# 08214 Lab 0: Introduction to Simulation and 3D Graphics

## Aim

The aim of this primer is to give you an overview of what is expected of you, and what you can expect from 08214 Simulation and 3D Graphics labs. There are no programming tasks, but there are descriptions of concepts that will be useful when understanding how 3D graphics work.

## OpenGL

OpenGL is an open source graphics API. Over the years since it was introduced it has iteratively evolved. For this module we will try to only use features from OpenGL 3.0 and above. To find out more about the specification of openGL 3 go here:

http://www.opengl.org/sdk/docs/man3/

http://www.opengl.org/sdk/docs/reference_card/opengl32-quick-reference-card.pdf

Before you can send any commands to OpenGL you have to create a GL Context. Whilst this is hidden from you by OpenTK it's important to appreciate that the mechanism. A common mistake when starting out is to put OpenGL calls in the constructor of a class.

OpenGL is a State Machine. That means that by calling OpenGL commands you can change the state of the graphics card. The state is held on the graphics card and not in your code, which means that the state persists throughout the current glContext. One way of thinking about it is to think of your program like a client (or host) and the graphics card like a server (or device). This terminology is quite common, so for example if we copy data from the host to the device, or from the client to the server it means we are copying data from your program into the graphics card.

### Debugging

Debugging both OpenGL and simulation applications is hard! Useful debugging techniques include simplifying your problem to minimize complexity and workings through problems by hand to check your results are as expected.

OpenGL has a GL.GetError

## OpenTK

OpenTK is a C# library that we will be using to provide us with a window to draw into. The OpenTK package has been installed using NuGet. NuGet is a package managing system. It will install packages, copy library files and make the appropriate updates to your project for you. NuGet is integrated into visual studio 2012. You can find out more about NuGet at www.NuGet.org

OpenTK provides a lot of useful utilities that are not part of OpenGL. Things like providing a window, calling an update method and detecting key presses and mouse movements. OpenTK also provides a set of Vector and Matrix maths functionality.

## Support

**If you find yourself struggling, that's ok.** Graphics is a hard topic to master. The important thing is that you get help early. There are a number of ways that you can get support for this module.

### Labs

You should consider the labs as your primary means for support. It is likely that you will not be able to complete the work in a given lab. If that is the case you should complete the lab in your own time, and if you get stuck ask for help!

### Forums

You can get help from your instructors and peers from the forums. This is a great way to ask questions and find solutions. In fact, sometimes just taking the time to figure out how to explain your problem on a forum will lead you to your answer. In order to benefit from the shared knowledge of the forum please follow these instructions to subscribe so that when someone posts a question (or answer) you are made aware.

1) Go to the following link: http://forums.net.dcs.hull.ac.uk/ForumSubscriptions.aspx
2) In the 'Undergraduate course' section find the row for '08214 Simulation and Rendering'.
3) In the second column (see diagram below) should be a link that either says 'Yes' or 'No'.
4) If it says 'No' then click it.
5) It should change to 'Yes'. You are now subscribed!
6) Welcome to the Forums.

### SVN

A folder in a SVN repository has been created for you. You can find the repository at the following URL.

https://visualsvn.net.dcs.hull.ac.uk/svn/08214-1415/XXXXXX where XXXXXX is your username. By working with this repository instructors will be able to see your code, so they will be more able to help you with any issues. This repository will also be used for the delivery of your personalised coursework specification and feedback. If you are not familiar with SVN please look for the help documents provided by the systems team on the intranet.

### The internet

The internet is a great way to find answers and example code. You may find you run into two problems though. The first is that most examples you find on the internet will be in C++. That shouldn't pose too much of a problem, and if you read through the code and try to understand it (as opposed to just copying and pasting code you don't understand) you should be able port the code to C# fairly easily, especially with the benefit of intellisense.

The second problem that you may come across is that much of the code on the internet will be using methods that are depreciated in OpenGL 3.x, which is what we are teaching in this course. The big differences will be explained later in the lab, but a good resource to figure out what you can and can't do in OpenGL 3.x can be found here: http://www.opengl.org/sdk/docs/man3/.

This reference card is a great resource for finding out what functions are available to you, but remember the content shown in blue is removed from the OpenGL 3.2 core specification!

http://www.opengl.org/sdk/docs/reference_card/opengl32-quick-reference-card.pdf

Another good resource to help you learn OpenGL 3.x can be found here: http://ogldev.atspace.co.uk/. The examples use C++ but should not be too hard to convert to C#.

## Lab Work

The labs that have been produced for you will take more time to complete than is available in the weekly lab sessions. It is intended that you complete much of the lab work in your own time, and come to the labs with questions. Each lab also has a forum thread associated with it.

In order to help keep you on track in labs, code snippets are colour coded. Don't worry if the categories don't make sense right away. Just be aware that this is a hint as to what code should be changed. There will also be direction given in the text.

Amber boxes are used for the **OnLoad** method – getting ready to draw.

Green boxes are used for the **OnRenderFrame** method – Green for go!

Blue boxes are used for the **OnUpdateFrame** method – this is where we update the simulation.

Red boxes are used for the **OnUnLoad** method – We've stopped drawing – time to clean up.

Turquoise boxes are used for other methods, or parts of other classes – kind of a catch everything else.

Violet boxes are for code in the Vertex shader

Fuchsia (or pink) boxes are for code in the Fragment (or Pixel) shader.

## Assessment

**This module is assessed by means of a single piece of coursework that is worth 100% of the marks. If you fail to hand in anything at all, by departmental rule a re-sit will not be permitted.**

When the assessment is released it will be personalised to you. This means that you should be able to collaborate and help each other with a reduced risk of plagiarism. In order to help each other you will have to understand each other's code works and applying the same techniques to your individual coursework.

OpenGL can be programmed in many languages. In this module you are able to choose the language you work in. The learning outcomes can be demonstrated equally well in any language that has OpenGL bindings, be it C# or C++, Python or Javascript. However, the labs are targeted at students using C# and the coursework is set at a level that assumes you will use C#, which is far faster to develop with than C++.

There are strong arguments for using C++. Use of C++ is a valuable skill and is worthy of your attention, especially if you want to go on to work in an industry in which C++ is prevalent, such as games. Whilst there is a strong argument that you might want to further develop your C++ skills, it is highly recommended that you use C# for this module. Development of C# is said to be four times faster than development using C++. The tasks set for the coursework are set at an appropriate difficulty assuming that you are using C#, and benefiting from its friendliness. The time you have to work on your ACW is limited, and it is highly likely that you will be able to demonstrate more of the learning outcomes and achieve a better grade if you use C#.

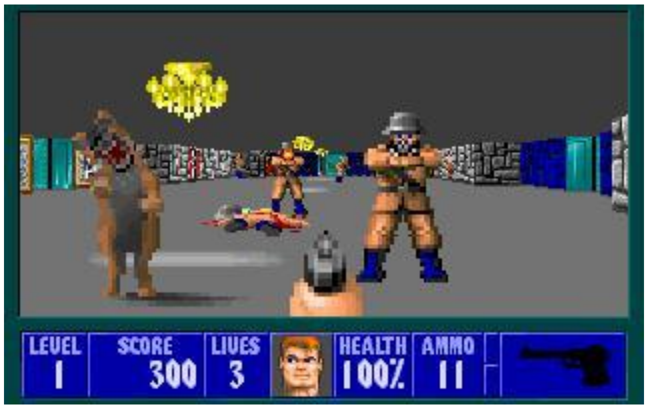
Using C# also saves you having to go to the effort of writing your own maths functions, and these are provided by the OpenTK library. If you want to do your assessment in C++ as a portfolio piece it is highly recommended that you do your assessment in C# and port it to C++ at a later date.

The coursework will be handed in using E-Bridge. Often ACW Hand in mechanism struggles with large files so before handing your code in you should be sure to clean your solution, which removes any files that are generated as part of the compilation process. Be sure to give yourself plenty of time to hand your work in.

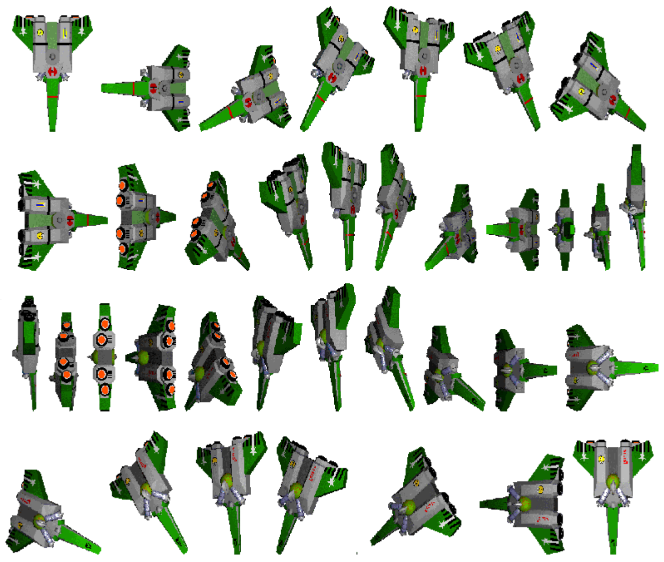
## The Evolution of Graphics Hardware

This section briefly describes the need for dedicated graphics hardware, and how this has evolved into the programmable pipeline that exists today.

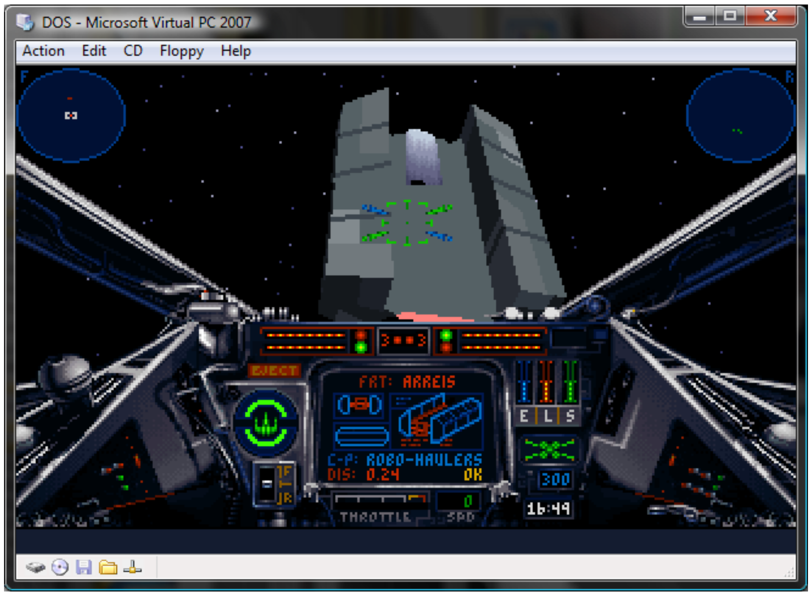
At one point processors were too slow to produce 3d graphics, but soon people started to think of ways to emulate 3d, producing 2.5D maze games like Wolf3D.



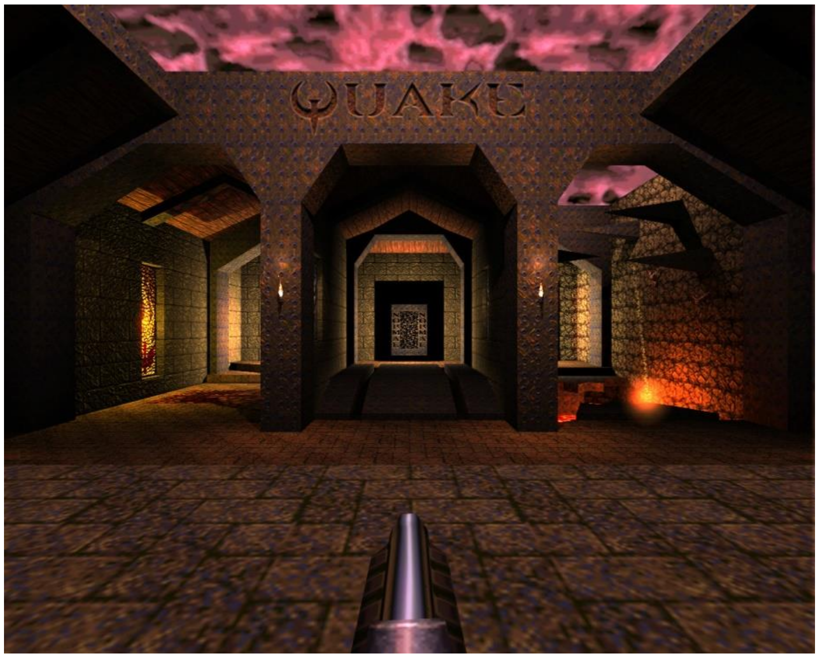
Next came 3D flight simulators, like Wing Commander, which used a sprite sheet of a ship from every angle, and fully 3D calculations to pick the appropriate frame at the appropriate scale.



As processors continued to speed up 3D games become more sophisticated, and it became possible to accurately render models in 3D space in games like X-Wing.



The sheer number of calculations necessary to achieve this was huge. You had to transform all the vertices of each model into the correct place given that both the model and the player were moving, then you had project these vertices onto the screen, and then using a scanline algorithm you had to calculate the pixels that needed to be rendered, and finally you had to calculate the colour of the pixel. In broad terms this is still how it is done today, but today we have hardware designed for this sort of work. The important thing to notice is that many of the stages of calculations are completely independent of other calculations in that stage, which means that they could be calculated in parallel, at the same time. That's what dedicated graphics hardware gave us, opening a door to a whole new world of gaming. The screenshot below is quake – the successor to the amazingly popular 2.5D doom series, and one of the first games to exploit the new graphics hardware.



To begin with, the pipeline was fixed. The way vertices were transformed was fixed. The way lighting worked was fixed. Everything was fixed. The method of programming mostly involved loading data onto the graphics card, and instructing the graphics card to send data through the pipeline. The method of loading data will become very familiar to you. Typically you ask OpenGL to generate an integer which refers to some asset (a texture, for example) loaded on to the hardware. To make that asset current you bind it. Then you can make changes, or use the bound asset. This has two implications. One is that the graphics card has a state. That state persists for the duration that the "context" exists. This means that unless you are careful you can change state from anywhere that you can get hold of the context. Effectively you have a bunch of global data, together with all the baggage that comes with that. The second implication is that, like in unmanaged languages, a programmer is responsible for removing the things that they load on to the graphics card.

As the hardware evolved more features became available in the fixed function pipeline. Coloured lights, dynamic lighting, transparent surfaces we just some of the features that were introduced, and as fast as new features were introduced programmers demanded more, or found ways to hack them to produce even more (unofficial and unsupported) effects. Eventually the ability to inject small, C like programs into the pipeline was added. The programs are known as shaders. The vertex shader is run once per vertex that is sent down the pipeline. The fragment shader is run once for every fragment of a triangle may contribute to the end scene. The important thing to remember is that much of this work is completed in parallel. You should exploit that. Whilst it's not totally avoidable spending too much time making draw calls and changing state from the cpu means that you run the risk of becoming cpu bound and starving the graphics card of data.

In more recent versions of OpenGL other parts of the pipeline have also been made programmable, and as OpenGL has evolved and become more flexible, some technologies have become useless, and have been removed from the more recent specifications. Functions are phased out through deprecation. You should avoid deprecated functions as much as possible.