General purpose GPU programming

Dimitar Atanasov

Abstract: *The paper describes programming features of the modern graphics hardware. It accents on the features introduced in Microsoft DirectX 9.0 and using them in general purpose programming of the graphics hardware for effective offload of the CPU. As reference is used NVIDIA's GeForce FX GPUs.*

Key words: *GPU, DirectX, Pixel and Vertex Shaders.*

INTRODUCTION

Modern computer graphics hardware contain extremely powerful graphics processing units (GPU). These GPUs are designed to perform a limited number of operations on very large amounts of data. They typically have more than one processing pipeline working in parallel with each other. They can in fact be thought of as highly parallel Single Instruction Multiple Data (SIMD) type processors.

The performance of these GPUs is also growing at an extraordinary rate. In fact over the last decade or so the processing power of GPUs has been growing at a rate faster than Moore's Law, which governs the performance growth rate of CPUs. In a presentation at Graphics Hardware 2003 titled "Data Parallel Computing on Graphics Hardware", Ian Buck estimated that the current Nvidia GeForce FX 5900 GPU performance peaks at 20 GigaFlops. This is equivalent to a 10-GHz Pentium 4 processor.

The latest generation of graphics hardware also contain much more programmable GPUs. Previously the number of operations that could be performed on the GPU was limited to certain fixed functions such as Texture and Lighting. However the GPU has evolved to a situation where we now have user programmable vertex and texture units. These are commonly referred to as vertex shaders and fragment shaders respectively. These programmable shaders allow for much more realistic visual effects in computer games especially, which is the main driving force behind the computer graphics hardware industry.

A further improvement in these new GPUs is the increase in pixel depth from 32 bits per pixel to 128 bits per pixel. This means that each red, green, blue, and alpha component can now have 32-bit floating point accuracy throughout the graphics pipeline. This increase in data accuracy combined with the increased programmability of the GPU means that the GPU is moving towards a more general purpose processor design.

In the last few years there has been an increase in research in the area of using graphics hardware for general purpose computing. Yang and Welch [1] show how to perform fast image segmentation and smoothing using graphics hardware. They implemented functions like erosion and dilation on the GPU. They state that this implementation is over 30% faster that a similar CPU implementation. Larsen and McAllister [2] describe a technique for doing fast matrix multiplies using graphics hardware. Typically these were very large matrices. There is also an implementation of the Fast Fourier Transform implemented on the GPU [3].

1. GPU

The accelerators are gradually approaching common general-purpose processors in several aspects:

- Considerable increase of clock speeds;
- The rough force is now being replaced by fine optimization algorithms and approaches;
- Computational aspect is in the forefront;
- Developed system of the general-graphic-purpose commands;

- Support of several universal formats (types) of data;
- Possible superscalar and speculative execution;
- Complexity and flexibility of programs are getting less limited.

A representative of programmable GPU is NVIDIA's GeForce FX family (fig. 1)

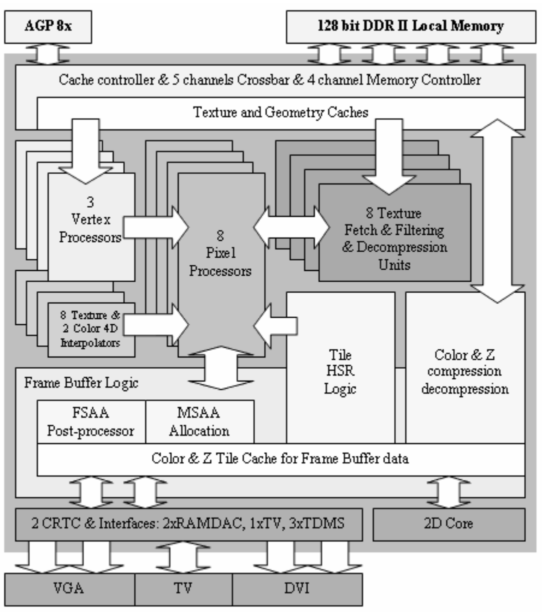
AGP 8x
128 bit DDR II Local Memory
Cache controller & 5 channels Crossbar & 4 channel Memory Controller
Texture and Geometry Caches
3 Vertex Processors
8 Pixel Processors
8 Texture Fetch & Filtering & Decompression Units
8 Texture & 2 Color 4D Interpolators
Tile HSR Logic
Color & Z compression decompression
Frame Buffer Logic
FSAA Post-processor
MSAA Allocation
Color & Z Tile Cache for Frame Buffer data
2 CRTC & Interfaces: 2xRAMDAC, 1xTV, 3xTDMS
2D Core
VGA
TV
DVI

Fig. 1. block diagram of the GeForce FX

The GPU has 3 vertex and 8 fragment processors.

1.1. Vertex Processor

The GeForce FX has three independent vertex processors which fully comply (and even exceed) with the DX9 specification for vertex shaders 2.0 (fig. 2).

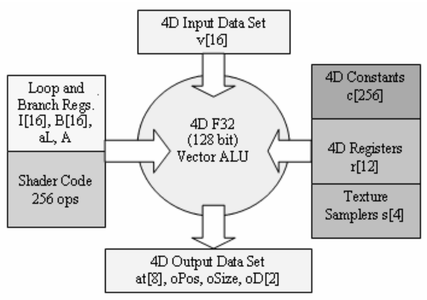
4D Input Data Set v[16]
Loop and Branch Regs. I[16], B[16], aL, A
Shader Code 256 ops
4D F32 (128 bit) Vector ALU
4D Constants c[256]
4D Registers r[12]
Texture Samplers s[4]
4D Output Data Set at[8], oPos, oSize, oD[2]

Fig. 2. Block diagram of vertex processor in DX9

Intermediate data are processed and stored in the floating-point format F32. At the input the shader's program has up to 16 4D vectors - the source geometrical data. At the output we have a vertex position in the screen coordinates, a pixel size if sprites are involved, 8 vectors of texture coordinates and 2 vectors of color values which are then interpolated (while triangles are rendered) for each pixel. After interpolation values of these vectors will get into the pixel shader as input parameters. Besides, there are 256 constant vectors assigned from outside and 12 temporary general-purpose registers used for storing intermediate data. There are also 4 special registers - samplers which let the vertex shader select values from textures for using textures as displacement maps and other similar effects.

From this point of view a vertex processor reminds any other general-purpose processor. A shader is a program which controls a vector ALU processing 4D vectors. A shader's program can be 256 ops long but it can contain loops and transitions. For organization of loops there are 16 integer registers of counters I which are accessible from the shader only for reading, i.e. they are constants assigned outside in an application. For conditional jumps there are 16 logic (one-bit) registers B. Again, they can't be changed from the shader. As a result, all jumps and loops are predetermined and can be controlled only from outside, from an application.

Also the set of instructions of vertex shaders in the GeForce FX was extended compared to the DX9, with normal trigonometric functions and conditional write and reorder instructions. It's interesting that realization of trigonometric functions in the GeForce FX is very quick - it takes the same time to calculate SIN or COS as a couple of additions. It seems that it operates with special matrix execution units together with big tables of constants.

Here are commands supported by the vertex processors of the GeForce FX:

- Add and multiply (ADD, DP3, DP4, DPH, MAD, MOV, SUB)
- Math (ABS, COS, EX2, EXP, FLR, FRC, LG2, LOG, RCP, RSQ, SIN)
- Set On (SEQ, SFL, SGR, SGT, SLE, SLT, SNE, STR)
- Branching (BRA, CAL, NOP)
- Address Registers (ARL, ARR)
- Graphics-oriented (DST, LIT)
- Minimum/maximum (MAX, MIN)

2. Pixel processors and texture units

The GeForce FX has eight independent pixel processors which fully comply with the DX9 specification for pixel shaders 2.0 (fig. 3).

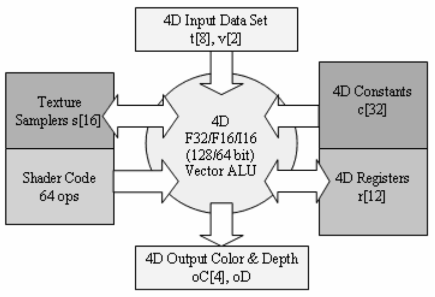

Fig. 3. Block diagram of pixel processor in DX9

At the input there are 8 texture coordinates interpolated across the triangle surface (with perspective correction) and two interpolated color vectors. Originally, these vectors were calculated for each triangle vertex, but to obtain their values for each pixel rendered by a pixel shader we have to interpolate them depending on a position of a given pixel relative to the vertices. From the programmer's point of view these vectors can contain anything, not just texture coordinates and color. The pixel shader will define how to use them.

At the output there are up to 4 different color values (each being a 4D vector) recorded into different output frame buffers and one depth value which we can change and record. There are 32 constant vectors and 12 temporary vector registers.

In course of execution of a shader GPU can fulfill texture fetching, up to 16 different textures are available (the depth of nesting of dependent fetches mustn't exceed 4). For texture fetching there is a special command which indicates where to send the result, from which texture (one of 16 registers of samplers) and according to which coordinates data are to be fetched. Contrary to the previous generation this is a normal command of a shader and can be used in any place in any order. But the number of commands of fetching is limited - although the total length of a shader can reach up to 96 instructions, the number of requests to textures mustn't exceed 32 and number of other instructions mustn't exceed 64.

Contrary to a vertex processor which always works with the F32 data format, a pixel processor supports three formats- F32, F16 and integer I12. The latter two formats are not just useful for compatibility with old shaders 1.x but also provide speed gains in calculations.

In the DX9 the system of commands of a pixel processor is similar to the system of commands of a vertex one:

- Add and multiply (ADD, LRP, DP3, DP4, LRP, MAD, MOV, SUB)
- Texturing (TEX, TXD, TXP)
- Partial Derivatives (DDX, DDY)
- Math (ABS, COS, EX2, EXP, FLR, FRC, LG2, LOG, NRM, POW, RCP,RSQ, SIN)
- Set On (SEQ, SFL, SGR, SGT, SLE, SLT, SNE, STR)
- Graphics-oriented (DST, LIT, RFL)
- Minimum/maximum (MAX, MIN)
- Macros (SINCOS, CRS)
- Pack (PK2H, PK2US, PK4B, PK4UB, PK4UBG, PK16)
- Unpack (UP2H, UP2US, UP4B, UP4UB, UP4UBG, UP16)
- Kill (KIL)

The performance of commands drops down twice while processing floating-point data compared to integer data (this is a pure computational performance without accounting for losses caused by increased data volumes). The pixel processor of the GeForce FX can execute up to two integer and one floating-point command or two texture access operations per clock — i.e. it acts as a superscalar processor in case of integer operations and reception of sampled texture values from texture units. From NV35 (GeForce FX 5900) the GPU can execute up to two floating-point command.

3. Programming GPU

The GPU is designed for operating on large continuous streams of vertex and fragment data. Vertices are points in 3-D space which definene graphics primitives like triangles, polygons, rectangles etc. These primitives are used to build up the geometry of the scene and define any 2-D or 3-D models to be displayed. In older hardware the geometry sent to the graphics card was static. If this geometry or model data had to be altered in some way, it had to be transformed on the CPU and the new vertices downloaded to the GPU. Vertex shaders were designed to allow more control over vertex transformation on the GPU itself. This has obvious benefits as it frees up the CPU for other processes and also eliminates the need to download the same model over and over again.

Textures are images which can be mapped onto any of the graphics primitives to add detail to a scene. For example a texture of skin can be mapped on to a model of a human to make it look more life-like. The texture mapping stage is after the vertex stage in the graphics pipeline. Fragments are the name given to the data in the pipeline before it gets output as pixels to the screen. Fragments are slightly different than pixels because there can be fragments which will occupy more than one pixel on the screen. At the texture stage the texture image is looked up for the correct color to add to the fragment before it is converted to a pixel(s). Similar to vertex shaders, previous generation hardware texture units were limited in the operations they could perform on fragments. Fragment shaders were introduced to allow much more control of how textures are applied to the fragments. Fragment shaders also tend to be more powerful than vertex shaders as they are usually operating on higher volumes of data. They can also perform memory reads as they look up values in textures. Pixels are finally rendered at end of the graphics pipeline in a conceptual device called the framebuffer.

Generally pixels rendered to the framebuffer are made visible on the screen to the user. However this framebuffer data can become corrupted if other windows on the screen are moved and overlap with the current window. This is because the same framebuffer, which is a piece of GPU memory, is shared for all applications on the desktop. To overcome this problem another useful feature of modern graphics hardware is exploited: off-screen rendering buffers called pixel buffers (Pbuffers). These Pbuffers reside in GPU memory and are similar to the framebuffer, except they are generated and controlled by the application. Also, the framebuffer does not benefit the increase in pixel depth and is limited to 32 bits per pixel. Pbuffers are necessary if a 128 bits per pixel rendering buffer and floating point computation are required. If further processing of the results generated is needed, the Pbuffers can be used as a texture and passed through the graphics pipeline again.

In order to use the GPU for general purpose computing the problem has to be mapped onto the GPUs programming architecture. In this case per-pixel operations on the GPU are required. The results are then read back to the CPU. To achieve this the images have to be downloaded as textures to the GPU for use in fragment programs. This involves rendering a rectangle and texture mapping the images to this rectangle. If this rectangle is the same size as the images then the fragments will correspond on a one to one basis with the pixels in the texture. Fragment programs can then be used to do the required calculations on the GPU, and results output to a Pbuffer.

For this work the cross platform OpenGL API was used for interfacing with the graphics hardware. The other main graphics API is Microsoft Direct X. Programming the GPU involves writing custom vertex and fragment programs to be executed by the vertex and fragment shaders.

One of the main bottlenecks when using the GPU for image processing is the time taken to read the data back from the graphics hardware to the CPU. The Accelerated Port Technology (AGP) standard is a means of transferring data quickly to the graphics hardware. It operates at a multiple of the PCI bus speed; AGP4x operates at 4 times the PCI bus speed and the latest AGP standard, AGP8x, operates at 8 times the PCI bus speed. However reading data back from the graphics hardware is limited by the standard PCI bus speed. With a PCI bus speed of 66 MHz the theoretical bandwidth available is approximately 260 MB/sec, whereas the AGP8x bus has a peak bandwidth of 2.1 GB/sec. In tests however a peak data read bandwidth of 180 MB/sec or 45 MPixels/sec at 32bits per pixel was achieved. For PAL resolution frames this gives a peak frame rate of 113 frames/sec for reading data from the graphics card. Downloading the image as a texture, doing the interpolation, and reading the data back resulted in 75 frames/sec at PAL resolution. Doing the same interpolation on the CPU achieved 24 frames/sec. A new PCI bus standard due out in 2004 called PCI Express, may help with this problem by providing greater bandwidth over the PCI bus.

CONCLUSIONS AND FUTURE WORK

Modern video processors delivers great computational power, which remains unused in standard applications. This allows use of the GPU as mathematical coprocessor operating on large continuous streams. Main difficulties are:

- low accuracy – 32 bit floating point number;
- limitation in shaders programs;
- low peak data read bandwidth;

Despite of those limitations use of the GPU as general purpose processor is increased in the last few years due to rapid development of new families of GPUs.

REFERENCES

1. R. Yang and G. Welch, *"Fast image segmentation and smoothing using commodity graphics hardware",* To appear in the journal of graphics tools, special issue on Hardware-Accelerated Rendering Techniques , 2003.
2. E. S. Larsen and D. McAllister, *"Fast matrix multiplies using graphics hardware"*,Supercomputing 2001 , November 2001.
3. K. Moreland and E. Angel, *"The FFT on a GPU"*, in Graphics Hardware 2003, July 2003.

ABOUT THE AUTHOR

dipl.-Ing. Dimitar A. Atanasov, Ph. D. student, Department of Computer Systems and Technologies, University of Gabrovo, Phone: +359 66 223 501, E-mail: d_atanasov@tugab.bg.