# 08960 Report

Contents

[08960 Report 1](#_Toc251252110)

[Contents 1](#_Toc251252111)

[1 - Class Diagram 2](#_Toc251252112)

[2 – Class Descriptions 3](#_Toc251252113)

[3 – Sequence Diagrams 5](#_Toc251252114)

[3.1 – Load Objects and Main Loop 5](#_Toc251252115)

[3.2 – MyWindow Draw In Detail 6](#_Toc251252116)

[3.3 – MyWindow Update In Detail 7](#_Toc251252117)

[3.4 – Particle System 8](#_Toc251252118)

[4 – Design Critique 9](#_Toc251252119)

[4.1 – Design Merits 9](#_Toc251252120)

[4.2 – Design Flaws 9](#_Toc251252121)

[4.3 – What Has Changed in the Design 9](#_Toc251252122)

[4.4 – What I Would Do Differently 9](#_Toc251252123)

[5 – Project Management Reflection 10](#_Toc251252124)

## 1 - Class Diagram



## 2 – Class Descriptions

**Name**Animation  
**Role/Responsibility**Provide animations for the various shapes. Animations are based on the trigonometric ‘sin’ function.

**Name**Collision  
**Role/Responsibility**Provides box/plane collision. Used to determine whether a particle has collided with one of the planes of the box.

**Name**Color  
**Role/Responsibility**Provide an easy to use interface for providing colours. Used by the Materials class.

**Name**Cube  
**Role/Responsibility**Create, display, and update a cube.

**Name**Glowball  
**Role/Responsibility**Create, display and update a glowball. Has 6 spotlights on the surface illuminating nearby objects.

**Name**Lights  
**Role/Responsibility**Provide an easy to use interface for creating and displaying OpenGL lights (Directional, Non-Directional, and Spot lights).

**Name**MyCylinder  
**Role/Responsibility**Provide a cone as a physical representation of the spot-lights. Create, display and update a cylinder/cone.

**Name**MySphere  
**Role/Responsibility**Create, display, and update a sphere.

**Name**MyWindow  
**Role/Responsibility**Create, display, and update an OpenGL window. Updates and draws the scene.

**Name**NullObject  
**Role/Responsibility**Create, display, and update a NullObject. It has no geometry and is uses for the spot-lights to point at and track.

**Name**Octahedron  
**Role/Responsibility**Create, display, and update an Octahedron shape.

**Name**Particle  
**Role/Responsibility**Create, display, and update itself.

**Name**ParticleBall  
**Role/Responsibility**Create, display, and update a particleball. Creates 6 lots of the ParticleEmitter class to have 6 evenly distributed particle emitters across its surface,

**Name**ParticleEmitter  
**Role/Responsibility**Create particles, call their update and draw methods to create various effects (fade out over time).

**Name**ScriptReader  
**Role/Responsibility**Read and parse a script file, creating the various objects and scene. Passing them back to the caller via an Accessor method.

**Name**Shader  
**Role/Responsibility**Read in the shader files and apply them to the scene.

**Name**Tetrahedron  
**Role/Responsibility**Create, display, and update a tetrahedron shape.

**Name**Vector3f/Vector4f  
**Role/Responsibility**Supply a mathematical vector class (supporting xyz/xyzw values accordingly).

## 3 – Sequence Diagrams

### 3.1 – Load Objects and Main Loop

### 

### 3.2 – MyWindow Draw In Detail



### 3.3 – MyWindow Update In Detail



### 3.4 – Particle System



## 4 – Design Critique

### 4.1 – Design Merits

One of the merits of the design is that the MyWindow class uses the ScriptReader class to parse the script file and create the various shapes accordingly without needing knowledge of what has been loaded in. This is because ScriptReader passes back a vector of pointers to the abstract Shape class. This results in MyWindow been able to cycle through the vector<Shape\*> and call their Draw and Update methods.  
Which leads onto another merit of the design, to make the above mentioned work virtual functions where incorporated into the design. This means that calling Update and Draw on each of the elements of vector<Shape\*> calls the correct Update and Draw method, e.g. Cube.Draw() instead of Shape.Draw().

### 4.2 – Design Flaws

One flaw in the design is the fact that MyWindow is the front end and also the game engine. So instead of been able to take away the front end and replace it with something else, and/or replacing the game engine with another one, serious work would have to be done to achieve this, MyWindow is highly coupled with the game engine.

### 4.3 – What Has Changed in the Design

Initially the Platonic Solids where to be linked into one generic class and loaded in from obj files (which had to be written ourselves to represent the geometric makeup of the shapes in question). However due to time constraints I opted (for what I believe) to be the easier, yet still appropriate, option of hard-coding the geometry of the shapes themselves in their respective classes, e.g. Tetrahedron.cpp/h, Octahedron.cpp/h, etc.  
Another thing that changed from the initial design was how the ParticleEmitter class functioned. It was to have a struct in itself container the makeup of the particles. However I thought it better design to have a separate class from the ParticleEmitter called Particle, this way ParticleEmitter didn’t have to know how Particle was made up or how it worked, this meant the Particle was no longer highly coupled with the ParticleEmitter and could be modified easily without affecting the ParticleEmitter class.

### 4.4 – What I Would Do Differently

Instead of getting stuck on one particular problem and focusing my attention and time on that (i.e. the Phong Shader), I would move on to another task from the task list and better focus my efforts there and then, if time permitting, would come back to the problematic task.  
Another thing I would do differently is change the design. Instead of having MyWindow highly coupled and dependant on the game logic I would separate them out, keep the MyWindow with all the GUI items, but also create a GameEngine class for the logic side. This would result in been able to change either one, or both, without effecting the other.

## 5 – Project Management Reflection

I had some difficulties, not with planning side of project management per-se, but more with the time keeping side of project management. For instance I created what I believed to be a good Gannt Chart along with a task breakdown list which really helped me with the design aspects of the project, and which also helped me break the project down into more manageable chunks of work.  
The only downside, as mentioned earlier, was my time keeping, and what is meant by this is that I simply just didn’t stick to the Gannt Chart. I would focus on one work task, get stuck, and instead of moving on to the next one (according to my Gantt Chart) I would stay working on the problematic task, even if it wasn’t a key task that was necessary before proceeding to work on another task.  
The key lesson learnt here, is that plans (such as Gannt Charts) are valuable tools, yet the plan often needs to be revised and updated to sync with the current work and that if it’s proving too difficult, if possible, move on to another task.