report I didn’t think it would improve the readability or functionality of the program.

In general I am really pleased with the outcome of my program, I think it is really effective giving the particles in each hash different colours because it clearly shows my update table functions working as it should; the picture below shows this in place.

