Luke Monaghan - Assignment 3 - Physics API's - Concluding Document

Build Class diagrams for all objects.

A class diagram has been given in the same folder as this document.

Describe techniques used to create interactivity.

The View is static until **Right Mouse** is **Clicked** to toggle the **Flycam**.
The user can **Fire** sphere **Projectiles** into the scene at any point with **Left Mouse Click**.
The use of the F keys, from **F4** to **F10** will **Reset** the **Scene** with given elements(Feedback is given in the console).
F11 will display the current **real** and **timestep FPS** as well as any debug information for the given scene.
F12 when held will enable **wireframe** mode.

Describe the objects and where they were used.

Ragdolls have been used to add simple human like elements to the scene. When viewing just the ragdolls more will be present.
Cloth has been added again in its simple form, this is mainly because of framerate. I have cloth working faster inside the tutorial set for PhysX.
Linking of actors has been added and displayed to create a fake cloth like element. This can collide with anything else in the scene.
Particles.

> **Basic particles** have been added in the center of the scene to display basic application.
> **Water** Has been added to one corner, You will need to activate the trigger volume(shoot a ball at the box in the middle) to toggle the flow.

Actors have been added all over the place, from the plane used a a floor, to the linked actors and player controlled capsule.
Trigger Volume and **Callback** have both been used to toggle the flow of water, Callbacks are also displayed in the console when actors collide or set off a trigger event.

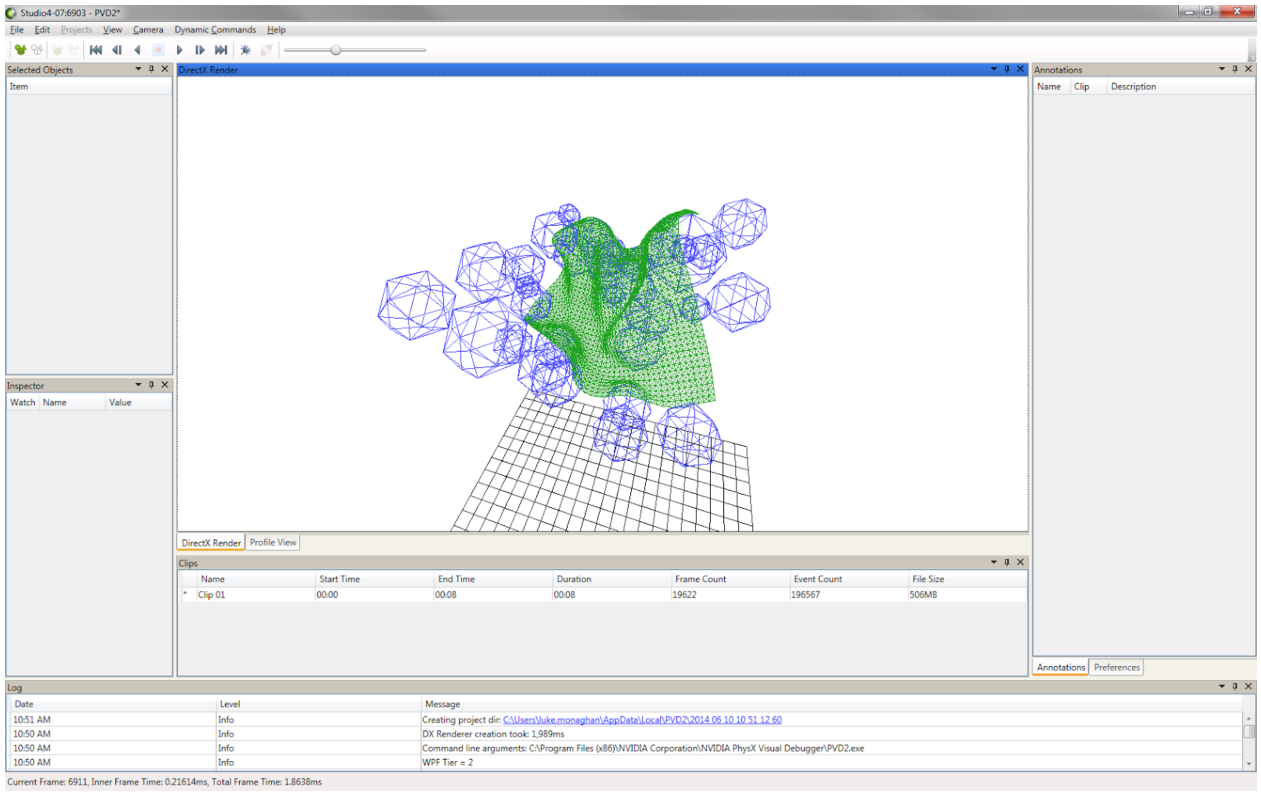
Document the libraries used.

Libraries used for this assignment are the following;

- GLFW - Context Creation and handling
- GLEW - OpenGL API
- SOIL - Texture loading
- PhysX - NVidia Cuda physics API
- AIE Framework - Wrapping general context creation and game loop.

Document your experiences of using the debugger

Use of the physx debugged was minimal for myself. Once I had visual key’s inside the application itself The use was simple for finding what would happen to elements if something went horribly wrong. Using the debugger itself is simple, open it, link it through c++ and PhysX, it just works. Only major problem I had was at home when opening it would display a blank white screen. I believe this was just an installation issue though. Performance itself was fast, the only time it was ever running at an odd speed was when the application started up. This caused the program to speed up as it had to catch up from to the AIE framework.



Clips

	Name	Start Time	End Time	Duration	Frame Count	Event Count	File Size
*	Clip 01	00:00	00:08	00:08	19622	196567	506MB

Log

```
Date        Level   Message
10:51 AM    Info    Creating project dir: C:\Users\luke.monaghan\AppData\Local\PVD2\2014 06 10 10 51 12 60
10:50 AM    Info    DX Renderer creation took: 1,989ms
10:50 AM    Info    Command line arguments: C:\Program Files (x86)\NVIDIA Corporation\NVIDIA PhysX Visual Debugger\PVD2.exe
10:50 AM    Info    WPF Tier = 2
```

Current Frame: 6911, Inner Frame Time: 0.21614ms, Total Frame Time: 1.8638ms