Evolution of Graphical Processing Units (GPU) and GPU Programming

Leon Johnson
963653

Submitted to Swansea University in fulfilment
of the requirements for the Degree of Master of Science

Swansea University
Prifysgol Abertawe

Department of Computer Science
Swansea University

May 7, 2021

Contents

List of Figures

Chapter 1

Introduction

The Graphical Processing Unit (GPU) has established itself as a ubiquitous element within modern day computer architecture. Initially designed to receive data from the CPU to process and translate into images that are then to be rendered to a display - the GPU has now evolved to a mass computing tool, involved in the most advanced computer science disciplines such as Machine Learning, Artificial Intelligence and Big Data. The nature of the device is that it is highly computationally parallelizable, this is the core principle that has led to the vast expansion of its applications within the computer science industry. Programming languages, libraries and techniques have been developed and adapted simultaneously with hardware development of the GPU to ensure programmers can harness the computational capabilities the device possesses. This tandem-like cycle between the GPU itself and the art of programming has been constant throughout its lifespan constantly pushing it to new limits.

This literature explores the journey the GPU has taken at a technical level, discussing advances in hardware, materials and changes in both physical and conceptual architectures, as well as the relationship between related software and hardware development. A discussion relating to the historical timeline of the GPU will be held, with deep exploration of climacteric time periods of technological advancements. In addition, we look to explore corporate influence on the technologies throughout its history as well as its economic impacts. We also identify and discuss milestones reached by GPU technologies as well as the opportunities it has unlocked both scientifically and commercially.

Chapter 2

Origins of the GPU

This chapter identifies predecessors and similar technologies to the GPU inspiring its creation, as well as its earliest instances, with examination of the key events that led to its invention.

2.1 Whirlwind I - 1951

Widely considered to be the earliest instance of a 3D graphics system, Whirlwind I - a flight simulator constructed by MIT during the Cold-War era to be used by the US Navy [1]. The device was designed to execute every computational cycle in *bit-parallel* across sixteen units, making it effectively sixteen times faster than other machines that were to run in *bit-serial*.

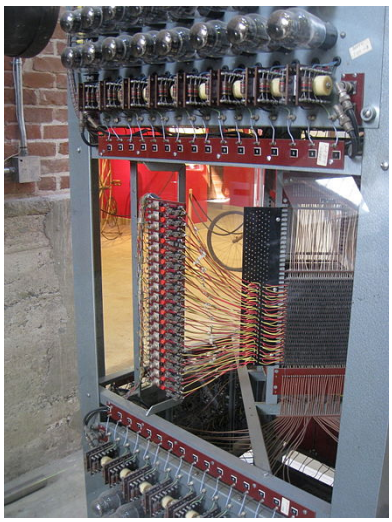


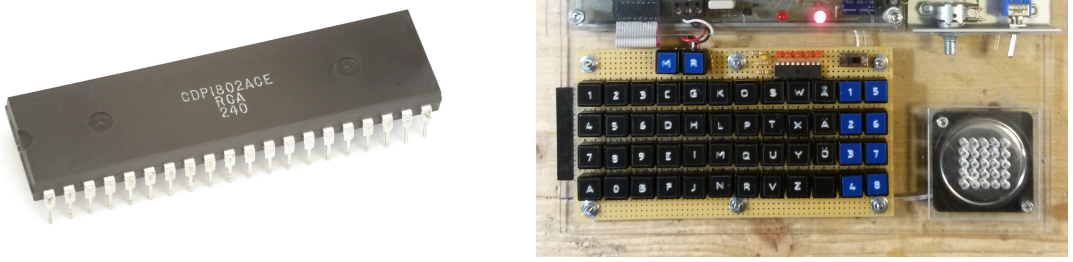
Figure 2.1: Memory Core of Project Whirlwind

2.2 Video Shifters & Video Address Generators

The mid-70s saw the emergence of the world's first commercial video game consoles. This is the era in which the foundations for 3D graphics were developed. Display controllers, namely Video Address Generators and Video Shifters were designed - essentially used as a connection between the Central Processing Unit (CPU) and the display. Typically, data streams were transformed to bit-mapped video output values, such as color and luminosity.

2.2.1 RCA CDP1861 - 1975

The *RCA CDP1861* also known as the *'Pixie'* graphics system was a supporting chip for the RCA 1802 microprocessor. By using the 1802's integrated Direct Memory Access controller it could display monochrome (black & white) bitmapped graphical output at 64x128 (where monochrome pixels occupy one bit each) [2]. The device was used in microcomputers such as the *Telmac 1800* and the *Oscom Nano*



A. Telmac1800 w/ Display

B. Telmac1800 w/ RCA 1861 Chip (Red Outline)

Figure 2.2

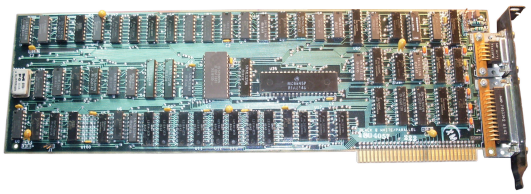
2.2.2 Television Interface Adapter (TIA) - 1977

A notable video shifter which soon followed was the *Television Interface Adapter* (TIA) - “a truly unique component” [3] which was embedded within the Atari 2600 games console released in 1977. The TIA displays a range of color palettes - varying with respect to the format of the television signal used, and also provided input handling from controllers and provided audio output.

2.2.3 IBM’s Monochrome Display Adapters (MDA) & Color Graphics Adapter (CGA) - 1981

Shortly after came the *Motorola’s MC6845* video address generator. The MC6846’s "main function is to form the address code of character buffer and attribute memory, and generate horizontal and vertical scanning synchronization" [4]. The job of delivering the video values to a Cathode Ray Tube (CRT) is assigned to other components, and is why it saw implementation in larger circuity, such as IBM’s MDA and CDA Display Adapters of 1981.

The MDA and CGA were drivers for the *IBM 5151 Monochrome Display* and *IBM 5153 Color Display* respectively. The MDA was tailored towards character based applications such as “word processing, spreadsheets, and software development”, due to its character fidelity and resolution being “excellent by the standards of the time” [5]. However due to the CGA’s functionality requiring larger computational expense due to deployment on RGB displays, character quality was lesser than its MDA counterpart. As a result, MDA’s applications involved “education, games and business graphics”. [5]



A. IBM Monochrome Display Adapter

B. Indiana Jones and the Temple of Doom in CGA

Figure 2.3

2.2.4 Alphanumeric Television Interface Controller (ANTIC) - 1978

ANTIC is a integrated circuit devoted to generating graphics to a computer display or a television - created in 1978 for the Atari 8-bit family of personal computers and the Atari 5200 [6]. ANTIC's role is to generate "playfield graphics" (i.e Walls, Backgrounds, Scores etc.), but will not process the color. In order to process the color the ANTIC passes the data to the target "Color Television Interface Adaptor".

2.2.5 High-Performance Graphics Display Controller 7220 - 1980

The High-Performance Graphics Display Controller 7220 (also known as PD7220) is a graphics controller capable of drawing basic geometry as well as character bit maps to a display. Developed by NEC Corp. to effectively display Kanji (Japanese characters) meant that the sharpness and resolution of text was very advanced for its time.

The chip was later licensed by Intel, named the 87220. The chip later became integrated within the *"iSBX 275 Video Graphics Controller Multimode Board"*, which hosts 32 kilobytes of on-board memory and provides a resolution of 256x256 on colored displays, and 512x512 on monochrome displays. [7]

2.3 The Birth of Array Technology Inc. (ATI) - 1985

In 1985, three Hong Kong nationals residing in Canada founded Array Technology Inc., initially operating within the Original Equipment Manufacturer (OEM) field with clientele such as *IBM* and *Commadore*. The following year, Array Technologies Inc. (re-branded to ATI) unveiled their first product; the *Color Emulation Card* - a 16KB device capable of outputting monochrome, green, amber and white phosphor text to a monitor. In 1987 ATI deployed the VGA Wonder and EGA Wonder card product lines, in which its graphics boards and chips would dominate the industry for many years. The EGA Wonder series 1-4 boasted 256KB of DRAM, compatible with CGA, MDA and EGA, capable of outputting 16 colours at 640x350 resolution.

2.4 Graphics Hardware Companies - (1987-1990)

During 1987 many new companies were founded and new product lines shipped, such as *Realtek*, *Oak Technology*, and *Trident*; as well as this, existing technology companies would also

develop and deploy their first graphics products trying to capitalise on the newly booming market, most notably *ATI*.

2.5 S3's Porsche 911 (S3 86C911) - 1991

Aptly named after the Porsche 911 due to its speeds and performance, *S3 Graphics 'S3 86C911'* was created to accelerate Graphical User Interface (GUI) within Microsoft's operating systems and architecture. The device introduced the "first single chip 2D-accelerator" [8] spawning many imitators in which soon became one of the first standards within the graphics hardware fiel. By 1995 all major graphics card manufacturers within the industry had integrated 2D acceleration support within their chips.

2.6 Birth of NVIDIA Corporation - 1993

In 1993, Jensen Huang, Curtis Priem and Chris Malachowsky founded NVIDIA corporation. All three had worked for rival companies within the graphical hardware and microprocessor industries; all shared the belief that focus within the next era of computing should be focused towards graphical-based computing. This is largely due to observations of the video game industry presenting the most challenging computational problems and its thriving market; as well as other computational problems being possible and realistic to solve using graphical-based approaches which general-purpose computation could not solve at the time. NVIDIA used video games as its angle into the industry, conducting large R&D projects to solve some of the problems within the field.

2.7 Era of 3D - 1995

The introduction of 3D add-on cards marked the beginning of the 'modern' era graphical hardware. With the emergence of the 32-bit operating system and the personal computers dropping to an affordable price there was a distinct opportunity for graphical hardware manufacturers to switch their efforts towards PC architecture.

2.7.1 NV1 - 1995

NVIDIA's strategic partnership with *SGS-Thomson Microelectronics* saw NVIDIA's debut product the NV1 launched. The multimedia PCI card boasted a complete 2D/3D graphical

core, with on-board VRAM or FPM DRAM memory. *NV1* utilized an interesting quadratic rendering scheme, and eventually was sold to *Diamond* and distributed as *Diamond Edge 3D*. Multiple Sega Saturn games were ported to formats to support NV1, but with the emergence of triangle polygon-based 2D and 3D accelerators the device struggled in the market which was largely due to Microsoft's announcement that their DirectX platform was based on triangle-polygon rendering methods.

2.7.2 3DFx Voodoo - 1996

Launching in Q4 of 1996 *3DFx's Voodoo Graphics* was able to be released to the consumer market due to the large price drop in EDO DRAM, in which the device typically held at a quantity of 4MB. The device was a dedicated 3D acceleration device, providing no 2D acceleration support; thus requiring a supporting 2D accelerator if required [9]. The device also consisted of a frame buffer processor and a texture mapping unit. The hardware saw use in popular game titles at the time such as Wayne Gretzky's 3D Hockey able to run at accelerated 3D speeds.

Voodoo Graphics cards led the 3D accelerator market with software developers and consumers by late 1997, taking about 85% of the market. Devices such as this caused standalone 2D render devices to become redundant.

The Voodoo2 which released in 1998 came mounted with three on-board chips and provided parallel support across its cards, one of the first of its kind.

2.7.3 Rendition's Vérité - 1996

With the combination of 2D GUI acceleration boards being used in conjunction with 3D rasterisation boards, Rendition saw the opportunity to unify the two chips into a fast, single-board solution, although slower than some of its multi-board rivals. Vérité was one of the industry's first acceptable implementations of the 2D/3D unification.

2.8 The Worlds First 'GPU' - 1999

In 1999 NVIDIA released their GeForce 256 chip, branded as a *"graphical processing unit"*, described to be "a single-chip processor with integrated transform, lighting, triangle setup/-

clipping, and rendering engines capable of processing a minimum of 10 million polygons per second".

The device containing hardware elements facilitating for transform and lighting is what separated the *GeForce 256* from its competitors at the time, as such features were usually delegated to the CPU.

2.9 The Great War: NVIDIA vs ATI - 2000

By the early 2000s, the front runners had emerged in the graphical hardware industry - NVIDIA and ATI. Both companies pushed each other to new breakthroughs within the industry with frequent graphics card releases and modifications to the graphical pipeline which would introduce new features and capabilities to their devices such as shadow volumes, vertex blending and refraction among other features.

2.10 General Purpose GPU - (2006 - present)

Until late 2006, GPUs were used to solve graphical problems such as rendering on PCs and console, however, NVIDIA and ATI (later acquired by AMD) saw opportunity to utilize the GPUs computational capabilities to problems in other fields.

2.10.1 Introduction of CUDA - 2007

In 2007, NVIDIA released the CUDA development environment. CUDA provided developers a way of programming to a GPU without the use of a *graphics API*, and also provided a way that GPU memory could be generally written to. CUDA enabled the GPU to be exposed as *"truly generic data-parallel computing device"* for any purpose on supported devices [10].

2.10.2 Introduction of OpenCL - 2009

Two years later saw the release of OpenCL, a platform which enabled development of *"applications that access all available programming resources: CPUs, GPUs and other processors"* [11]. OpenCL became supported on a wide range of devices, emphasizing on portability. CUDA and OpenCL were major platforms causing the GPU to be seen as a general purpose computing device.

Chapter 3

GPU Hardware Architecture & Assembly

The content contained in this chapter provides description and discussion of internal components of the several iterations of notable GPUs and predecessors mentioned in the previous section.

3.1 General Architecture

This subsection introduces integral components and concepts shared across many GPU implementations.

3.1.1 DRAM (Dynamic Random-Access Memory)

DRAM is a type of subset of random access memory (RAM) and is a type of semiconductor memory. DRAM stores bits of data in a combination of a capacitor and a transistor which will hold the value of 1 or 0 dependant on the voltage supplied to the components. The 'dynamic' property of DRAM is derived from the electric charge supplied to the capacitor and transistor needing to be refreshed every few milliseconds. This is due to voltage leakage from the capacitor. The benefit of using DRAM in the context of a GPU is that high memory bandwidth can be stored in DRAM (GDDR) compared to other options, and memory accessed can be amended and erased whilst threads are executed.

3.2 NVIDIA's Architecture

The following section describes methodologies used within NVIDIA's GPU implementations as well as key architecture of their GPUs.

3.2.1 CUDA

CUDA (Compute Unified Device Architecture) is NVIDIA's software and hardware architecture which enables NVIDIA GPUs to be programmable by languages such as C, C++ and OpenCL. CUDA programs call parallel kernels which execute a set of parallel threads. Kernel programs are instantiated across a grid of parallel thread blocks by the the GPU. Instances of the kernel program are executed on threads which are indexeable by threadID within its thread block, registers, program counter and other components. [12]

Thread blocks can be used in combination when resource barrier synchronization is used, and also share memory with threads within the block.

Within CUDA's parallel programming model thread blocks are contained within grids, where a grid is an array of thread blocks executing upon the same kernel program. Thread blocks within the grid have reads and writes to the same global memory unit, and can synchronize across the grid. Grids then output results into Global Memory space after kernel-wide global synchronization calls.

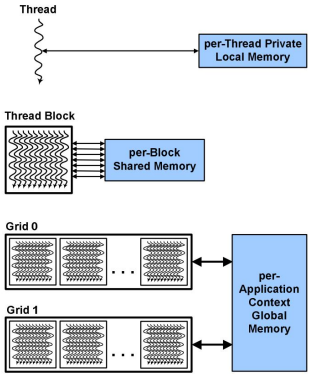


Figure 3.1: CUDA Thread, Thread Block and Grid hierarchy

NVIDIA CUDA architecture is extended via CUDA cores which are contained within SM units *(See Section 3.2.2)*. CUDA cores have units capable of executing integer and floating

point instructions, with its own logic unit, move and compare units, as well as branch instruction units.

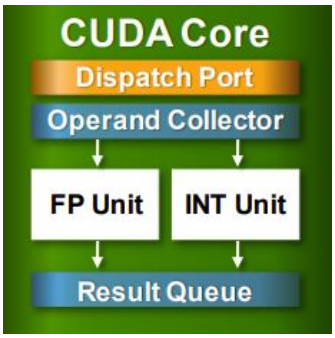


Figure 3.2: CUDA Core Architecture

3.2.2 Streaming Multiprocessors (SM)

SMs are processing units designed by NVIDIA, consisting of NVIDIA's CUDA cores. SMs have internal schedulers, registers and L1 cache memory which is shared across all CUDA cores with independent execution pipelines. SMs are the collective elements of NVIDIA GPUs that perform the actual computation [13]. There are large amounts of CUDA cores within each SM, and there are large amounts of SMs contained within the overall GPU architecture. This is what achieves the mass parallelism within the GPU, due to the massive amount of processing elements within the device.

3.2.3 NVIDIA GeForce 8800 - G80 Architecture

NVIDIA's G80 Architecture was a revolutionary architectural update from its predecessors and GPU programming in general. Revisions to the pipeline model and data flow enabled the G80 to reduce the amount of pipeline stages and altered the sequential flow of execution to be more loop oriented [14].

3.2.4 NVIDIA Fermi Architecture

NVIDIA Fermi was the greatest revision and improvement of architectural design for GPUs at the time (2010). The GPUs implementation hosted 3 billion transistors with 512 CUDA cores. The CUDA cores are distributed in batches of 32 where they are contained with 16 *Streaming Multiprocessor* units.

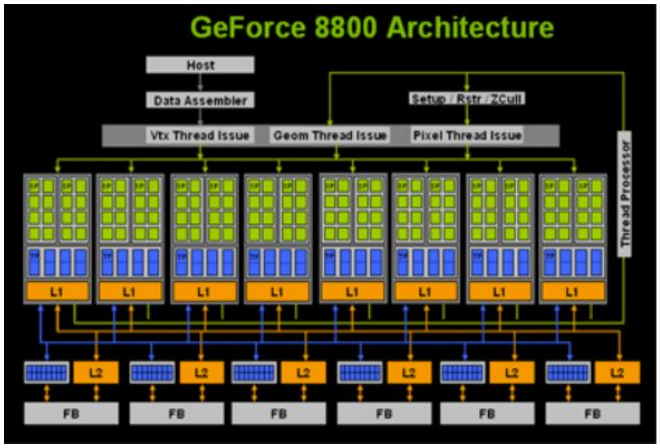


Figure 3.3: Geforce 8800 GPU Architecture

The GPU is built around a 384-bit memory interface, subdivided into six partitions of 64-bit memory, capable of 6GB of GGDDR5 DRAM. The GPU is connected to the CPU via a host interface through the PCI-Express bus.

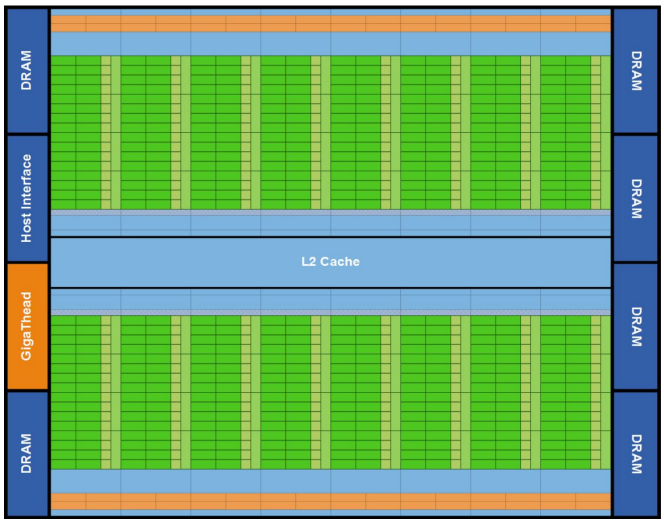


Figure 3.4: Fermi Architectural Diagram

3.2.5 NVIDIA Ampere Architecture

NVIDIA Ampere is NVIDIA's most recent GPU microarchitecture (2020). Ampere devices are frequently used as "data centers" to train complex deep learning models and to accelerate AI training models. The Ampere devices boast up to 128 SM units, the SM units also have

a revised internal architecture to extract further performance. With 40GB of High Bandwidth Memory these devices are the pinnacle of both speed and bandwidth of GPU architecture in recent times.

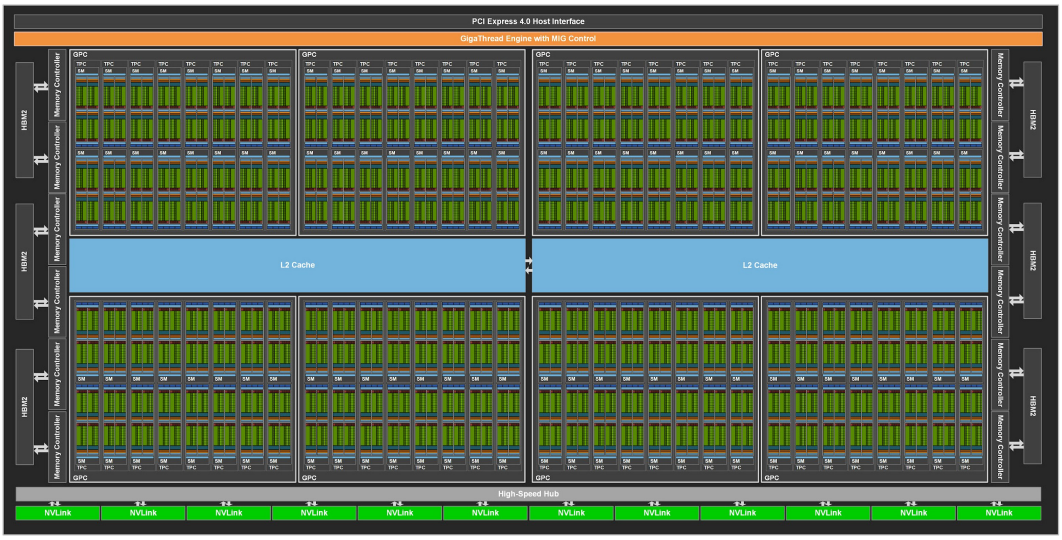


Figure 3.5: NVIDIA Ampere Architectural Diagram

3.3 AMD's Architecture

This section introduces AMD's conceptual perspectives and implementation of GPU architecture.

3.3.1 Streaming Processors (SP), Streaming Processing Units (SPUs) & SIMD Cores

AMD utilize SPs in what they assign the name Arithmetic Logic Units (ALU) within their GPUs. These devices perform mathematical operations. SPs are encased with SPUs with control logic components and registers. Several of these SIMD cores are then assembled with registers, caches and data units to make up the GPU. An example of this is visible in *Figure 3.7* where ten SIMD cores are assembled to construct the RV770.

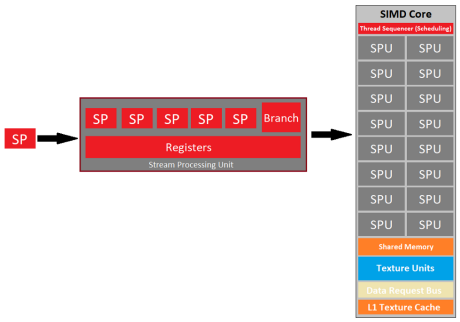


Figure 3.6: AMD SP, SPU and SIMD Core Architecture (TeraScale)

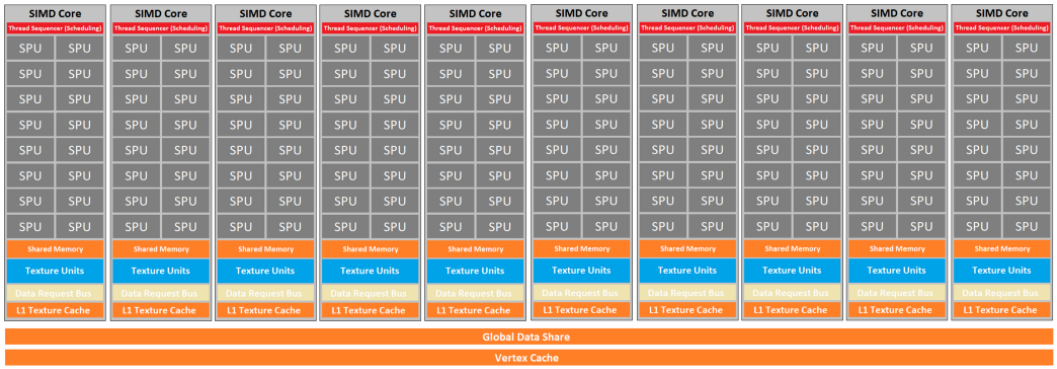


Figure 3.7: RV770 Architectural Diagram containing SPUs

3.3.2 TeraScale

In 2007 AMD revealed their new GPU microarchitecture which would compete with NVIDIAs Tesla microarchitecture. TeraScale utilizes components discussed in the previous section *(See Section 3.3.1)*.

3.3.3 GCN (Graphics Core Next)

In 2012 AMD revised their microarchitecture with their new GCN technologies [15], with key amendments to the organization and assembly of SIMD and compute units. GCN moved away from VLIW4 SIMDs used in previous microarchitecture to a Quad-Core SIMD system. Where SIMDs in each GCN compute unit have combination of private and shared resources for efficiency.

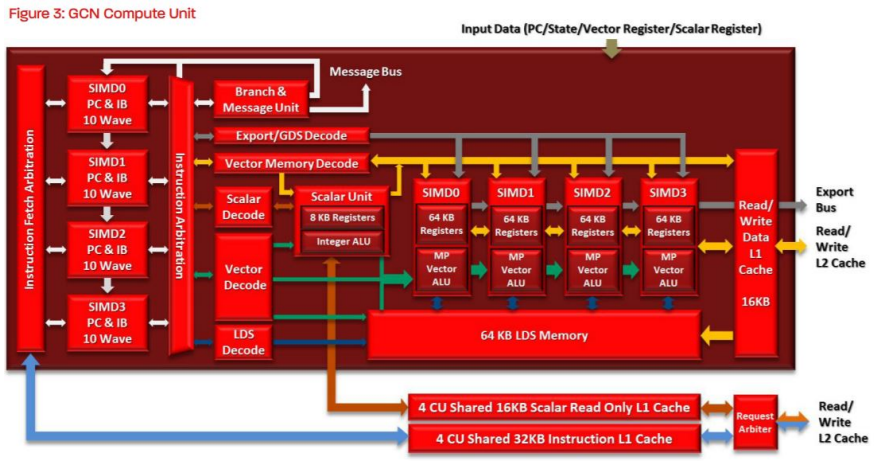


Figure 3.8: AMD GCN Architectural Diagram

3.3.4 AMD RDNA (Radeon DNA)

AMD's RDNA architecture released in 2019 utilizes dual-compute units to perform parallel programming, a new technology not before utilized within its predecessors [16]. Within these dual-compute units are four SIMD (Single Instruction Multiple Data) units that include 32 ALUs which are twice as fast and wide as ALUs exhibited in AMDs previous generation of architecture, designed for mixed-precision and efficiently compute a wide variety of data types to improve machine learning and scientific computing on their GPU devices.

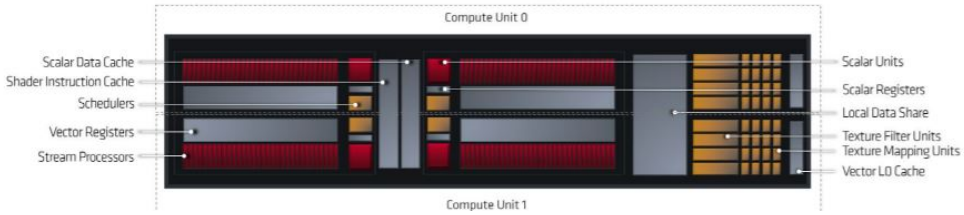


Figure 3.9: RDNA Dual-Compute Unit

3.4 Discussion

It can be observed that over time the software requirements for the GPU have inspired and pushed GPU manufactures to construct architectures that can support them. The early GPU micro-architectures had a clear focus on delivering fast calculation of vector arithmetic in mass parallel to accelerate graphics programs, it has over time evolved to satisfy not only the needs

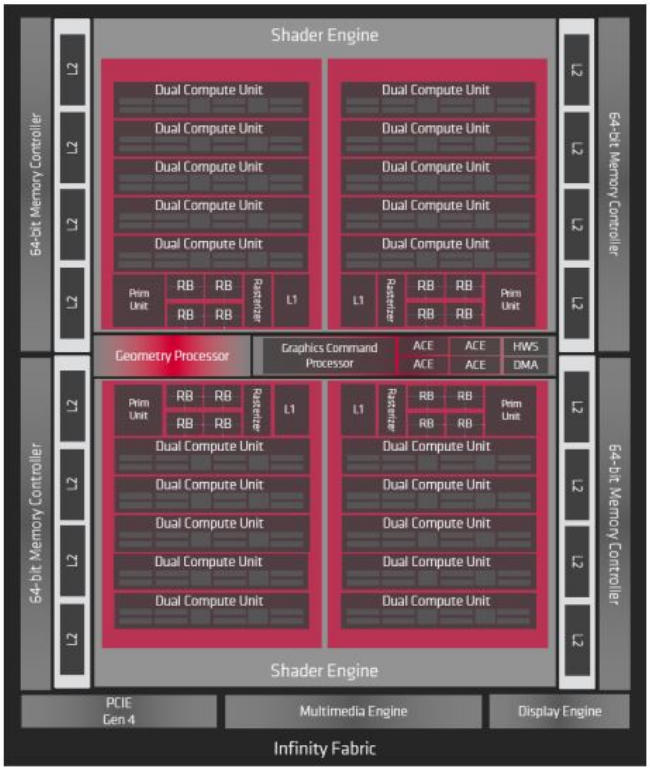


Figure 3.10: RDNA Micro-architecture

of the computer graphics industry but also the machine learning and deep learning industries and this can be seen in the refinements to architecture to facilitate for broader data types and arithmetic operations.

Chapter 4

GPU Software Architecture

Modern GPUs functionality and hardware architecture is developed in conjunction with software architecture and paradigms, where break through in one architecture discipline results in revisions in the other. This hardware/software relationship means that the GPUs software requirements and task implementations are capable of being fully supported and optimized by the device.

This chapter introduces major pipelines used to program GPU, as well as programming languages, and paradigms designed to fully harness the hardware functionality of the GPU.

4.1 Graphical APIs

4.1.1 DirectX

Microsoft's DirectX is an API tailored to multimedia tasks such as film and video game programming written in C++. DirectX currently operates on version DirectX12 enabling the most modern of graphics technologies such as ray tracing and variable rate shading. DirectX is limited to operate on only Microsoft operating systems, but does so very efficiently and effectively. DirectX12 can easily be used in conjunction with high level languages such as C++ and C, due to DirectX being written in C++ it's very intuitive to program using C++ as a base language. Direct3D has its own Graphical Pipeline *(See Section 4.3)* which is programmable by DirectX's accompanying shader language HLSL (High Level Shader Language).

HLSL is a C-like language enabling you to program the entire graphical pipeline, capable of implementing renders, physics simulations and even machine learning based projects.

4.1.2 OpenGl

Originally published by Silicon Graphics and since developed by Khronos Group,"OpenGL is the most widely adopted 2D and 3D graphics API in the industry" [17], operating on multiple platforms OpenGL exposes all functionality of modern day graphical hardware. OpenGL is accompanied by shader language GLSL (OpenGL Shading Language) which is derived from C, enabling the developer to program elements of the graphical pipeline. OpenGL is considered the most simple to learn of the main graphics APIs as elements of resource tracking and memory handling is partially controlled by the API itself, simplifying the role of the programmer, but introducing undesired overhead to created programs.

4.1.3 Vulkan

Vulkan is a cross platform 3D graphics API developed in C by AMD and Khronos Group and DICE released in 2016. Vulkan is an "explicit API" meaning almost everything is the responsibility of the programmer [18], such as state tracking synchronization and memory management. This makes learning Vulkan more difficult however, very rewarding as with correct configurations of programs resources can be handled in the most efficient manner with little to no overhead. Unlike other APIs Vulkan shader code has to be specified in a bytecode format named SPIR-V, although Khronos released its own independent compiler which compiles GLSL (OpenGL Shading Language) code into SPIR-V format making life easier for the programmer. [19]

4.2 General Programming APIs

4.2.1 OpenCL

OpenCL is an API capable of writing programs across heterogeneous platforms, most applicably across GPUs and CPUs. OpenCL is an extremely versatile cross platform tool making it a popular choice to perform parallel programming and to accelerate a wide spectrum of applications such as scientific software and neural network training [11] [20].

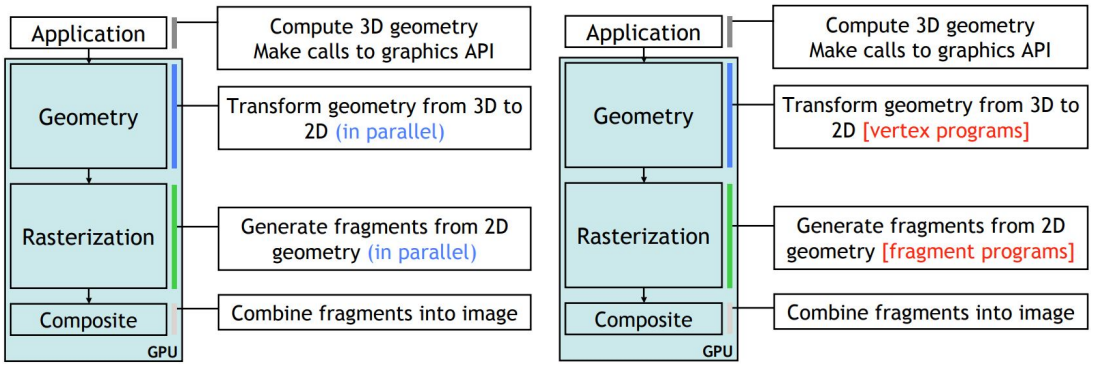
4.2.2 CUDA

CUDA API enables programmers to use CUDA-enabled GPUs and to create General Purpose applications.

4.3 Graphical Pipeline

Commonly known as the Rendering Pipeline, the graphical pipeline is a conceptual model outlining the protocol a graphics program issues to render a 3D scene onto a 2D display. The term pipeline describes the manner in which the output of one phase will be delivered as input to the next. Different GPUs and APIs have slight variations of the pipeline to suit specialist features of their hardware or software requirements, however the general outline of the pipeline remains consistent.

A general rendering pipeline consists of an *Application* phase, where a high level language such as C++ or C will define and prepare vertices, indices and models, and make calls to the graphical API to prepare the pipeline. The next phase is known as the *Geometry* phase, this phase is responsible for masking transformation in manipulating the vertices provided in the previous phase, performing some calculations and manipulations to the data and eventually transforming the 3D geometry into 2D in a parallel fashion. The next stage of the pipeline is generally known as the *Razterization* stage, this stage will take the 2D geometry from the previous phase, and generate fragments from it; this phase is also executed in parallel. The final stage, is known as the *Composite* phase; which is responsible for combining fragments from the previous stage into an output image to be output to a render target such as a display. Basic diagrams illustrating the pipeline are provided from [21] *(See Figure 4.1)*.



A. General Rendering Pipeline B. General Programmable Pipeline

Figure 4.1: Illustrations of a General Graphics Pipeline Detailing its Programmable Elements

Expansions have been made on the pipeline construct over time, introducing intermediate elements, and making more phases of the pipeline programmable; giving the graphics programmer greater control over modern GPU applications.

4.3.1 Direct3D 10 Programmable Pipeline

Microsoft's Direct3D 10 pipeline was a revolutionary change to its predecessors, designed to provide "graphics for real-time gaming applications" [22] the programmer now has large control of the pipeline stages. The pipeline *(See Figure 4.2)* also contained more features such as the output-merger stage which will be discussed shortly, these revisions to the pipeline drastically improved the efficiency of the pipeline.

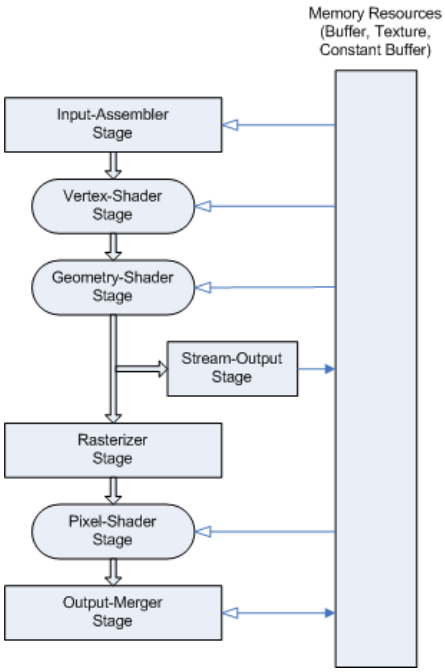


Figure 4.2: Direct3D 10 Programmable Pipeline

- Input-Assembler Stage - The input-assembler stage is responsible for supplying data (triangles, lines and points) to the pipeline.
- Vertex-Shader Stage - The vertex-shader stage processes vertices, typically performing operations such as transformations, skinning, and lighting. A vertex shader always takes a single input vertex and produces a single output vertex.
- Geometry-Shader Stage - The geometry-shader stage processes entire primitives. Its input is a full primitive (which is three vertices for a triangle, two vertices for a line, or a single vertex for a point). In addition, each primitive can also include the vertex data for any edge-adjacent primitives. This could include at most an additional three vertices for a triangle or an additional two vertices for a line. The Geometry Shader also supports limited geometry amplification and de-amplification. Given an input primitive, the Geometry Shader can discard the primitive, or emit one or more new primitives.
- Stream-Output Stage - The stream-output stage is designed for streaming primitive data from the pipeline to memory on its way to the rasterizer. Data can be streamed out and/or passed into the rasterizer. Data streamed out to

memory can be recirculated back into the pipeline as input data or read-back from the CPU.

- Rasterizer Stage - The rasterizer is responsible for clipping primitives, preparing primitives for the pixel shader and determining how to invoke pixel shaders.
- Pixel-Shader Stage - The pixel-shader stage receives interpolated data for a primitive and generates per-pixel data such as color.
- Output-Merger Stage - The output-merger stage is responsible for combining various types of output data (pixel shader values, depth and stencil information) with the contents of the render target and depth/stencil buffers to generate the final pipeline result.

[22]

All stages above described by Microsoft [22] are configurable using the DirectX API, where rectangular boxes in the pipeline *(See Figure 4.2)* contain common shader cores which are programmable by DirectX shader language HLSL.

4.3.2 OpenGL Graphical Pipeline

OpenGL have their own rendering pipeline implemented when rendering objects, which has its own slight adaptations from the general outline of the rendering pipeline.

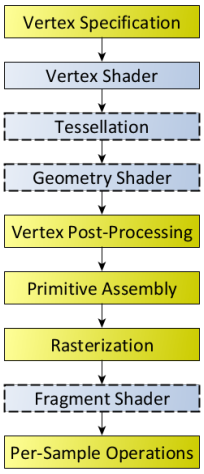


Figure 4.3: OpenGL Pipeline

- Vertex Specification – Vertices are specified externally, vertex definitions can also be paired with index and texture specifications.
- Vertex Processing – Vertices retrieved from the vertex array are processed by a Vertex Shader. Dependant on the primitives chosen to represent the vertices, may involve the tessellation stage and geometry shader.
- Vertex Post-Processing – Performs transformation upon the output from the Vertex Processing phase.
- Primitive Assembly – Primitives are divided into sequences of base primitives.
- Rasterization – Primitives are further broken down into discrete fragments.
- Fragment Shading – Each fragment is processed, controlled by the fragment/pixel shader.
- Per-Sample Processing – A set of tests that control how processed fragments are output to various buffers. Test that belong to this phase include:
  - Blending: Tests that use fragment rgba values (specifically '*a*' (alpha) which is often used to represent opacity) to combine with other fragment color values stored in buffers to produce a blended color result.
  - Depth Testing: Test that calculates the value to be stored in the depth buffer.
  - Stencil Testing: Test that determines fragments that require culling.

4.3.3 Vulkan Graphical Pipeline

Khronos' Vulkan also has its own perspective of rendering pipeline. Descriptions of the pipeline elements will be omitted for conciseness of the document although they are identical or similar to the definitions provided in the aforementioned pipelines. [23]

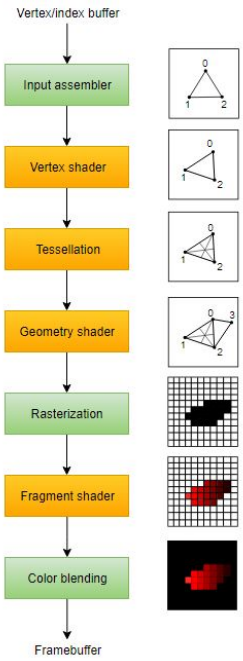


Figure 4.4: Vulcan Pipeline

Chapter 5

Application of the GPU

This chapter explores the various applications of the GPU within the computer science industry and the world itself. Discussion will be held comparing early instances of the relative application to modern implementations to gain perspective on how techniques and technologies concerning the GPU have evolved.

5.1 Rendering

GPUs and rendering have become almost synonymous, as discussed in Chapter 3 the early purpose of GPUs were to perform graphical related calculations to enable shapes, colors and objects to be drawn to a monitor.

5.1.1 Brief Timeline

Ray Casting (1968)

Ray casting is one of the most simplistic rendering techniques stemming from algorithmic ray tracing. It is constructed using the notion that rays are omitted from the eye, one per pixel, and discover the closest object obstructing the ray. Using the material and light properties in the scene, colors are then calculated to generate the shading for the object. [24]

Gouraud Shading (1971)

Gouraud Shading which was named after Henri Gouraud is a shading technique based on interpolation, using vertex normals in combination with a reflection model to produce colors at

each vertex. The color at each pixel value will then be interpolated using the values calculated at each vertex. [25]

Phong Shading (1973)

Phong Shading is an interpolation based shading technique invented by Bui Tuong Phong. The technique performs interpolation using surface normals across rasterized polygons in which it calculated pixel colors from the active reflection model and interpolated normals [26]. Phong [26] also discovered a local illumination model, now known as Phong Reflection. It makes use of three components ambient (base color), diffuse (reflection from rough surfaces) and specular (reflection highlights from shiny surfaces). The amalgamation of these three components make up the Phong Reflection model.

Blinn-Phong Reflection Model (1973)

The Blinn-Phong reflection model is an expansion of the Phong Reflection model by James F. Blinn [27]. By default, the shading is executed on each vertex using Gouraud Shading instead of the more computationally expensive Phong Shading model. The revision to the technique focuses on a weakness in calculation of the specular component, making for more realistic specular highlights for a greater amount of viewing angles.

Ray Tracing (1980)

Ray tracing is a technique where images are generated by tracing the path of light as pixels within an image plane. The objects the ray encounters are simulated accordingly relative to properties of the object. Images produced using this technique are generated at a great level of visual accuracy and realism but calculations are computational expensive. Over time, ray tracing based solutions have been implemented with greater amount of recursion as GPUs have evolved allowing for faster computation. [28]

Radiosity (1984)

Radiosity is a rendering method for surfaces that reflect light diffusely. The technique can be seen as a global illumination model as it considers light from other surfaces reflecting light not just the light source. This method can achieve more visually accurate results for diffuse illumination but with a slight computational cost increase compared to direct illumination methods. [29]

Rendering Equation (1986)

The Rendering Equation was introduced into the computer graphics field by David Immel et al. [30] and James Kaijiya [31], the equation provides description for the amount of light emitted from a particular point along a view direction - giving a bidirectional reflection distribution function (BDRF) and providing a function of incoming light. [31] This is the general equation that various rending methods attempt to solve. Although due to the general nature of the equation, large amounts of light reflection phenomena are not captured.

Transform, Clipping and Lighting (1993)

Transform, clipping and lighting known as TCL is a term given to several computer graphics techniques. Transformation is the production of a 2D view of a 3D scene, clipping is the omittance of scene elements not to be included in the final image, and lighting is the calculation of color on surfaces within the scene.

TCL was introduced in arcade game boards in 1993 [32] and by 1994 was introduced into Sega Saturn's SCU-DSP. In 1999 NVIDIA's GeForce 256 introduced hardware support for TCL in the consumer graphics market. A significant moment in history was Direct3D switching vertex transformation to a CPU operation to a GPU operation as until then only Open-GL based 3D first person shooters implemented TCL.

Directional Lighting (1993)

The technique of directional lighting [32] illuminates all objects equally from a light source in a particular direction, with no distance fall-off (illumination intensity based on distance from light source) of light.

Z-culling (1993)

Z-Culling is a technique first introduced in 1993 [32] which eliminates processing of pixels based on its depth within a scene. This technique provides an increase in performance as elements which would not be visible do not need to have their potentially expensive lighting calculations performed. This technique is often used to optimize programs within computer graphics.

Ambient Occlusion (1994)

Ambient occlusion is a technique for shading and rendering used to calculate how much a point in a scene is exposed to ambient lighting. This method is a global illumination based technique unlike other methods using ambient components such as Phong Shading. This method attempts to create a more realistic visual appearance of surface based on a more accurate ambient color base with respect to the objects surroundings. [33]

Photon Mapping (1995)

Photon mapping is a global illumination method developed in 1995 by Henrik Wann Jensen and Niels Jørgen Christensen [34]. It is a two-pass method which emits rays from the light source and rays from the camera which are traced interdependently. This is performed until the termination criteria is satisfied, the rays are then connected and used in the next step where a radiance value is generated. This is a useful but computationally expensive technique, and is capable of simulating advanced light phenomena such as refraction and caustics.

Present

As seen in the above timeline the rendering foundations were set down in the early stages of computer graphics. When 2D graphics accelerators were introduced further abstractions of these techniques were developed as more complex calculations were able to be performed at reasonable speeds. 3D based rendering and modelling solutions began to arise when 3D graphics accelerators saw greater support. More recently in computer graphics, there has been less invention, however the pre-existing methods have been expanded and implemented at a much larger scale. An example of this is ray tracing, which has been expanded to support much larger levels of recursion producing photo-realistic output.

5.1.2 Examples

This section demonstrates the visual quality of computer graphics both past and present, to provide basis for discussion on how computer graphics has evolved throughout time.

5.1.2.1 NVIDIA Grass Demo (GeForce 256) (2000)

Figure 5.1: NVIDIA Grass Demo (GeForce 256)

Upon release of NVIDIAs GeForce 256, NVIDIA released a software demo to demonstrate the computational power and rendering possibilities capable on their devices. The demo generates up to 10,000 blades of grass dynamically. The ability to deliver this in a real-time application was advanced for graphics applications at the time.

5.1.2.2 Tomb Raider (2008) vs. Shadow of the Tomb Raider: The Nightmare (2013)

It is evident from the two renders of Tomb Raider that drastic improvements have been made regarding lighting and modelling. Geometric primitives used to model Lara Croft in Tomb Raider 2008 *(See Figure 5.2A.)* are extremely visible with the non-smooth polygonal model. When compared to the more recent render of Lara Croft *(See Figure 5.2B.)* it can be seen that levels of realism with the model and lighting are achieved.

A. Tomb Raider (2008)

B. Shadow of the Tomb Raider: The Nightmare (2013)

Figure 5.2: Geometric primitives are extremely prominent due to the limitation of vertices able to be used per model in Figure A. Figure B showcases a near photo-realistic render where geometric primitives used are unnoticeable

5.1.2.3 'Reflections' Unreal Engine 4 Ray Tracing

Figure 5.3: 'Reflections' Unreal Engine 4 Ray Tracing

The visual accuracy achieved by this render of Storm-troopers in Unreal Engine 4 showcases the levels of photo-realism achievable using modern GPU programming techniques on the most powerful hardware. Extreme levels of recursion of ray tracing were utilized to achieve the level of reflection present in the image.

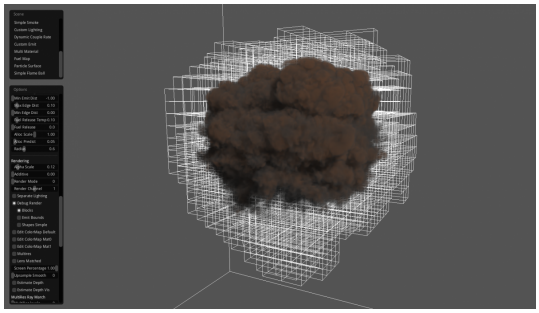
5.1.3 Discussion

The improvement in visual quality throughout the course of GPU graphics development has been drastic. Early renders would use small amounts of geometric primitives to model objects due to the limitations of the hardware. Modern renders use extremely large amounts of primitives even to render simple objects, where extremely powerful recursive lighting algorithms are performed to simulate photo-realistic lighting phenomena. The quality of visual renders continues to progress towards true photo-realism.

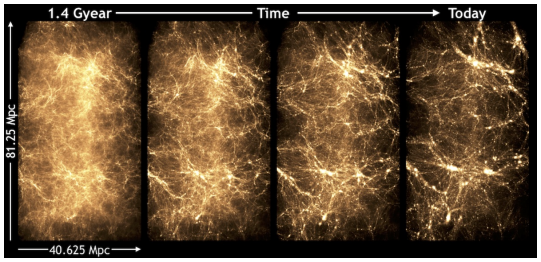
5.2 General Purpose

5.2.1 Simulation

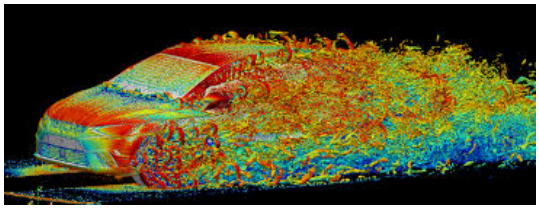
GPUs have been used extensively for simulations, in Computational Fluid Dynamics, Particle Analysis [35], Aerospace [36] and more. Elements of calculations which can be executed in parallel are ideal for implementation on GPUs. These parallel computations can be performed in shaders, known as compute shaders, which deal with computations rather than graphics. Or using a GPU programming framework such as CUDA, OpenCL or others. The results of simulations can then be transformed to form vertices which can then be passed to a programmed rendering pipeline to generate a visual output of the simulation. Recent works of simulations within the field of GPUs and computer graphics is NVIDIA's flow, which offers combustible fluids, fire, smoke and liquid simulations.



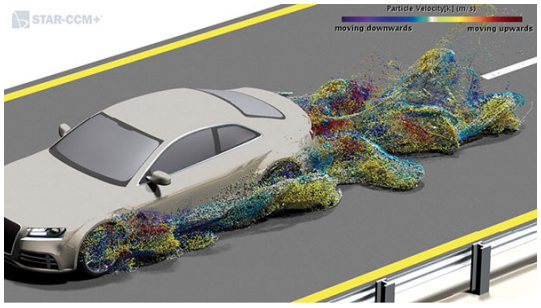
A. NVIDIA Flow Framework



B. GPU-Accelerated Cosmological Analysis on the Titan Supercomputer (NVIDIA)



C. Computational Fluid Dynamics



D. Lagrangian Mass Particle Simulation

Figure 5.4: Example of various GPU based simulations, with NVIDIA Flow Fluid Dynamics *(Figure A)*, GPU-Accelerated Cosmological Analysis on the Titan Supercomputer by NVIDIA *(Figure B)*, Computational Fluid Dynamics *(Figure C)* and Lagrangian Mass Particle Simulation *(Figure D)*

5.2.2 Automotive

NVIDIA marked the first integration of the GPU within the automotive industry with their collaboration with AUDI [37]. In 2010 AUDI elected to install NVIDIA GPUs within all of their navigation and entertainment systems within the vehicles they distributed worldwide. In more recent times, Tesla embed NVIDIA Pascal GPUs to power their interactive display and navigation system [38].

5.2.3 Machine Learning & Deep Learning & Artificial Intelligence

GPUs have seen great use in the fields of Machine Learning and Deep Learning. Training with extremely large neural networks can be computed in parallel to significantly reduce training time. Utilizing the large bandwidth of data available on GPUs Machine Learning processes are computed much faster and more efficiently.

5.2.4 Film

GPUs have been used to enable animators and graphical artists to produce elements of films or even whole films using Computer Generated Imagery (CGI). Large scale battles, deep galaxies, oceans [39], mythical creatures and detailed fur are among imagery created by this technology.

5.2.5 Finance

In very recent times arrays of powerful GPUs have been mass-assembled into stations known as crypto-mining rigs [40]. The GPUs are used to calculate hash functions to be added to the block-chain. These calculations can take a long time to calculate but when executed across an array of GPUs many of these calculations can be calculated simultaneously. As well as in the crypto market, GPUs are used to process real-time stock market behaviours as well as providing detailed visual analysis on the applications. GPUs are also used to accelerate general finance applications.

Chapter 6

Conclusion

From the life cycle of the GPU, it can be observed that its application evolved since the days of being a device dedicated towards the acceleration of basic graphics operations. However, this evolution has not had a clear path. The emergence of new technologies and concepts such as machine learning and deep learning have forced the hand of GPU hardware manufactures to adapt the hardware to be flexible enough to facilitate for a wide range of data types and operations, whilst still being able to deliver acceleration for graphics applications. Both aspects of GPU technology are continuing to develop as a result of the recent requirements for modern graphics to be delivered at interactive speeds with photo-realistic visual quality. As well as this, machine learning models are becoming more complex as AI continues to develop, meaning GPUs have to continue to adapt to this advancement. It can be concluded that the evolution of the GPU has been a combined effort of hardware and software technological advancements, a relationship that will likely continue in the future.

However, the emergence of processing units such as the IPU (Intelligence Processing Unit) developed by Graphcore - a processing unit optimized for simulations, machine learning, deep learning and AI [41] [42] - could rival the GPU in those respective industries. How GPU manufacturers respond to the emergence of these devices could alter the course of the GPUs development. If GPU manufactures decide to concede dominance in the machine learning, deep learning and AI market the GPU could see its focus return to being a more specific device, dedicated to delivering high performance graphics.

Either way, the future for the GPU remains interesting, and will continue to be a driving force for many key disciplines within computer science.

Bibliography

[1] T. M. S. Kent C. Redmond, *Project Whirlwind : The History of a Pioneer Computer*. Digital Press, 1980.

[2] RCA, *RCA 1800 Microprocessors, Design Ideas Book*. RCA, 1975.

[3] I. B. Nick Montfort, *Racing the Beam: The Atari Video Computer System*. MIT Press, 2009.

[4] D. C. Lili Zhao, "Research and design of crt controller based on CPLD," 2012.

[5] E. D. Larry Press, "IBM PC," 2003.

[6] "Atari Inc.", "Data processing system with programmable graphics generator," 1979, "U.S Patents".

[7] Intel Corporation, *iSBX 275 Video Graphics Controller Multimodule Board Reference Manual*, 1982.

[8] CM Wyant, CR Cullinan, TR Frattesi, "Computing Performance Benchmarks among CPU, GPU, and FPGA," 2012.

[9] L Kelty, P Beckett, L Zalcman, "Desktop simulation," 1999.

[10] NVIDIA Corporation. (2006) CUDA - Presentation. [Online]. Available: http://www.nvidia.com/object/cuda_home_new

[11] A Munshi, B Gaster, TG Mattson, D Ginsburg, *OpenCL Programming Guide*. Addison-Wesley, 2011.

[12] NVIDIA Corporation, "NVIDIA's Next Generation CUDATM Compute Architecture: Fermi," 2009.

[13] ——, "GPU Fundamentals," 2016, Presentation.

[14] ——, "Technical Brief - NVIDIA GeForce 8800 GPU Architecture Overview," 2006.

[15] M. Mantor, "Amd radeon™ hd 7970 with graphics core next (gcn) architecture," in *2012 IEEE Hot Chips 24 Symposium (HCS)*. IEEE, 2012, pp. 1–35.

[16] N. Otterness and J. H. Anderson, "Amd gpus as an alternative to nvidia for supporting real-time workloads," in *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[17] Khronos Group. OpenGL. [Online]. Available: https://www.khronos.org/opengl/

[18] A Munshi, B Gaster, TG Mattson, D Ginsburg, *Vulkan Programming Guide: The Official Guide to Learning Vulkan*. Addison-Wesley, 2011.

[19] Khronos Group. Graphical Pipeline Basics: Shader Modules. [Online]. Available: https://vulkan-tutorial.com/Drawing_a_triangle/Graphics_pipeline_basics/Shader_modules

[20] ——. OpenCL. [Online]. Available: https://www.khronos.org/opencl/

[21] John Owens, UC Davis, "GPU Architecture Overview," 2007, presentation.

[22] Microsoft Corporation. (2018) Pipeline Stages (Direct3D 10. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/direct3d10/d3d10-graphics-programming-guide-pipeline-stages/

[23] Khronos Group. Graphical Pipeline Basics: Introduction. [Online]. Available: https://vulkan-tutorial.com/Drawing_a_triangle/Graphics_pipeline_basics/Introduction

[24] A. Appel, "Some techniques for shading machine renderings of solids," 1968.

[25] H. Gouraud, "Continuous Shading of Curved Surfaces," 1971.

[26] B. Tuong-Phong, "Illumination of Computer-Generated Images," 1973.

[27] J. F. Blinn, "Models of light reflection for computer synthesized pictures," 1973.

[28] J. T. Whitted, "An improved illumination model for shaded display," 1979.

[29] Cindy M. Goral et al., "Modeling the interaction of light between diffuse surfaces," 1984.

[30] D. I. et. al., "The rendering equation. Computer Graphics," 1986.

[31] J. Kajiya, "The rendering equation. Computer Graphics," 1986.

[32] Namco. (1993) System 16 - Namco Magic Edge Hornet Simulator Hardware (Namco). [Online]. Available: www.system16.com

[33] G. Miller, "Efficient algorithms for local and global accessibility shading," 1994.

[34] N. J. C. Henrik Wann Jensen, "Photon maps in bidirectional monte carlo ray tracing of complex objects," 1995.

[35] T. Amada, M. Imura, Y. Yasumuro, Y. Manabe, and K. Chihara, "Particle-based fluid simulation on gpu," in *ACM workshop on general-purpose computing on graphics processors*, vol. 41. Citeseer, 2004, p. 42.

[36] G. Sudhakaran, T. C. Babu, and V. Ashok, "A gpu computing platform (saga) and a cfd code on gpu for aerospace applications," in *Proceedings of the ATIP/A* CRC Workshop on Accelerator Technologies for High-Performance Computing: Does Asia Lead the Way?*, 2012, pp. 1–5.

[37] N. Automotive, "Autonomous cars: no longer just science fictionby: Esther francis."

[38] K. Liu and R. Mulky, "Enabling autonomous navigation for affordable scooters," *Sensors*, vol. 18, no. 6, p. 1829, 2018.

[39] L. S. Jensen and R. Golias, "Deep-water animation and rendering," in *Game Developer's Conference (Gamasutra)*, 2001.

[40] D. Draghicescu, A. Caranica, A. Vulpe, and O. Fratu, "Crypto-mining application fingerprinting method," in *2018 International Conference on Communications (COMM)*. IEEE, 2018, pp. 543–546.

[41] I. Kacher, M. Portaz, H. Randrianarivo, and S. Peyronnet, "Graphcore c2 card performance for image-based deep learning application: A report," *arXiv preprint arXiv:2002.11670*, 2020.

[42] L. R. M. Mohan, A. Marshall, S. Maddrell-Mander, D. O'Hanlon, K. Petridis, J. Rademacker, V. Rege, and A. Titterton, "Studying the potential of graphcore ipus for applications in particle physics," *arXiv preprint arXiv:2008.09210*, 2020.