

*University of Derby*

*College of Engineering and Technology School of Electronics, Computing and Mathematics*

# 5CC510 - Graphics II

*Report for Graphics II - 5CC510*

100485878

March 2020

# CHAPTER 1

## Introduction

This report will compare a range of Graphics API, and the evolution of their key features and support for writing computer games. The report will also consider their latest versions, current implementations and potential for future development.

### The Graphics API

There are many graphics API used today, some having seen use and development since the early 90s. API, like programming languages can be seen as low or high level, some are cross-platform and some are designed for use in web-browsers, based on JavaScript.

This report will focus mainly on Direct3D, OpenGL & Vulkan, while also highlighting ideas drawn from Metal (Apple’s API for their iOS, macOS & tvOS) & Mantle (an API available to AMD Graphics Cards which led to the development of Vulkan).

Initially, video games like Quake (1996) were software-rendered, meaning they would run using the CPU to render and rasterize graphics. The problem with this is that there is a huge amount of mathematical calculation being done, which is where the GPU entered. Quake received a port in the following months supporting hardware acceleration via a very early chipset known as the Rendition Vérité; offering vast improvements like fuller colour, reduced pixilation, dynamic lighting and anti-aliasing. Not only would the game look better, but it would run better. Unfortunately, the port was tailored to this specific line of GPUs, offering no compatibility with other chipsets. As the GPU market would start to increase, no games creator would be willing to write separate code for every GPU using a different API on the market. John Carmack, the lead developer, began supporting non-proprietary (or open-source) APIs.

# CHAPTER 2

## API Development

### Evolution of Direct3D

DirectX came about because of Windows. As Microsoft moved away from DOS, which allowed access to all hardware and on to Windows, where access was restricted to these, there became a need to standardise a way of accessing the GPU to offload all heavy graphics computation to create better graphics.

It started off as Direct2D, part of the “Game SDK” on Windows 3.1 and started to gain popularity. At this time however, Microsoft was working on porting the rival OpenGL to Windows95 as it could handle 3D graphics. ‘This project failed but led to the development of Direct3D, which included a joystick control applet and support for Intel’s MMX technology, with later bugfixes.’ (Eisler, Craig's Musings, 2006)

DirectX 4.0 would contain logic for use in laptop video cards. However, with the latter being delayed, it never released. During or soon after this time, DirectX 5.0 was being developed with force feedback support for joysticks and gamepads, multiple monitor support, a new control panel for game controllers, a better User Interface and better MMX support which made its way into Windows 98.

Future versions improved upon this framework with continuous boosts to performance, and additions like bump mapping, texture compression and stencil buffers. These are methods still used in versions 20+ years later. With further improvements to DirectX 7.0 and 8.0, it added pixel and vertex shaders, better lighting as well as increased performance, although was considered unpleasant to use. DirectX 8.1 addressed usability and became the basis for the very successful XBOX’s API, from Microsoft’s collaboration with nVidia.

DirectX 9.0 released with a new version of the High Level Shader Language (HLSL) and shader model 2.0, MRT, an invaluable feature for real time shading and performance.

DirectX 10 came with big changes. Many features from D3D9 were deprecated, including fixed-functions, had a much cleaner rendering pipeline and massive performance boosts as predicated rendering allowed occlusion culling, which only renders what is visible. This meant that environments could be much vaster and / or have much more complex geometry per scene.

DirectX 11 allowed for tessellation, a method of using polygon data to produce extra detail. This can provide smoother shading from a lower polygon count, maximising quality and performance.

|  |
| --- |
| <https://guardianlv.com/wp-content/uploads/2014/04/Tessellation-Stones.jpg> |

Figure 1 - Tesselation

### Evolution of OpenGL

OpenGL began as a cross platform, open source version of an API designed for use on specific chips. It has since become a popular alternative to Direct3D, as it is language independent and cross platform.

1997 saw the first update to OpenGL, giving important data types such as textures & vertex arrays. Early days of OpenGL were ahead of Direct3D, developers were known to have preferred working with it over early Direct3D. For example, OpenGL 1.3 Saw Multitexturing, Direct3D did not achieve this until version 5, however most GPUs on the market could not keep up with the performance loss from multitexturing.

Versions 1.4 and 1.5 in the following years allowed for occlusion and shadows, greatly improving the level of detail and ambience of a game world.

OpenGL was updated to version 2.0. This replaced the very low-level shading languages with a high-level language. This allowed for direct control of hardware. Version 3.0 fully deprecated many features in favour of a shader-based approach. Version 4.0 Included tessellation and was designed to support Direct3D. Various updates have been released supporting later Direct3D packages and greater efficiency.

### Birth of Vulkan

The Khronos group, who manage development of OpenGL since 2006, announced the Vulkan API in 2015. It was initially referred to as the “Next Generation OpenGL Initiative” (Khronos 2014).

AMD’s proprietary Mantle API was discontinued and donated to Khronos to devlop a brand new cross-platform, low-level API standard for the industry. The intent is to unify the API between high-end graphics cards and mobile device hardware, to run on Android, Linux, Windows & iOS & macOS systems across the board. Reduce the load on CPU and work better with the GPU to share computation. Also, unlike Direct3D11 and OpenGL4, scale better with multi-core CPUs, current APIs are initially designed for single cores and thus do not maximise the potential of multiple cores. It also uses a shader language more like Direct3D’s HLSL, allowing for a variety of shaders and faster optimisation.

# CHAPTER 3

## Latest Version Features

### Direct3D 12

The latest release of Direct3D includes a rebuild from the ground up to unify PC game graphics development. Changes to the rendering pipeline and code structure are present with increased performance and developer tools.

One of the biggest and most acclaimed features is Ray Tracing support. Although ray tracing is not a new concept, it is a new addition to games. It has been used previously in CGI for movies, to create lifelike simulations of reflections and lighting. The computational complexity is rather large, and with the latest (at the time of writing this report) series of nVidia chipsets, provide the necessary performance to do this in real-time.

Among the features is Variable Rate Shading, enabling shaders to be toned down in areas where it is not needed, and increase in areas of focus; increasing performance and enhancing the image in VR cases.

### OpenGL

### Vulkan

Although Direct3D first contributed ray tracing and multi-GPU functionality to modern graphics, Vulkan 1.1’s release integrated the same ideas into its arsenal. The multi-GPU features added allow for splitting workload between different GPUs, which could make use of an integrated Intel or AMD GPU which are normally unused in most gaming systems.

Vulkan is currently supported on Windows, Linux & Android, giving a huge pool of platforms on which to release games, whereas Direct3D is limited to Windows & Xbox.

# CHAPTER 4

## Conclusion

## Bibliography

1. Y. Achdou, F. Camilli and I. Capuzzo-Dolcetta, “Mean Field Games: Numerical Methods for the Planning Problem”, *SIAM Journal of Control and Optimization*, vol. 50, pp. 77-109, 2012. Available: 10.1137/100790069.
2. R. Albert, H. Jeong and A. Barab´asi, “Error and attack tolerance of complex networks”, *Nature*, vol. 406, no. 6794, pp. 378-382, 2000. Available: 10.1038/35019019.
3. F. Bagagiolo and D. Bauso, “Mean-Field Games and Dynamic Demand Management in Power Grids”, *Dynamic Games and Applications*, vol. 4, pp. 155-176, 2014. Available: 10.1007/s13235-013-0097-4.
4. A. Baraba´si and R. Albert, “Emergence of Scaling in Random Networks”, *Science*, vol. 286, no. 5439, pp. 509-512, 1999. Available: 10.1126/science.286.5439.509.
5. M. Bardi, “Explicit Solutions of Some Linear-Quadratic Mean-Field Games”, *Network and Heterogeneous Media*, vol. 7, no. 2, pp. 243-261, 2012. Available: 10.3934/nhm.2012.7.243.
6. A. Barron and J. Plath, “The evolution of honey bee dance communication: a mechanistic perspective”, *The Journal of Experimental Biology*, vol. 220, no. 23, pp. 4339-4346, 2017. Available: 10.1242/jeb.142778.
7. D. Bauso, “Consensus via Multi-population Robust Mean-Field Games”, *Systems & Control Letters*, vol. 107, pp. 76-83, 2017. Available: 10.1016/j.sysconle.2017.07.010.