# Implementation Issues

## Script Reader

The implementation of the script reader didn’t have any actual problems with the algorithm that was designed for it. The problems came purely from the implementation side. At first I was creating variables for all the shapes (e.g. Cube \_cubes, Sphere \_spheres), and as a result the algorithm for parsing the shapes couldn’t be generalised, I ended up with a lot of repeated and messy looking code. To combat this I scrapped the strategy of individual Shapes and created a combined one (Shape\* \_myShapes). This way the algorithm became more clearer and concise because I could get rid of the repeated code and easily implement a template function for parsing the shapes.  
As a result of this I could now transfer back to the caller (MyWindow) a vector of Shape (vector<Shape\*>) which meant I could get rid of all the Get methods such as GetCubes(), GetOctahedrons(), etc. This meant that I could now add more child classes of type Shape, add the relevant include file to the ScriptReader class and now make any changes to the front end (MyWindow) because I could get all the shapes using GetAllShapes() which returned the vector<Shape\*> mentioned above.

## Particle System

The major implementation issue with the Particle System was efficiency. This was always going to be the case because potentially thousands of particles could be drawn on the screen and updated per frame. Instead of using OpenGL’s immediate mode (glBegin/glEnd) vertex arrays had to be used to make significant gains in efficiency.  
Instead of drawing all of the particles in one go (because this would have broken the object-orientated paradigm of only the class itself should be responsible for drawing itself) I drew them individual. I.e. \_particle[i].Draw  
What this meant was that potentially effiency could be an issue, to combat this I supplied a method (for any developer using the class) in the ParticleEmitter class, that could get all of the particles (GetParticlesToModify), and then the developer using the class could easily draw all of the particles in one go by calling \_particle.GetVertexArray on all of the particles and building a vertex array of all the particles to be used with glDrawElements.  
*N.B. The GetParticlesToModify method is missing from the submitted code, I forgot to add it.*