University of Nevada
Reno

Evolution of the Graphical Processing Unit

A professional paper submitted in partial fulfillment
of the requirements for the degree of
Master of Science
with a major in
Computer Science

by

Thomas Scott Crow

Dr. Frederick C. Harris, Jr., Advisor

December 2004

Dedication

To my wife Windee, thank you for all of your patience, intelligence and love.

Acknowledgements

I would like to thank my advisor Dr. Harris for his patience and the help he has provided me. The field of Computer Science needs more individuals like Dr. Harris.

I would like to thank Dr. Mensing for unknowingly giving me an excellent model of what a Man can be and for his confidence in my work.

I am very grateful to Dr. Egbert and Dr. Mensing for agreeing to be committee members and for their valuable time.

Thank you jeffs.

Abstract

In this paper we discuss some major contributions to the field of computer graphics that have led to the implementation of the modern graphical processing unit. We also compare the performance of matrix-matrix multiplication on the GPU to the same computation on the CPU. Although the CPU performs better in this comparison, modern GPUs have a lot of potential since their rate of growth far exceeds that of the CPU. The history of the rate of growth of the GPU shows that the transistor count doubles every 6 months where that of the CPU is only every 18 months. There is currently much research going on regarding general purpose computing on GPUs and although there has been moderate success, there are several issues that keep the commodity GPU from expanding out from pure graphics computing with limited cache bandwidth being one.

Table of Contents

List of Figures

Chapter 1 Introduction

The idea that we can get from a vacuum tube to a graphics processor, whose main purpose it is to perform graphical computations for incredible computer game graphics effects and is capable of more raw computation than a computer's central processor would be the stuff of science fiction novels even only just 20 years ago. But yet, here we are at that point and the performance gap continues to grow with no end in sight. In 1999, the 3D graphics pipeline was first fully implemented in the hardware of a GPU. Before that, some parts of the pipeline if not all were implemented via software that was then computed by the CPU. What this all meant was that computer graphics was primitive at best and very exotic hardware and software were required to do anything other than very low-end graphics. This all changed with the advent of the low-cost personal computer and the surge in computer games that drove a demand for better computer games and software which in turn demanded better hardware. Nowadays, you can buy a $399 computer graphics card for playing virtually real-time computer games that is more than 3 times as fast as a high-end personal computer costing nearly 10 times as much.

Even though GPUs today have more computational horsepower, they are fine-tuned for the type of computation that is required for computer graphics which is highly parallel, numerically demanding with little to no data reuse. Although many types of computations demonstrate qualities of high parallelism and numerical intensiveness, they also require significant data reuse. CPUs have large caches with high bandwidths

that facilitate the reuse of data that makes them very suitable for general purpose computation whereas a GPU has much smaller caches with lower bandwidths since they are geared for the type of computations that are required for graphics.

The rest of this paper is outlined as follows: in Chapter 2 we talk about some of the more important milestones that have a direct bearing on the field of computer graphics and specifically the GPU. Chapter 3 is a discussion of the modern GPU and how the 3D graphics pipeline, which is the basic principle behind 3D graphics, has moved from the CPU to GPU and thus now allows for a user to actually write custom programs that run directly on the GPU. Chapter 4 talks about general purpose, non-graphics computing performed on the GPU. For this paper a test case was implemented that performed matrix-matrix multiplication on both the CPU and the GPU to see which would perform better and the results are discussed in 4.2. Chapter 5 completes this paper with a brief discussion of the future of the GPU.

Chapter 2 Computer Graphics Milestones

The term "Computer Graphics" was devised by a Boeing graphics designer in 1960 to describe the work he was doing for Boeing of designing a more efficient cockpit space. His contributions are a computer generated orthographic view of the human body and some of the first 3D computer animation [25]. There are many additional technologies and advancements that have helped to propel the field of computer graphics to its current state. In this section we will recognize many milestones from the first digital computer to the beginnings of the personal computer (PC). The ramifications of these events far exceed their contributions to the field of computer graphics. Most people would agree that the first digital computer is where the first computer graphics systems started and that is where we will begin the next section [41].

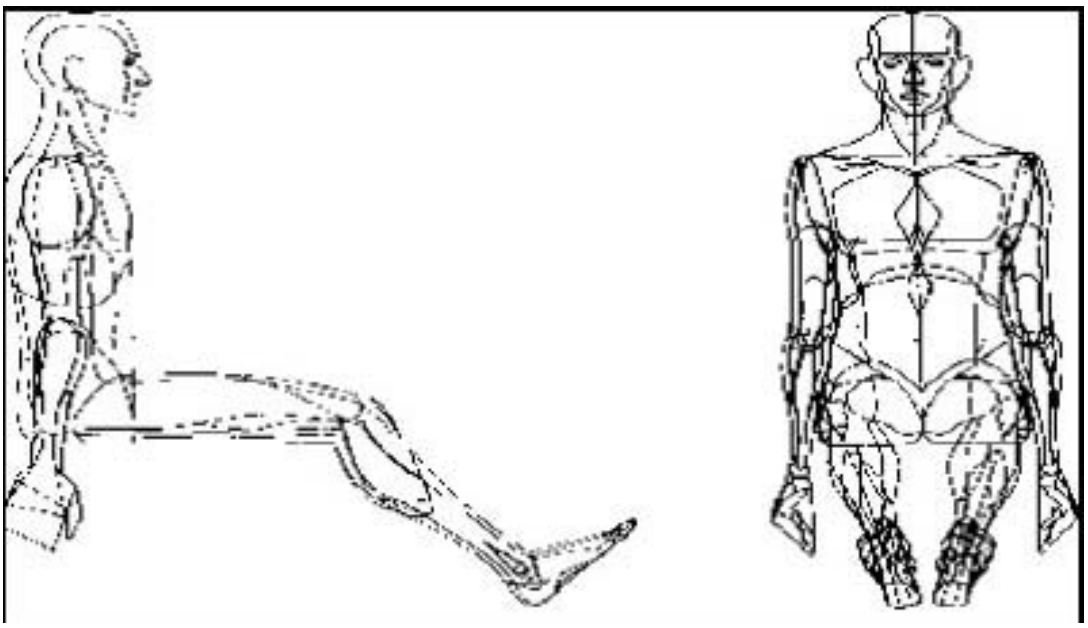

Figure 2.1: William Fetter's Orthographic "First Man" form [25]

2.1 The Whirlwind and Sage Projects

By almost any measure—scale, expense, technical complexity, or influence on future developments—the single most important computer project of the postwar decade was MIT's Whirlwind and its offspring, the SAGE computerized air defense system [21]. The design and implementation of the Whirlwind computer project began in 1944 by Jay Forrester and Ken Olsen of MIT. The project came out of the Navy's Airplane Stability and Control Analyzer project (ASCA). The idea was to come up with a design for a programmable flight simulation computer which could be programmed to provide training for Navy pilots on any aircraft without having to customize a new computer for every aircraft type. Although Whirlwind was not the first digital computer, it was the first computer built specifically for interactive, real-time control which displayed real-time text and graphics on a video terminal [64]. Because the memory at the time was not fast enough to allow Whirlwind to be a real-time control computer, a new type of memory was created by Jay Forrester called core memory. This was the technique of using a matrix of wires with donut shaped ferrite ceramic magnets (called a core) at each junction to produce random access memory shown in Figure 2.2.

Although the focus of the Whirlwind project shown in Figure 2.3 started out as a general-purpose flight simulator, it soon morphed into a design for a general-purpose, real-time digital computer with the ability to do more than flight simulation calculations. The Air Force saw a potential for this real-time, general purpose computer, where there was none before and took over funding of the Whirlwind project to support

a project of their own termed SAGE (Semi-Automatic Ground Environment) [37].

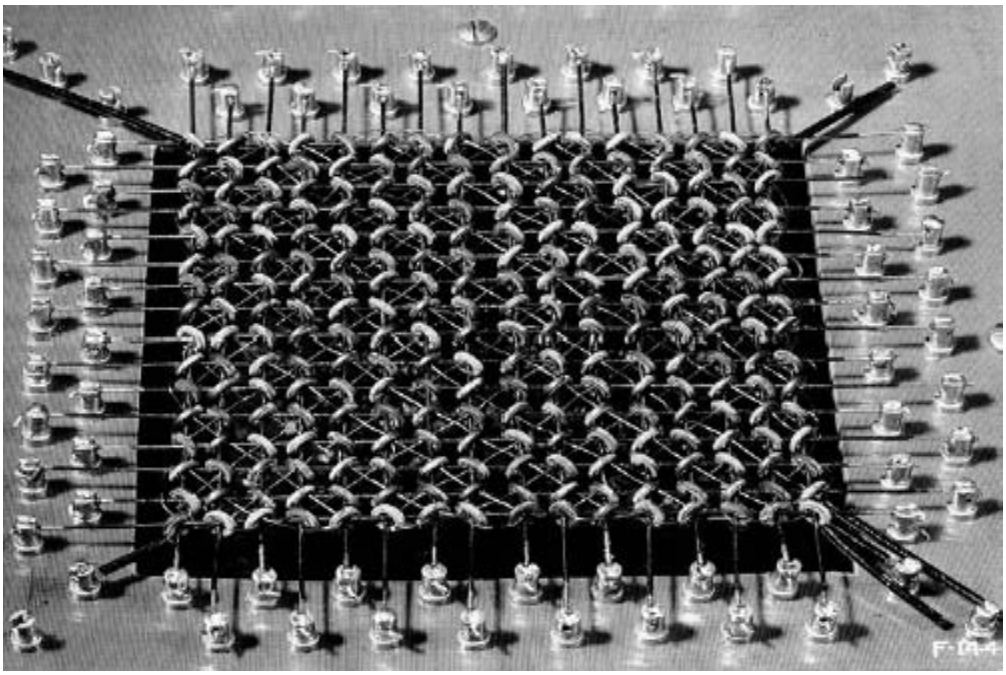

Figure 2.2: Core Memory layout from [15]

Figure 2.3: Whirlwind computer project from [83]

By 1958 the Air Force project SAGE had its first operational directional center using the Whirlwind computer as the control center [52, 78]. The SAGE project shown in Figure 2.4 was designed to use radar information to track and identify aircraft in flight by converting it into computer generated pictures. The SAGE project continued the advancements of the Whirlwind project and the combination of Whirlwind and SAGE presented several advancements that helped create the field of computer graphics. Core memory allowed the replacement of slower, physically larger and smaller memory sizes that relied on vacuum Tubes and thus helped propel real-time computing. Although the CRT (cathode ray tube) was being used as a display for televisions, oscilloscopes and some other computer projects, it was the Whirlwind/SAGE projects that showed the CRT to be a feasible option for display and interactive computing. The Whirlwind/SAGE projects also introduced the light pen shown in Figure 2.5 as an input device to allow the user to mark spots on the CRT for the computer to be able to store and use the data for computations [65].

2.2 MIT's TX-0 and TX-2

The Whirlwind computer was built using 5,000 vacuum tubes. The SAGE computer system using the newer Whirlwind II still used vacuum tubes, but with the advent of the transistor now also incorporated over 13,000 transistors [43, 47, 64 and 68]. As the transistor replaced the vacuum tubes in computers, it allowed for greatly reduced temperature and space requirements. The first real-time, programmable, general purpose computer made solely from transistors was the MIT built TX-0 (Transistorized

Figure 2.4: SAGE Project from [69]

Figure 2.5: SAGE with the Light Pen from [70]

Experimental Computer Zero) shown in Figure 2.6 [21]. The TX-0 was basically a transistorized Whirlwind. The Whirlwind computer was so large that it filled an entire floor of a large building, but the TX-0 fit in the relatively smaller area of a room and was somewhat faster [21, 29]. The TX-0 was the fastest computer of its era and was the predecessor of the minicomputer and the direct ancestor of the DEC PDP-1. The Lincoln laboratory was paid by the Air Force to build the TX-0 to determine if it was feasible to build a major computing system based on only transistors instead of vacuum tubes. It was also used to test the viability of larger, complex core memory first implemented in Whirlwind [14, 21]. The TX-2 project shown in Figure 2.7 was begun shortly after the successful completion of TX-0 in 1956. It was much larger than the TX-0 and was built using 22,000 transistors and was key to the evolution of interactive computer graphics [14]. There are many contributions that the TX-0/TX-2 projects provided for the field of computer graphics. Besides the obvious contribution of showing a much smaller, more powerful transistorized computer is possible; the project made extensive use of the CRT and light pen as an interactive graphics workstation and pushed the next level of computer graphics evolution. The TX-0/TX-2 would be used by Ivan Sutherland to create the famed interactive graphics program Sketchpad [50]. The project allowed the co-creator, Ken Olsen, to take the knowledge he gained on the project and start a company that would go on to become one of the most influential computer companies of the 1970s and 1980s, Digital Equipment Corporation (DEC).

Figure 2.6: M.I.T. TX-0 Computer from [79]

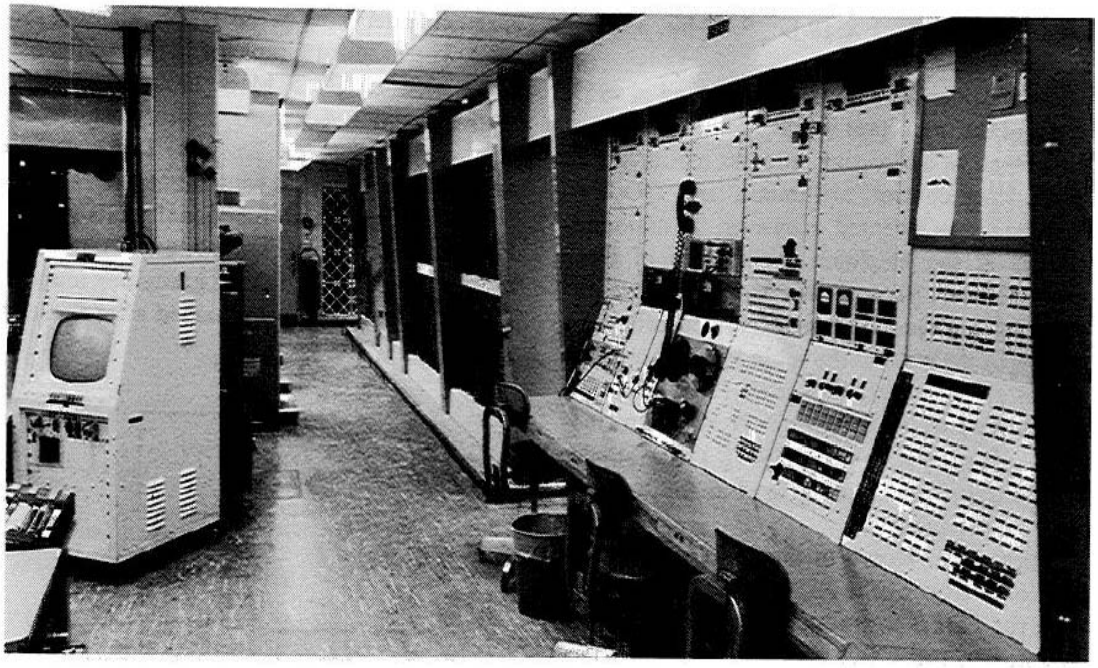

Figure 2.7: M.I.T. TX-2 Computer from [80]

2.3 Sutherland's Sketchpad

Not only did the TX-2 have a 9 inch CRT display, but it also had a number of other input and output devices such as the light pen, a magnetic tape storage device, an on-line typewriter, a bank of control switches, paper tape for program input and the first Xerox printer [75, 77]. All of these new man-machine interfaces made for an environment that was ripe for the right person to take advantage of them and that person was Ivan Sutherland at MIT.

In his 1963 Ph.D. thesis [77], using the powerful TX-2, Ivan Sutherland created Sketchpad shown in Figure 2.8, which as time has revealed was the precursor of the direct manipulation computer graphic interface of today [75]. For his work beginning with Sketchpad, he is widely recognized as the grandfather of interactive computer graphics [75]. Sketchpad was a program written for the TX-2 that allowed a user to draw and manage points, line segments, and arcs on a CRT monitor using a hand-held light pen. These drawings were very precise and offered a scale of 2000:1 allowing for a relatively large drawing area [2].

These drawings were not merely pictures, but were more importantly computer data that was presented and manipulated by the user graphically. The user had the ability to create object relationships using the various primitives that were allowed and could build up complex drawings by combining different elements and shapes. Sketchpad was important for many reasons, but most importantly it freed the user from having to program the computer with Instructions to perform. Instead, it allowed the

Figure 2.8: Sutherland's Sketchpad and the TX-2 from [74]

user to interact with the computer via the light pen and the CRT monitor thus setting the ground work for more advanced man-machine interaction. Sketchpad is the ancestor of computer aided design (CAD) and the modern graphical user interface (GUI) [50, 67].

2.4 Digital Equipment Corporation and the Minicomputer

Shortly after starting on the design of the TX-2 in 1957, Ken Olsen, one of the original designers and builders of the TX-0 and the TX-2 computers left MIT and became the founder of the Digital Equipment Corporation (DEC). In Ken Olsen's own words, "*I was building the hardware, somebody else was designing the logic and they couldn't settle down. So after a year or two of that I got impatient and left"* [1]. That would prove to be the best move of his life not to mention a necessary step in the evolution of the computer graphics industry. Shortly after in 1960, the DEC PDP-1 (programmed data processor) shown in Figure 2.9 became the commercial manifestation of the Air Force funded, MIT

designed and built TX-0/TX-2 [14, 19]. Although only 49 PDP-1s were ever sold, DEC went on to build some of the most influential computers of their time with the first minicomputer the PDP-8 and also the 16-bit word PDP-11 being their most successful and important computer with over 600,000 units and over 20 years of production of the PDP-11 alone [17, 71]. Not only was the PDP-11 the impetus behind several successful CPU architectures such as the VAX supermini and the Motorola 68000 microprocessor family, it is also important for its general computing nature since it had over 23 different operating systems written for it and 10 different programming languages. The DEC VAX would become the workhorse of the CAD industry in the 1980s [9, 22].

Figure 2.9: DEC PDP-1 Minicomputer from [60]

2.5 High-End CAD Graphics Systems

As mentioned earlier in this chapter, computer graphics and most certainly Computer Aided Design (CAD) can be traced back to Sutherland's work with Sketchpad. The idea of having the computer do all of the graphical calculations, being able to manipulate an existing drawing, not having to manually draw, correct and redraw a drawing and all dependent drawings appealed to industries that relied on graphical representations like automotive and aerospace.

General Motors took the idea of CAD and CAM (Computer Aided Manufacturing) and teamed up with IBM to begin working on one of the first CAD/CAM systems, which was the DAC-1 (Design Augmented by Computer) shown in Figure 2.10 [39, 49]. The DAC-1 system expanded the Sketchpad idea by creating, rotating and viewing 3D models, but the system still utilized a time-shared model of computing [22].

In 1967, the IDIIOM (Information Displays, Inc., Input-Output Machine) became the first stand-alone CAD workstation [10]. Building on the demand and success of previous CAD workstations, IDI, a company that manufactured primarily custom CRT displays, began looking at combining their displays with a small third-party, integrated circuit computer to produce a relatively inexpensive stand-alone CAD workstation. The IDIIOM helped move the CAD and graphics industry forward by taking the computing-expensive, time-sharing mainframe component out of the CAD environment by implementing their system with a much less expensive minicomputer instead of a

mainframe. They also designed and programmed their own CAD software called IDADS (Information Displays Automatic Drafting System), which greatly reduced current drafting costs and turn around time. The benefit to providing the user-friendly CAD software application with IDIIOM was that the user was no longer required to code their own software to utilize the machine.

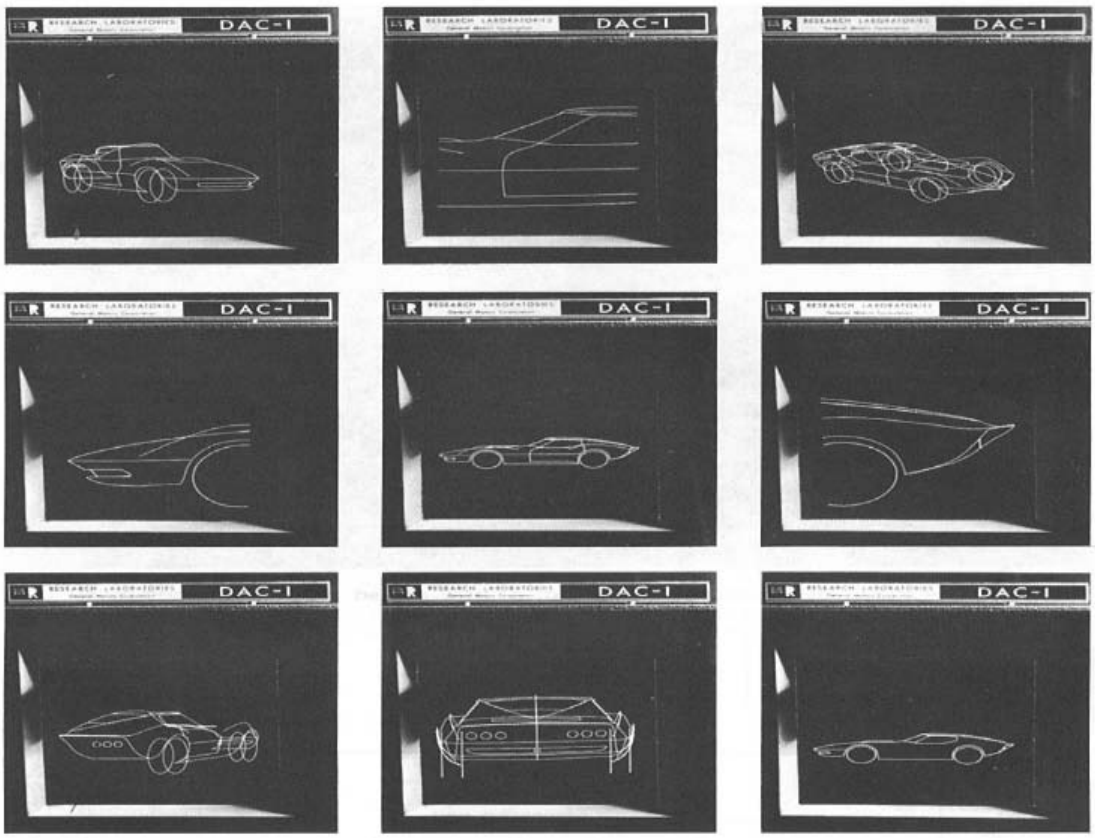

Figure 2.10: General Motor's DAC-1 Computer system from [13]

2.6 The PC Revolution

The most important part of the computer is the central processing unit (CPU). Generically, a CPU has circuit logic for a control unit that takes instructions from memory and then decodes and executes them and also circuitry for arithmetic and

logical operations mainly referred to as an arithmetic logic unit (ALU). Early computers used transistors and/or vacuum tubes for this logic. With the advent of integrated circuits (IC) the microprocessor was able to take all of this circuitry and put it onto a single silicon IC, a single chip. This allowed for much more complex logic to be put into small chips thus reducing size, cost and temperature. With the creation of the first microprocessor in 1971, the Intel 4004 [36], development of computers for personal use became appealing and affordable.

It is hard to pinpoint when the first PC came into existence as the idea is highly ambiguous at best. Thus to simplify, I define PC as being digital, including a microprocessor, being user-programmable, commercially manufactured, small enough to allow it to be moved by the average person, inexpensive enough to be affordable by the average professional and simple enough to be used without special training. The MITS (Micro Instrumentation Telemetry Systems) Altair 8800 shown in Figure 2.11 featured on the cover of Popular Electronics magazine in 1975 is the computer that fits the criterion [3]. It doesn't quite resemble what we think of today as a PC since it didn't have a CRT or a keyboard. The user programmed the computer by flipping toggle switches and read the output from a panel of neon bulbs. With the announcement of the Altair 8800 many other companies starting creating PCs most notably Apple Computer Inc [45].

PCs were initially only being purchased by consumers in small numbers, but in 1981 IBM came out with the IBM PC and instantly legitimized the PC market. With

Figure 2.11: Altair 8800 Personal Computer from [4]

IBM's entry into the PC market, corporations finally started purchasing PCs for business use. This mass adoption by corporations also had the effect of creating standards for the PC industry which further propelled the market. With the advent of the PC and industry standards for hardware as well as software, the computer graphics industry was poised to take great strides in development.

Chapter 3 The Modern GPU

The early 1980s have generally been credited with being the roots of the modern era of Computer Graphics. In 1981 IBM created the first video cards available for the PC. The monochrome display adapter was monochrome text only and could not address a single pixel, but had to change the screen in a 9 x 14 pixel area. Although these cards were very limited and only supported non-colored text, at the time they were a good solution considering the limited abilities of the original PC. As video cards evolved adding the ability to perform with higher resolutions, greater color depths, and the ability to control individual pixels termed "All Point Addressable", they were still very limited as they relied on the main system CPU to perform all of the work. These new video capabilities only increased the workload on the CPU.

In 1984, IBM created the first processor based video card for the PC. The Professional Graphics Adapter (PGA) incorporated its own Intel 8088 microprocessor onboard and took over all video related tasks freeing the main system CPU from having to perform any video processing computation. This was a very important step in graphical processing unit (GPU) evolution since it helped to further the paradigm of having a separate processor perform computations of the graphics computing system family. The PGA video card required 3 card slots in an XT or AT system, a special professional graphics monitor, and at over $4000 was very expensive [62]. This all added up to a short life span for the PGA card, although to its credit it was targeted at the high-end business market for engineering and scientific applications and not at end users and

was capable of 60 frames of 3D animation per second.

With the introduction of VisiOn in 1983, the first graphical user interface (GUI) for the PC, larger system requirements and more computational power were prerequisites for the changing and more important role of the graphics computing system family [83]. New industry wide standards and concepts for graphics computation were required and Silicon Graphics Inc. (SGI) would step in to provide the next stepping stone in the evolution of the GPU.

Silicon Graphics Inc. (SGI) was an up and coming computer graphics hardware company that focused their resources on creating the highest performance computer graphics computers available. Along with their hardware they were also creating industry standards that are the basis for today's computer graphics hardware and software. One of the software standards of today is the platform-independent graphics API OpenGL [33, 58]. Created by SGI in 1989, OpenGL has become the industry's most widely used and supported 2D and 3D application programming interface (API) [57]. Another concept (that will be covered later) pioneered by SGI that is now an intricate part of the design of graphics hardware is the idea of a graphics pipeline. This design approach is nearly the same in all of the major graphical processing unit (GPU) manufacturers like nVIDIA [56], ATI [5], Matrox [42], etc., and has been a major impetus behind the exploding technology of GPUs. The most recent impetus behind the surge in design and implementation in GPUs has been the creation of three-dimensional PC games such as Quake III, Doom, Ultimate Tournament, flight simulators and others.

These games have pushed high powered GPUs into the PC at a phenomenal rate and as GPUs have become more powerful, other 3D applications have been emerging. Some are high-end, supercomputer-class applications that have migrated to the PC, such as geophysical visualization applications used by oil companies [53]. Computer Aided Design (CAD) has done for high-end graphics systems what 3D games have done for the PC. Although both PC graphics and high-end industrial graphics have made great progress in hardware design, it should be noted these two different rendering systems do not have the same goals in mind. Offline rendering systems, such as those used in CAD applications, emphasize accuracy over frame rate where real-time rendering systems, such as game engines and simulators, stress frame rates for smooth and fluid animations and are willing to sacrifice both geometry and texture detail to do this.

One of the things that will be looked at in Chapter 4 is the issue of using the GPU for problems other than for 3D graphics. As GPUs have continued to evolve they have become much more complex than a general purpose CPU. If you just look at transistor count, nVIDIA's current line of GeForce 6 Series GPUs have more than double that on a Pentium 4 with 222 million transistors [8]. Moore's Law states that the density of transistors at a given die size doubles every 12 months, but has since slowed to every 18 months or loosely translated, the performance of processors doubles every 18 months [6, 48]. Within the last 8 to 10 years, GPU manufacturers have found that they have far exceeded the principles of Moore's Law by doubling the transistor count on a given die

size every 6 months [46]. Essentially that is performance on a scale of Moore's Law cubed. With this rate of increase and performance potential one can see why a more in depth look at the GPU for uses other than graphics is well warranted.

The general purpose CPU has a single-threaded processing architecture that allows multiple processes to be run through the same single-threaded pipeline, which accesses their specific data through a single memory interface. GPUs, on the other hand, have a completely different architecture, which is called stream processing. This design is much more efficient at handling large streams of data, which is typical of graphics workloads. A GPU can utilize thousands of stream processors each connected to another stream processor using a very fast dedicated pipeline. Unlike the CPU, since the interconnectivity of the stream processors is through a dedicated pipeline, there are no conflicts or waiting. In the stream processing model, every single transistor on the die is running all the time. This lends itself to the question of whether the GPU will ever overtake the CPU as the main processor in the PC.

3.1 3D Graphics Pipeline

The graphics pipeline is a conceptual model of stages that graphics data is sent through. It can either be implemented in software like OpenGL or Microsoft's Direct3D or it can be implemented in the graphical hardware such as the GPU or some combination. It is simply a process for converting coordinates from what is easier for the application programmer into what is more convenient for the display hardware. The top of the pipeline consists of 3D data in the form of models or shapes made up of

triangles, 3D coordinates (x, y, z), and the pipeline processes that data into pixels that are displayed on the 2D space of the monitor representing the surface of the objects.

There are many different representations that embody the idea of a 3D graphics pipeline, some very complex and some very simple. The entire process, in its simplest form, can be generalized into 2 stages. The first stage is the geometry processing stage that changes 3D object coordinates into 2D window coordinates. The second stage is the rendering stage which fills the area of pixels between the 2D coordinates with pixels to represent the surface of the object [53], shown in Figure 3.1.

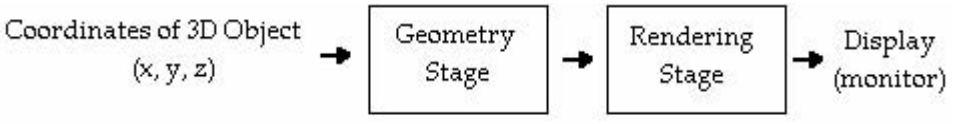

Figure 3.1: Simplified 2-Stage 3D Graphics Pipeline

The geometry stage can be divided into its 2 main components which are transform and lighting and triangle setup shown in figure 3.2.

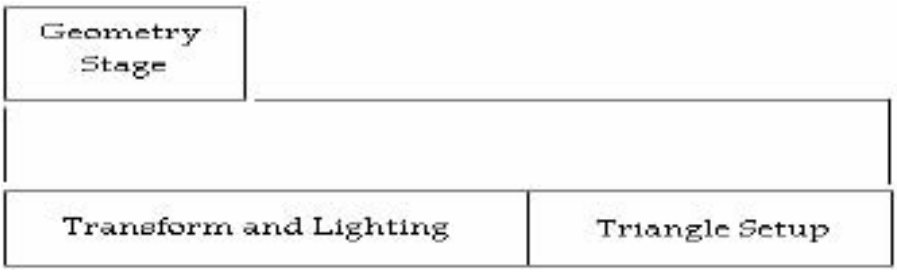

Figure 3.2: Main Components of the Geometry Stage

Implemented in software or hardware, all 3D graphics systems will include these 3 stages. Figure 3.3 shows a 3D pipeline used by NVIDIA Corporation in its GPUs [55].

When GPUs first came about, they started implementing the functionality of the pipeline from the rendering side of the pipeline first. This was a relatively simple process since rendering is as simple as drawing lines with the right color pixels. The rendering engine is fed data that amounts to starting and ending points of lines that make up triangles. The process is repeated for all rows of pixels that make up the triangle, which is very repetitive.

Application Tasks	The application controls the movement of objects (including the camera) and their interaction in the 3D world. Problems such as realistic physics calculations (objects have momentum) and collision detection (did my race car bump the wall or bump another car?) affect how an object moves in the scene.
Scene-level Tasks (culling, LOD, display list)	Scene-level tasks include object level culling (car #3 is completely behind the camera, so it doesn't need to be sent to the next stage), selecting the appropriate detail level (car #1 is far away, so a low-detail model is best), and creating the list of all objects in the current camera view.
Transform	The transform engine converts 3D data from one frame of reference to a new frame of reference. The system must transform the data to the current view before performing the following steps (lighting, triangle setup and rendering). Every object that is displayed and some objects that are not displayed must be transformed every time the scene is redrawn.
Lighting	Lighting is the next step in the 3D pipeline and provides high visual impact. Lighting effects are essential for enhancing the realism of a scene and bringing rendered images one more step closer to our perception of the real world.
Triangle Setup and Clipping	The triangle setup engine is a floating-point math processor that receives vertex data and calculates all of the parameters that the rendering engine will require. This unit 'sets up' and 'clips' the triangle for the rendering engine.
Rendering	Rendering is calculating the correct color for each pixel on the screen, given all of the information delivered by the setup engine. The rendering engine must consider the color of the object, the color and direction of light hitting the object, whether the object is translucent, and what textures apply to the object. If rendering is done in multiple stages, it is called multi-pass rendering.

3D Application and API

3D Graphics Pipeline

Figure 3.3: NVIDIA 3D Graphics Pipeline from [55]

As GPUs progressed they next took on the task of implementing the triangle setup stage in hardware. The triangle setup stage is where the rendering engine gets the starting and ending points of the triangles from. When triangle setup was implemented in software, if a triangle was 1000 rows high, the main CPU had to compute and transfer 1000 different sets of starting and ending points to the graphics board for rendering. Since this is simple but repetitious work, the CPU became the bottleneck in the graphics pipeline. The advantage to having a hardware implementation for the triangle setup stage is that now only one set of 2D coordinates for a single triangle are needed to be transferred to the graphics hardware and the GPU now figures out the starting and ending points of the rows. This also frees up the CPU to be able to do more geometry and application calculations, which allows the CPU to output more triangles.

Now that graphics hardware was handling the triangle setup stage allowing the CPU to output more triangles to be drawn, the CPUs were able to output more triangles than the GPUs could handle. No more was the main computer processor the bottleneck in the graphics pipeline. This created the new problem of how to speed up the graphics hardware to allow it to keep up with the amount of triangles the CPU is outputting. This led 3D hardware designers to use the concepts of pipelining and parallelism. GPUs work similarly to CPUs in that they do an amount of work on data every clock cycle and on early graphics hardware, they took several clock cycles to process each pixel that was drawn. Several clock cycles were used to figure out the start of a row, look up color data for the pixel and also write all of this data to memory. The whole process for drawing a

single pixel took many clock cycles. The idea of pipelining is analogous to an assembly line. A job is made up of multiple tasks each performed sequentially. At each step the same work is performed so instead of processing a set of instructions one pixel at a time, the rendering process is broken up into the individual task components thus allowing each component to complete its specific task on the pixel data and quickly pass it to the next step. Even though it may take several clock cycles to process a single pixel, since the entire process is sequential, there are many pixels in the pipeline at one time each being processed along the way. This means that there will be one pixel rendered per clock cycle. Now that a single pixel can be processed every clock cycle, hardware designers could just double the clock speed effectively doubling the output of pixels per cycle.

Although pipelining drastically increased the pixels per clock cycle, high speed CPUs were still producing more triangles to be processed than the graphics hardware could handle. Even speeding up clock speeds on the GPUs was not enough to handle all of the triangle data from the CPU. Rendering 3D graphics is an extremely repetitive task and since the data is processed the same way every time it is very well suited for the use of parallelism. By adding another pipeline running in parallel to the first, the GPU could now render 2 pixels per clock instead of 1. Current GPUs now have 16 parallel pipelines rendering 16 pixels per clock cycle and up to 8 billion pixels and almost 800 million vertices per second [84]. At this point in time, the bottleneck in the 3D graphics pipeline is no longer the GPU, but now is once again the CPU. There is basically no

limit in using parallelism in graphics hardware. There are some hardware manufacturers with plans for products with hundreds and even thousands of pipelines running in parallel. The issue now is there is no use adding additional parallel pipelines if the CPU can't provide enough data to keep them busy. Since the bottleneck is back to CPU and the transistor count for the GPU doubles every 6 months instead of every 18 months for the CPU, a way was needed to offload more work to the GPU thus freeing up the CPU so that it can provide more triangles to the rendering pipeline. The way the hardware designers did this was to take on the last step of the 3D graphics pipeline, transformation and lighting.

Transformation and lighting is the most difficult part of the 3D graphics pipeline. Transformation is the process of displaying a three-dimensional object onto a two-dimensional screen and lighting is the process of providing lighting effects to the scene [55]. Transformation and lighting are very math intensive processes and was the main reason that CPUs did these computations. The other reason is that the high cost of duplicating the floating-point functions required for transformation and lighting in the graphics processor was prohibitive. Figure 3.4 shows how the GPU has taken on the tasks of the 3D graphics pipeline with time [55].

The geometry calculations that define transform and lighting are accomplished by a special matrix multiplication known as a transform matrix, which is a 4 x 4 matrix. The transform matrix is made up of interim action matrices multiplied together to get the final transform matrix, which are all actions to be performed on the object. An

interim action matrix would be for such actions as scaling, rotation and translation. The transform matrix is multiplied by a 4 x 1 matrix, which is a 3D vertex (corner) of each

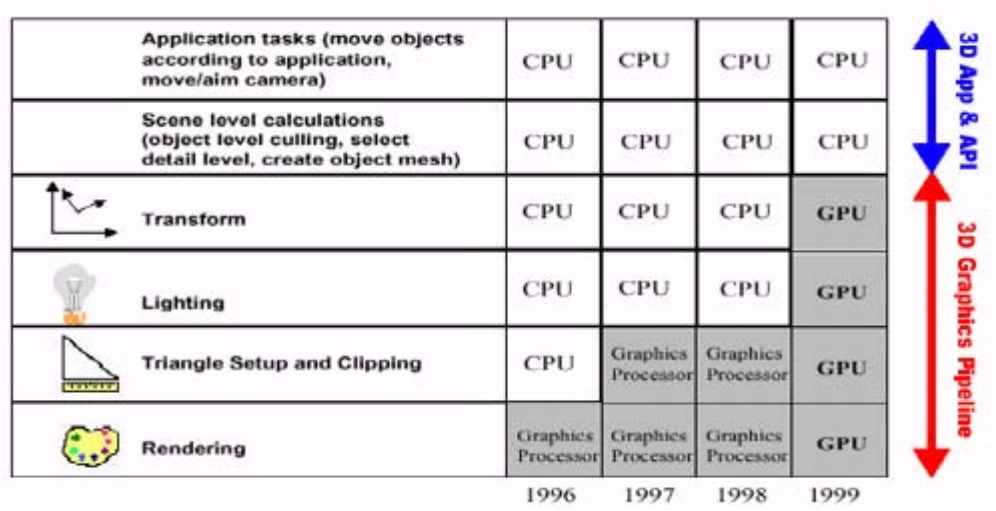

Figure 3.4: NVIDIA 3D Graphics Pipeline Timeline from [55]

triangle. This works out to be at least 12 simple multiplications and 9 additions for each vertex of a triangle shown in Figure 3.5 [55].

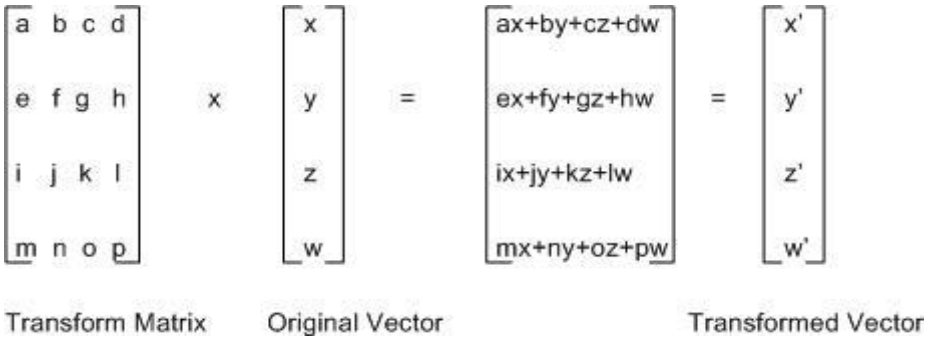

Figure 3.5: Transform Matrix Multiplication from [55]

3. 2 Fixed Function Pipeline vs. Programmable Pipeline

The concept of the GPU has only been recently implemented as of 1999 [16, 18, and 63]. The idea that the entire 3D graphics pipeline is implemented in its entirety on a

single die, similar to the computer's main CPU, has only as of recently been able to be accomplished. Before that time, graphics processing was handled by a combination of the main CPU, graphics co-processors and multi-board graphics add-in cards. This process was generally not a problem for high-end UNIX workstations with almost unlimited budgets. Figure 3.4 showed graphically the transition from CPU to GPU in graphics processing [55].

In 1999, NVIDIA implemented the last step of the 3D graphics pipeline in hardware, thus creating the first consumer level GPU [16]. This was a huge step in graphics rendering because now the GPU did all the work on the graphics data that previously was done by the main CPU. A modern GPU will process 3D graphics pipeline data 2 to 4 times faster than the fastest PC CPU and the added bonus of not having to transfer graphics data from the GPU to the CPU and back is extremely efficient [55]. The graphics data is sent directly to the GPU where it is processed from start to finish of the 3D graphics pipeline and the results are sent directly to the viewing screen (monitor). This frees the CPU up to be used for purposes that it is better equipped to handle like physics calculations, artificial intelligence, application functions and system resource management.

The first implementations of the complete 3D graphics pipeline in hardware (GPU) were called fixed function pipelines because the programmer could not modify data once in the pipeline and its exact functionality was determined by its supported standard, i.e., DirectX or OpenGL. Although polygon processing was much faster, once

the GPU received the position, normal, color, transparency, and texture data the developer had very little control over how the hardware created the final image. The major problem with the fixed function pipeline model is the inflexibility of graphics effects thus hampering creativity. The developer is limited to the specific set of features that are defined by the supported standard i.e., DirectX or OpenGL. As these standards change, the hardware that implements the fixed function pipeline will not be able to take advantage of the newer feature sets whereas a programmable pipeline model would be able to.

The programmable pipeline is the next step in the evolution of the GPU. In 2001 with the introduction of NVIDIA's GeForce3, the ability to program the previous non-programmable pipeline was introduced [32, 73]. Instead of sending the position, normal, color, transparency, and texture data to the GPU and relying on the fixed pipeline functionality of older GPUs to do standard processing of the data, the developer now sends this data to the GPU and writes vertex and pixel shader programs to interpret the data and create an image this allowing for greater flexibility. Vertex shader programs operate at the vertex level and replace the transform and lighting section of the 3D graphics pipeline whereas pixel shader programs operate at the pixel level and replace the multitexturing and blending stages of the pipeline [20]. These shader programs are low-level shading languages that are similar to an assembly language for a general purpose CPU, but different in that their instruction sets are very limited and contain specific instructions for graphics programming [20]. To simplify the

development of graphical programming, two APIs (Application Program Interface) have been created. Microsoft's version is Direct3D which is part of Microsoft's DirectX APIs. OpenGL is the other major API for creating graphical applications. OpenGL was created by SGI and will run on most operating systems where Direct3D/DirectX will only run on Microsoft operating systems.

Just as high-level programming languages for the general purpose CPU emerged to enable the programmer to write programs that are more or less independent of a particular type of computer, so too have high-level languages emerged for the GPU. The Cg programming language or "C for Graphics" is a new C-like high-level programming language for 3D graphics developers which was created by NVIDIA along with Microsoft's help and provides higher level interfaces to DirectX and OpenGL. Microsoft has written its own version that is included in DirectX 9 called HLSL or "High Level Shader Language", which is syntactically and semantically equivalent to Cg except that it can only be used with Direct3D [26]. Glslang is the high level shading language for OpenGL. With the newest GPUs being programmable it allows the developer to be more flexible and creative, but also allows the use of the GPU for other types of computing besides graphics processing since a developer can write a program that runs on the GPU based on the instruction set from the GPU.

3.3 The Stream Processing Model

Getting data to the processor has long been a concern of processor engineers. For as long as we have been using computers, the Von Neumann style architecture has

created memory bottlenecks between main memory and the CPU [6]. Considerable time and energy has been spent finding methods to lessen these bottlenecks. Different approaches such as caches, faster memories, and pre-fetching strategies have been implemented and although these methods have helped to increase the bandwidth between main memory and CPU, the bottleneck continues to grow. Because of this, over 60% of the real estate of the modern CPU is devoted to cache circuitry [34]. Since the types of computations done by graphics cards are very computing intensive, GPUs are very sensitive to these bottlenecks. Because of this GPUs have evolved into special-purpose stream processors.

A significant reason why GPUs have such impressive speed is because they can be considered a special-purpose stream processor; that is, they are highly optimized for stream computations. The computations performed by the GPU are generally simple and require much less memory than the CPU, but the ability to perform many computations very fast and in parallel as much as possible is required. Since the stream model requires little to no memory the memory bottleneck is essentially eliminated allowing the GPU to enable the use of all transistors which increases its computational ability unlike in the CPU. Since GPUs have a Moore's Law that is faster than CPUs and the stream processing model is better suited for computing intensive operations such as operations performed on graphics data, a very interesting consequence of this is the increasing use of the GPU as a general-purpose stream processing engine. It is argued that the GPU can perform any computation that can be mapped to the stream

computing model [63].

Stream computing is different than traditional computing in that it uses "Streams" and "Kernels" to deal with memory-bandwidth bottlenecks. Streams are sets of sequential data elements requiring related computation. A stream is created by appending elements to the back of the stream and consumed by taking elements from the head. They can be combined with other streams through operations that append as well as indexing into other streams. Kernels are small programs that operate on the entirety of streams. A kernel takes one or more streams as input and produces one or more streams as output.

There are several reasons why the streaming model is better suited for applications that tend to have large amounts of data like media (audio/graphics), applications that use sensors, telephone record analysis, database queries, and theoretical Computer Science. One is the idea of parallelism. Taking advantage of parallelism allows high computation rates that are necessary for these types of applications. Three levels that parallelism can be revealed are instruction level, data level, and task level.

A kernel may well perform hundreds of independent instructions on every element of a stream. Many of these instructions can be performed in parallel. This form of parallelism is termed instruction level parallelism. Since kernels perform the same instructions on each element of a stream, data level parallelism can be exploited by performing these instructions on many stream elements at the same time. Task level

parallelism is the ability to have multiple stream processors divide the work from one kernel or have different kernels run on different stream processors [34].

The way a traditional processor tries to reduce off-chip memory access is by using caches. Caches take advantage of the spatial and temporal locality which works well for traditional processors, but not stream processors. Stream processors do not take advantage of the caching capabilities of traditional processors since stream processor input datasets are much larger than most caches can handle and the input primitives have little to no reuse since they are generally rendered and discarded. Because the output stream of each kernel is the input stream to another kernel, a stream processor takes advantage of the producer/consumer locality. Since memory access is very expensive, a stream processor utilizes a bandwidth hierarchical system which keeps most of the memory storage/access local for temporary use during calculations, but does allow for infrequent use of the stream register files for memory storage/access for uses between kernels and rarely accesses main memory for large, sparsely accessed data. Stream processors are particularly proficient at minimizing the memory to arithmetic operations ratio. The traditional accumulator has a ratio of 1:1 while a scalar processor has a ratio of 1:4 and a stream processor has a ratio of 1:100 [66, 67].

Chapter 4 General Purpose GPU Computing

"All processors aspire to be general-purpose"

-Tim Van Hook, Keynote, Graphics Hardware 2001

Graphics hardware, as its name suggests, has come into existence with image and graphics computations in mind, but newer more programmable hardware has enabled the exploitation of more general computations. Much of the calculations involved with the images seen on the screen are not directly related to images, but have more general mathematical characteristics. It is this more general mathematical characteristic along with its stream processing model of computation that allows the GPU to be extremely well suited for fast, efficient, non-graphical computing.

The stages of the 3D graphics pipeline utilized by the GPU have evolved from a software implementation to a graphics co-processor of the main CPU to a fixed-function graphics pipeline to a programmable graphics pipeline and the stages are becoming more and more programmable and most importantly more general. As these programmable stages become more general in nature the GPU can be thought of as a general-purpose stream processor. Since CPUs provide higher performance on sequential code it is becoming much more difficult to just add additional transistors to improve performance for this type of code. Conversely, since programmable GPUs are optimized for highly parallel vertex and fragment shading code, GPUs can use additional transistors much more effectively which allows GPUs to have a higher rate of performance as semiconductor technology improves [40].

Because GPUs have evolved into very efficient and computationally powerful processors, researchers have studied ways in which the power of GPUs could be harnessed to solve problems other than graphics. There are many different websites and research papers devoted to this topic. Many different problems have been studied from linear algebra to fast Fourier transforms and a UNIX password cracker that will "crack" any UNIX password in 2 days [38].

The GAMMA research group at UNC Chapel Hill uses GPUs for such non-graphics problems as Voronoi computations, collision detection, cloud simulation, ice crystal growth simulation, viscous paint simulation and more [27]. At a company named Frantic Films in Canada, they used a GPU to do computational fluid dynamics to produce accurate visual fluid effects [27]. Although there were some technical limitations with the current version of GPU used, they realized a 50% speed up over their fastest CPU-based version and felt the gap would widen with newer versions of GPUs.

At Stanford University, they are working on a prototype processor that is based on the stream architecture. This processor is a single-chip programmable processor that supports the stream programming model and is named Imagine [35]. Although Imagine is beyond the scope of this paper, a few points are brought out to show the power of the stream processor. Figure 4.1 shows a block diagram of this processor. Each ALU (arithmetic logic unit) cluster contains 6 ALUs thus allowing Imagine to support 48 ALUs. These clustered ALUs reach a peak computation rate of 16.2 GOPS (giga, or

billions, of operations per second) for both single precision floating point and 32-bit Integer arithmetic [66]. Figure 4.2 shows the bandwidth hierarchy of the Imagine processor. This 3-tiered storage bandwidth hierarchy is the reason Imagine is able to support 48 ALUs. The greatest data bandwidth demand is within a kernel. From figures 4.1 and 4.2 one can see that Imagine provides the most data bandwidth from within a cluster and maps kernel execution onto the clusters [35].

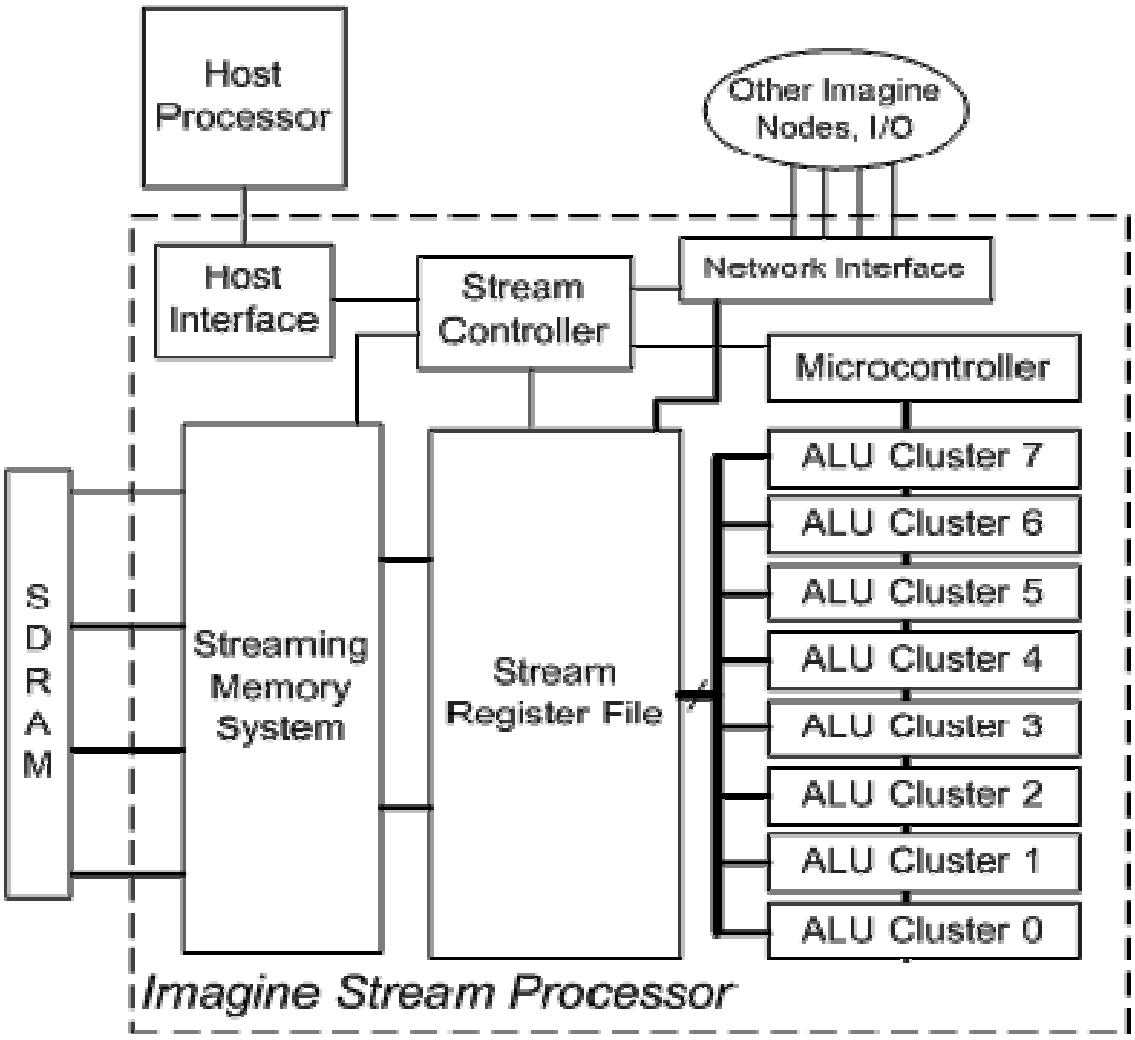

Figure 4.1: Imagine Stream Processor Block Diagram from [35]

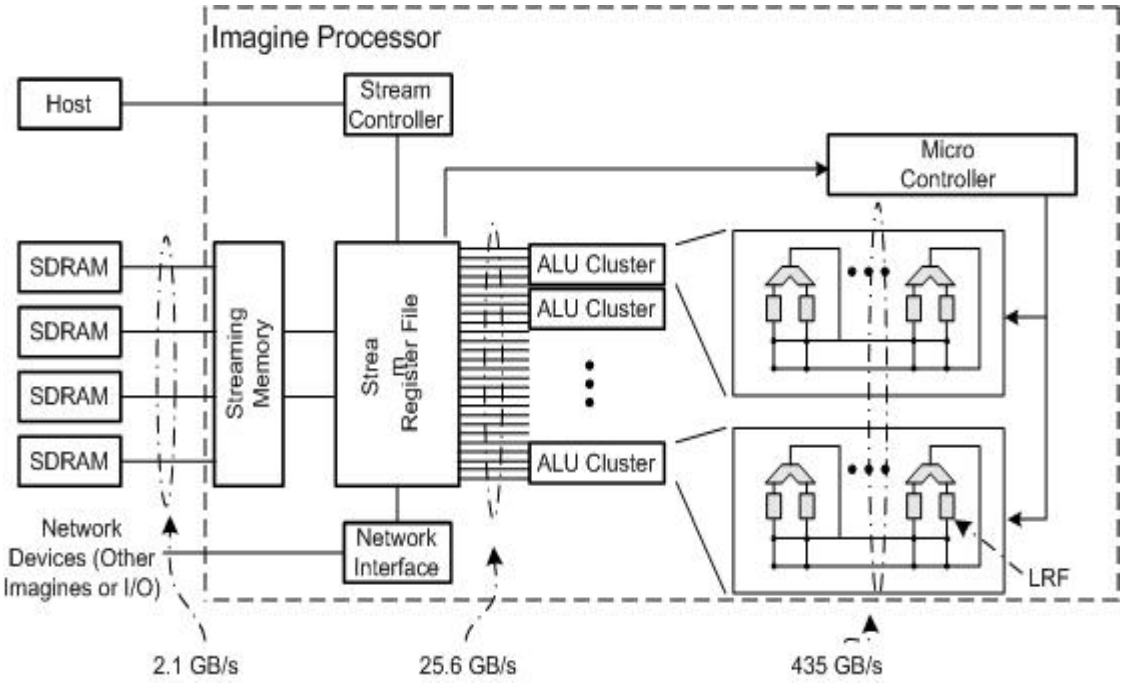

Figure 4.2: Imagine Stream Processor Bandwidth Diagram from [35]

4.1 Matrix-Matrix Multiplication – A Test Case

There are several reasons why you might attempt to use a GPU for general purpose computing, the most important of which is the raw computational power of a GPU. Another equally valid reason would be one of cost. The GPU beats the CPU in both raw computational power measured in GFLOPS as well as overall cost as will be shown later in this section.

A measurement of performance commonly used to gauge the processing speed of a processing unit is FLOPS (floating-point operations per second). Floating-point operations are those that involve fractional numbers. A 3 GHz Pentium 4 CPU can perform 6 GFLOPS (gigaflops), whereas the GPU on an NVIDIA GeForce FX 5900 Ultra can perform 20 GFLOPS, which is roughly equivalent to a 10 GHz Pentium 4 [12].

The newest card from NVIDIA is the GeForce 6 Series. Although we were unable to find performance data on this new card, the technical data shows that with nearly 5 GB/sec increase in memory bandwidth, 256 bit memory interface compared to 128 bit, 16 pixel pipelines compared to 8 and the capability of producing 6.4 billion texels/sec compared to 3.8 all in a single rendering pass, the GeForce 6 Series should well outperform the FX 5900 Ultra [56].

Linear systems can be found throughout scientific computing like mathematics and engineering. The importance of Linear Algebra to scientific computing cannot be stressed enough. Thus, Matrix-matrix multiplication is a good indicator of how well a processor will perform on scientific computation or general purpose computing.

We compared the performance of matrix-matrix multiplication of different sized matrices on a CPU versus the same sized matrices on a GPU. In this test case the CPU was a Pentium III 750MHz and the GPU was an NVIDIA GeForce FX 5200. The GeForce FX 5200 was the first GPU with a fully programmable 3D graphics pipeline and the Pentium III 750MHz was chosen as a close comparison for this GPU. The software package used to test the CPU was ScienceMark 2.0, which uses BLAS (Basic Linear Algebra Subprograms) to test processor performance [72]. Source code from the GPUBench suite of performance testing tools was used to test the GPU, which is Cg (C for Graphics) coded functions that are computed directly on the GPU [31]. The many subprograms of this suite test several performance aspects of newer programmable GPUs from NVIDIA and ATI. Requirements for the programming environment are

Microsoft Visual Studio .Net 2003 and Cygwin, which is a Linux environment for the Windows operating system. Once the environment was set up and the source code successfully compiled, matrix-matrix multiplication was performed on square matrices of dimensions from 32x32 up to 1344x1344.

4.2 Results

One of many different benchmark variables used to measure processor performance is that of GFLOPS. Comparison of the results of the CPU vs. GPU in Figure 4.3 shows that the GPU is out performed by the CPU in matrix-matrix multiplication.

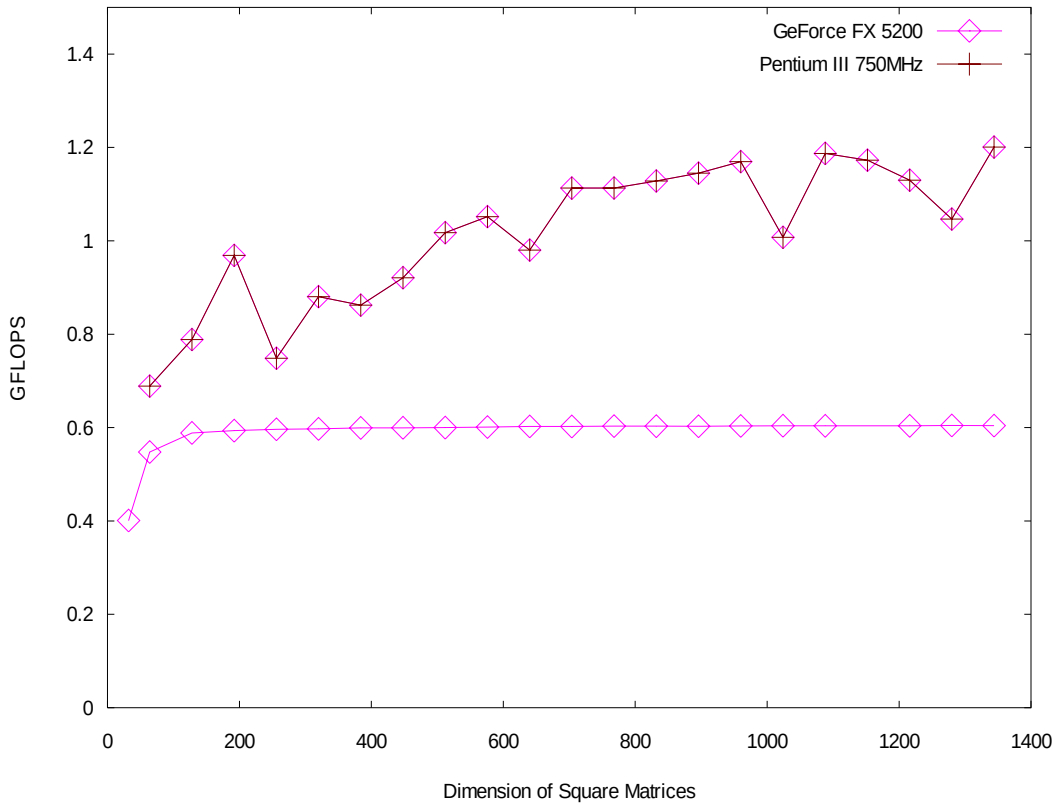

Figure 4.3: Matrix-Matrix Multiplication Results

The GPU performance curve is fairly smooth while the performance curve for the CPU has several peaks and valleys. This is explained by the fact that the cache of the CPU is relatively large compared to the memory of the GPU. As is the case with computing matrix-multiplication on large matrices and computing with large datasets in general and data reuse, a processor must utilize its caches rather frequently. Since the size of memory the GPU has available to it is rather small and the matrices in this test were larger than the memory could store, the GPU was constantly accessing the system's main memory in contrast to the CPU which accessed the main memory much less frequently. The performance curve of the GPU was rather smooth because throughout the tests it was constantly accessing main memory since the GPU memory was always full. The dips in the CPU performance curve occurred when the CPUs cache filled up and it had to access main memory. So unlike the GPU, the CPU was able to perform substantial computations while staying within its own cache which is very beneficial since when a processor has to access the main system's memory there is a substantial bandwidth bottleneck that severely degrades performance.

Theoretical peak GFLOPS is the calculation of maximum number of FLOPS the processor could perform in an absolutely ideal environment with no bottlenecks. The equation for theoretical peak GFLOPS for an Intel Pentium III or 4 processor is four times the MHz of the processor so that a Pentium 4 3GHz can theoretically perform 12 GFLOPS [61, 82]. In this test case the Pentium III 750MHz can theoretically perform 3 GFLOPS. While getting information on GPUs has been much more difficult due to the

fact that GPUs have a much faster time to manufacturing, it can be shown that the GeForce FX 5200 has a theoretical peak of approximately 4 GFLOPS [28]. In this test case the GPU is capable of 25% more GFLOPS than the test CPU, but was found to perform half as well as the CPU.

Comparing the efficiency of both the CPU and GPU implementation, which is the ratio of measured peak GFLOPS to theoretical GFLOPS, shows the CPU had an efficiency of about 40% while the GPU was only able to reach about 15% efficiency. This shows that while doing the GPU implementation of matrix -matrix multiplication, much of the transistors of the GPU were idle. A similar, but much more thorough investigation was performed with the most current GPUs from ATI and NVIDIA (ATI 9800XT, ATI X800XT, GeForce FX 5900 Ultra, GeForce 6800 Ultra) that showed similar results [24]. The highest efficiencies achieved by NVIDIA and ATI were found to be 17% and 19% respectively.

Since GPUs have a Moore's Law much greater than that of a CPU and can perform more peak GFLOPS , it was expected that the test case would show the GPU to far outperform the chosen CPU, but this in fact was not the case. As mentioned earlier, graphics computation starts by sending 2D coordinates of a triangle to the GPU and graphics computation requires very little data reuse because of its mapping to the stream model. Because of this type of computation, GPUs are highly tuned for graphics rendering algorithms, which do not account for high levels of data reuse. Once data needs to be recycled, the bottlenecks become the bandwidth between the GPU and

graphics card memory and the bandwidth between the graphics card and the computer's main memory and the CPU. General purpose computation generally requires data reuse and accumulation of data from previous results which the GPU is not very good at especially since the GPUs memory is relatively small and memory bandwidth is much slower than the CPUs [24]. It is no wonder CPUs are more efficient at general purpose computing since 60% of their die size is for cache circuitry.

As the study in [24] shows, the GeForce 6 Series does in fact outperform a 3 GHz Pentium 4 CPU on the problem of matrix-matrix multiplication. The fact that it does so while being so inefficient compared to the CPU shows the raw performance potential of the GPU. The Current GPU is not well suited for all types of problems. It has been designed and tuned to be extremely efficient at single-pass graphics rendering specifically for gaming graphics as that is the intended market for these chips. With the current architecture of the GPU, any problem that requires large amounts of data reuse will not run as efficiently as it would if implemented on a mainstream CPU. One solution to this problem would be to look at designing algorithms to better exploit the advantages the current GPU architecture design presents while at the same time limiting the GPU weakness of smaller memories and slower data paths. Another solution would be for the graphics card manufacturers to include rather small overall hardware enhancements such as larger memories and wider data paths to their cards that would not only benefit general purpose computing and expand their market, but also their target market of gamers.

Chapter 5 Future of the GPU

One of the major reasons the GPU has succeeded where many other non-mainstream processors have failed is simply economics. It is not that the GPU is simpler or cheaper to build than other processors, it is that the GPU has a booming market that is driving sales at high volumes and that market is computer games. As many researchers and non-gamers have found, many computations have just the sort of parallelism that maps well to the pseudo-stream processing GPUs. Many experiments have shown that computation and simulation can be accelerated by an order of magnitude just by simply investing a few hundred dollars on a commodity graphics card to do the computation [7, 30, 40, and 63].

The future of the GPU certainly has much more promise on the horizon than the general-purpose CPU. Although the GPU will not over take the CPU as the main processor, we do think the GPU has much more potential for expanding our computing experience. The CPU has a large amount of logic dedicated to branch prediction, whereas stream processing does not require as much of this type of logic. The GPU is much better at parallelism than is the CPU and as the gap in transistor rate expansion continues to grow the GPUs parallelism performance will also continue to grow. One of the issues that will need to be addressed as the GPU continues to outpace the CPU is that of being able to program highly parallel stream processors such as GPUs to be able to solve massive and complex problems, graphical and non-graphical alike. It is not a simple task to present a standard sequential programming problem in such a way that it

can effectively take advantage of several processors, not to mention hundreds or even thousands. It is also not a simple matter to do the same for a feed-forward stream processor such as the GPU. The power of solving these highly parallel problems has immense implications to the scientific community because we are then able to change the evolution of scientific computation from the CPU growth curve from doubling every 18 months to the GPU growth curve that doubles about every 6 months.

As the GPU design continues to advance there are a few areas that need to develop inline with the hardware. One area of work needed is designing new algorithms to take advantage of the ability of these new GPUs to work in parallel. This could start with the idea of requiring problem solving from the aspect of parallel dataflow on streams instead of restating problems to fit the new GPUs. Another area is constructing new languages specifically to take advantage of the high level of parallelism and data streaming capabilities of GPUs. Lastly is the area of constructing compilers and other software tools to advance parallel stream programming. There is currently work underway at Stanford University in the field of language and compiler design specifically for the GPU. It is a project called BrookGPU and is designed to work on modern GPUs [11]. It is a general purpose language, compiler and run-time environment designed to allow developers to implement general purpose programming on GPUs as well as take advantage of data parallelism and arithmetic intensity.

At this point in time, the GPU is still considered a special-purpose processor geared for compute-intensive graphics calculations. "Wheel of reincarnation" is a term

coined in an early graphics paper by Myer and Sutherland [51] that is used to refer to a known fact in computing whereby functions are shifted out to special-purpose hardware because of speed and because special-purpose hardware is more expensive than general-purpose hardware, eventually this functionality is included in the next iteration of the general-purpose hardware and this cycle is continually repeated. Although the inclusion of the GPU circuitry into the general-purpose CPU could prove a retardant to computing advancement, there are two reasons it would be difficult for this to happen. One is the fact that the cost of the commodity GPU is relatively inexpensive mainly because high volumes are sold to PC gamers. This fact makes the trade-off of having high cost special-purpose hardware less of a factor. The second reason is that the Moore's Law of the GPU is three times as great as the CPU and will continue to grow in the future. This makes it impossible for CPU manufacturers to keep up with the rapid pace of GPU advancement and would be too costly to re-die a new CPU every time a new GPU chipset came out.

One of the things that could help with general purpose computation on the GPU is to make the data path from the GPU to its cache wider lowering the cost of data reuse. Another improvement would be to widen the bandwidth from the GPU to the main system. Currently the AGP data bus used by main system boards to connect the graphics subsystem to the CPU and main memory uses AGP 3.0 and can transfer data at a rate of 2.1GB/sec (gigabytes per second) [54, 73]. The new standard that has just been released is PCIExpress and data can be transferred at a rate of 4GB/sec and will be

scalable in the future [59].

With the advent of the PCI Express standard and the extra bandwidth it brings, nVIDIA has announced a new technology called SLI (scalable link interface) aimed at PC game enthusiasts and high-end workstations [76]. This new technology will allow two high-end GPUs to be combined on one motherboard working in parallel essentially doubling the performance provided by the graphics subsystem. Of course only their highest-end GeForce 6 Series cards can be used and a special motherboard that has two PCIExpress card slots is required, but with the introduction of the SLI technology a regular computer user can now have the benefit of a two GPU cluster. It is just a matter of time that this technology is expanded or gives way to an even better technology that allows the casual user to chain multiple GPUs together on one motherboard instead of just the allowed two GPUs with the SLI technology.

Beowulf clusters came about from the desire to have a high performance supercomputer without the expense of an actual supercomputer by using cost-effective lower end computers and open source Linux [44]. Much like Beowulf clusters, with all the interest of general purpose computing on a GPU because of the high performance and the low cost of a commodity GPU, GPU clusters are now being created to do very useful computing. At Stony Brook University they have created a 32 node GPU cluster that has an increase of 512 GFLOPS over an identical 32 node CPU cluster [23]. This cluster is used to simulate the transport of airborne contaminants in the Times Square area of New York City and has shown a speed-up of 4.6 times and better compared to

the same model run on a 32 node CPU cluster.

> "The graphical processing unit is visually and visibly changing the course of general purpose computing"
>
> -Michael Macedonia, IEEE Computer, October 2003, "The GPU Enters Computing's Mainstream"

References

[1] Allison, D., "Ken Olsen Interview", retrieved November 2004 from http://www.cwheroes.org/oral_history_archive/ken_olsen/index.asp#

[2] The Amazing World of Computer Graphics, Computer History Museum Homepage retrieved November 2004 from http://www.computerhistory.org/VirtualVisibleStorage/artifact_main.php?tax_id=04.06.05.00#0

[3] Altair, Smithsonian National Museum of American History Homepage retrieved November 2004 from http://americanhistory.si.edu/csr/comphist/objects/altair.htm

[4] Altair 8800, Computer History Museum Homepage retrieved November 2004 from http://www.computerhistory.org/VisibleStorage/images/IMG_2757_lg.jpg

[5] ATI, ATI Homepage retrieved November 2004 from http://www.ati.com/

[6] Backus, J., Can Programming Be Liberated from the von Newmann Style?, A Functional Style and Its Algebra of Programs, *Communications of the ACM* 21, pp. 613-641, August 1978

[7] Batty, C., Wiebe, M., Houston, B., "High Performance Production-Quality Fluid Simulation via NVIDIA's QuadroFX", retrieved November 2004 from http://film.nvidia.com/docs/cp/4449/frantic_GPUaccelerationoffluids.pdf

[8] Baumann, D., "NV40 – GeForce 6800 Ultra Review", retrieved November 2004 from http://www.beyond3d.com/previews/nvidia/nv40

[9] Bell, G.C., Mudge, C.J., McNamara,J.E., *Computer Engineering: A DEC View of Hardware Systems Design*, pp. 33, Digital Press, 1978

[10] Bissel, D., Was the IDIIOM the first stand-alone CAD platform?, *IEEE Annals of the History of Computing* , Vol. 20, No. 2, pp. 14-19, 1998

[11] BrookGPU, Stanford University BrookGPU Homepage retrieved November 2004 from http://www.graphics.stanford.edu/projects/brookgpu/index.html

[12] Buck, I., Purcell, T., A Toolkit for Computation on GPUs. In R. GPUGems: Programming Techniques, Tips, and Tricks for Real-Time Graphics, Fernando R., (Ed.), 1st ed. (pp. 621-636) Boston: Addison-Wesseley 2004

[13] Carlson, W., "A Critical History of Computer Graphics and Animation", retrieved November 2004 from http://accad.osu.edu/~waynec/history/lesson3.html

[14] Computer Graphics Comes of Age, An Interview with Andries Van Dam, *Communications of the ACM*, Vol. 27, No. 7, July 1984

[15] Core Memory, Computer History Museum Homepage retrieved November 2004 from http://www.computerhistory.org/VirtualVisibleStorage/artifact_main.php?tax_id=02.03.00

[16] De Gelas, J., "A Long Look at the GeForce 256", retrieved November 2004 from http://www.aceshardware.com/Spades/read.php?article_id=80

[17] *DEC used by Digital itself: PDP-11 Processor Handbook*, pp. 8, Digital Press, 1973

[18] Dempski, K., *Real-Time Rendering Tricks and Techniques in DirectX*, Premier Press, March 2002

[19] *Digital at Work*, pp. 33, Digital Press, 1992

[20] Ecker, M., "XEngine Programmable Graphics Pipeline Architectures", retrieved November 2004 from http://xengine.sourceforge.net/pdf/Programmable%20Graphics%20Pipeline%20Architectures.pdf

[21] Edwards, P.N., *The closed world: computers and the politics of discourse in Cold War America* , MIT Press, 1996

[22] Fallon, K.K., Early Computer Graphics Developments in the Architecture, Engineering, and Construction Industry, *IEEE Annals of the History of Computing*, Vol. 20, No. 2, 1998

[23] Fan, Z., Que, F., Kaufman, A., Yoakum-Stover, S., GPU Cluster for High Performance Computing, ACM/IEEE Supercomputing Conference, November 2004

[24] Fatahalian, K., Sugerman, J., Hanrahan, P., Understanding the Efficiency of GPU Algorithms for Matrix-Matrix Multiplication, Graphics Hardware 2004

[25] Fetter, W., A Computer Graphic Human Figure System Applicable to Kinesiology, *Annual ACM IEEE Design Automation Conference, Proceedings of the no 15 design automation conference on Design automation*, pp. 297, 1978

[26] Fosner, R., *Real-Time Shader Programming*, Morgan Kaufmann Publishers, January 2003

[27] GAMMA, UNC Team GAMMA Homepage retrieved November 2004 from http://www.cs.unc.edu/~geom/hardware/index.shtml

[28] Gasior, G., "Differentiating features, or the lack thereof", retrieved November 2004 from http://techreport.com/etc/2003q1/nv31-34pre/index.x?pg=2

[29] Gilmore Jr., J.T., Peterson, H.P., "A Functional Description of the TX-0 Computer", Lincoln Laboratory, Massachusetts Institute of Technology, Memo. 6M-4789-1, October 3, 1958, retrieved November 2004 from http://www.tixo.org/documents/6M-4789-1_TX0_funcDescr.pdf

[30] GPGPU, UNC General-Purpose Computation Using Graphics Hardware Homepage retrieved November 2004 from http://www.gpgpu.org

[31] GPUBench, sourceFORGE Homepage retrieved November 2004 from http://sourceforge.net/projects/gpubench

[32] Hanrahan, P., "Why is Graphics Hardware so Fast?", retrieved November 2004 from http://www.graphics.stanford.edu/~hanrahan/talks/why

[33] Hawkins, K., Astle, D., LaMothe, A., "What is OpenGL?", retrieved November 2004 from http://www.developer.com/tech/article.php/947051

[34] Humphreys, G., Houston, M., Ng, R., Frank, R., Ahern, S., Kirchner, P.D., Klosowski, J.T., Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters, *Presented at SIGGRAPH*, San Antonio, Texas, 2002

[35] Imagine, Stanford University Imagine Homepage retrieved November 2004 from http://cva.stanford.edu/imagine

[36] Intel 4004, Intel 4004 Homepage retrieved November 2004 from http://www.intel4004.com/

[37] Jacobs, J.F., *The SAGE Air Defense System: A Personal History*, MITRE Corporation, 1986

[38] Kedem, G., Ishihara, Y., "Brute Force Attack on Unix Passwords with SIMD Computer", retrieved November 2004 from http://www.usenix.org/events/sec99/full_papers/kedem/kedem.pdf

[39] Krull, F.N., The Origin of Computer Graphics within General Motors, *IEEE Annals of the History of Computing*, Vol. 16, No. 3, pp. 40-56, 1994

[40] Lindholm, E., Kilgard, M.J., Moreton, H., A user-programmable vertex engine, *Proceedings of ACM SIGGRAPH 2001,* pp. 149-158, 2001

[41] Machover, C., A Brief, Personal History of Computer Graphics, *IEEE Computer*, pp. 38-45, November 1978

[42] Matrox, Matrox Homepage retrieved November 2004 from http://www.matrox.com/

[43] McKenzie, J.A., *TX-0 Computer History*, MIT RLE Technical Report No. 627, June 1999

[44] Merkey, P., "Beowulf History", retrieved November 2004 from http://www.beowulf.org/overview/history.html

[45] Mims III, F.M., The Altair Story; Early Days at MITS, *Creative Computing* , Vol. 10, No. 11, Pp. 17, November 1984

[46] Moller, J., *Real-Time Rendering,* A. K. Peters, 2002

[47] Moore, G.E., Cramming more components onto integrated circuits, *Electronics,* Vol. 38, No. 8, pp. 114-117, April 1965

[48] Moore's Law, Webopedia Homepage retrieved November 2004 from http://www.webopedia.com/term/m/moores_law.html

[49] Myers, B.A., A Brief History of Human-Computer Interaction Technology, *ACM Interactions,* Vol. 5, No. 2, pp. 44-54, March/April 1998

[50] Myers, B.A., Hollan, J., Cruz, I., Strategic Directions in Human-Computer Interaction, *ACM Computing Surveys*, Vol. 28, No. 4, December 1996

[51] Myer, T.H., Sutherland, I.E., On the Design of Display Processors, *Communications of ACM*, Vol. 11, No. 6, June 1968

[52] National Research Council, *Funding a Revolution: Government Support for Computing Research* , Washington, DC, National Academy Press, 1999

[53] NVIDIA Corp., "3D Graphics Demystified", retrieved November 2004 from http://developer.nvidia.com/object/3d_graphics_demystified.html

[54] NVIDIA Corp., "Technical Brief: AGP 8X Evolving the Graphics Interface", retrieved November 2004 from http://www.nvidia.com/object/LO_20020923_5956.html

[55] NVIDIA Corp., "Technical Brief: Transform and Lighting", retrieved November 2004 from http://www.nvidia.com/object/Technical_Brief_TandL.html

[56] NVIDIA Corp., NVIDIA Homepage retrieved November 2004 from http://www.nvidia.com/page/home

[57] OpenGL, "OpenGL Overview", retrieved November 2004 from http://www.opengl.org/about/overview.html#1

[58] OpenGL, SGI OpenGL Homepage retrieved November 2004 from http://www.sgi.com/products/software/opengl/

[59] PCIExpress, PCMagazine Homepage retrieved November 2004 from http://www.pcmag.com/article2/0,1759,1636639,00.asp

[60] PDP-1, Computer History Museum Homepage retrieved November 2004 from http://www.computerhistory.org/VisibleStorage/images/DSC02956_lg.jpg

[61] Processors, Intel Homepage retrieved November 2004 from http://support.intel.com/support/processors

[62] Professional Graphics Adapter (PGA), Inside Computers Homepage retrieved November 2004 http://library.thinkquest.org/C006208/data/multimedia-videocards.php

[63] Purcell, T.J., Buck, I., Mark, W.R., Hanrahan, P., Ray Tracing on Programmable Graphics Hardware, *Proceedings of ACM SIGGRAPH 2002,* pp. 703-712, 2002

[64] Redmond, K., Smith, T.M., *From Whirlwind to MITRE: The R&D Story of the SAGE Air Defense Computer*, M.I.T. Press, 2000

[65] Redmond, K.C., Smith, T.M., *Project Whirlwind: The History of a Pioneer Computer*, Digital Press, 1980

[66] Rixner, S., Dally, W.J., Kapasi, U.J., Khailany, B., Lopez-Lanunas, A., Mattson, P., Owens, J.D., A Bandwidth-Efficient Architecture for Media Processing, *Proceedings of the 31st Annual ACM/IEEE International Symposium on Microarchitecture,* pp. 3-13, 1998

[67] Rixner, S., Dally, W.J., Kapasi, U.J., Mattson, P., Owens, J.D., Memory Access Scheduling, *Proceedings of the 27th International Symposium on Computer Architecture,* pp. 128-138, June 2002

[68] SAGE, Computer History Museum Homepage retrieved November 2004 from http://www.computerhistory.org/timeline/timeline.php?timeline_category=cmptr

[69] SAGE, Computer History Museum Homepage retrieved November 2004 from http://www.computerhistory.org/VirtualVisibleStorage/popup_image.php?base_name=X260.83---X277.82

[70] SAGE Light Pen, Computer History Museum Homepage retrieved November 2004 from http://archive.computerhistory.org/resources/still-image/SAGE/AN_FSQ-7_SAGE.102622768.lg.jpg

[71] Schein, E.H., DeLisi, P.S., Kampas, P.J., Sonduck, M.M., *DEC Is Dead, Long Live DEC: The Lasting Legacy of Digital Equipment Corporation*, Barrett-Koehler, 2003

[72] ScienceMark, ScienceMark Homepage, retrieved November 2004 from http://www.sciencemark.org

[73] Shimpi, A., "NVIDIA Introduces GeForce FX (NV30)", retrieved November 2004 from http://www.anandtech.com/printarticle.html?i=1749

[74] Sketchpad, Computer History Museum Homepage retrieved November 2004 from http://www.computerhistory.org/VisibleStorage/images/102637018_lg.jpg

[75] Sketchpad, Sun Microsystems Ivan E. Sutherland Homepage retrieved November 2004 from http://www.sun.com/960710/feature3/sketchpad.html#sketch

[76] SLI, nVIDIA SLI Homepage retrieved November 2004 from http://www.nvidia.com/page/sli.html

[77] Sutherland, I. E.,"Sketchpad: A man-machine graphical communication system", retrieved November 2004 from http://www.cl.cam.ac.uk/TechReports/UCAM-CL-TR-574.pdf

[78] Tropp, H.S., A perspective on SAGE: Discussion, *Annals of the History of Computing* , Vol. 5, No. 4, pp. 375-398, 1983

[79] TX-0, Bitsavers Organization Homepage retrieved November 2004 from http://www.bitsavers.org/pdf/mit/tx-0/pics/TCMR-V08-P04.jpg

[80] TX-2, Sun Microsystems Homepage retrieved November 2004 from http://www.sun.com/960710/feature3/tx-2.html

[81] VisiOn, GUI Gallery Homepage retrieved November 2004 from http://toastytech.com/guis/vision.html

[82] Whaley, R.C., "x86 Overview", retrieved November 2004 from http://www.cs.utk.edu/~rwhaley/ATLAS/x86.html

[83] Whirlwind, CEDMagic Homepage retrieved November 2004 from http://www.cedmagic.com/history/whirlwind-computer.html

[84] Zealot, F., "ATI Announces New High End Graphics Card RADEON X800", retrieved November 2004 from http://www.gamepro.com/computer/pc/games/news/35333.shtml