JASS 2009 – “Numerical Simulation - from Models to Software”:

GPU Computing

Andreas Barthels

March 29 – April 7, 2009

1 Introduction

The field of using Graphics Processing Hardware for General Purpose Computation (*GPGPU*) has gained a lot of attention within the last six months. This is mostly due to the graphics cards manufacturers releasing official software packages and increased marketing and public relations work done by the big players in the industry.

Also within the last 6 months, the specifications for the “Open Computation Language”[1] and DirectX 11[2] were released, which both pave the way for standardized usage of GPGPU.

The paper that you are reading accompanies my talk on GPU Computing that I held on JASS 2009 in St. Petersburg. Since the graphics industry is moving very fast, I will not detail too much this article because its relevance will decay within few months. The sole purpose of this paper is to phrase the slides of my talk in more detail which gave an insight into the GPGPU situation at that time.

2 GPU Hardware

After giving an introduction to the topic, my talk went on with describing GPU Hardware.

2.1 GPU Functionality

The functionality of a GPU is to speed up the process of rendering three dimensional models and scenes on a computer. These models typically are defined through a geometry consisting of polygons that is assigned one and up to many textures. The typical operation purpose of a GPU is the playing of video games or using rapid prototyping etc..

Textures can be processed in a multitude of formats. In the original meaning, textures are basically bitmaps defining color values for each texture element (*texel*). As texture elements, there can be integers and floating point numbers with a varying number of representational bits. Due to this flexibility in storage format, textures were started to be used for other things than just

color values in a bitmap sense. For example, textures can be used as normal maps which contain the normal vectors of a polygonal surface. These normal maps thus typically consist of floating point vectors for each texel.

With recent hardware extensions, the graphics cards became more and more flexible in working with their textures and polygonal models. Textures can be used for all kinds of things and can be used as the source as well as the target for computations. This already made the hardware usable for general purpose programming for graphics computing experts.

2.2 GPU–CPU Comparison

In order to achieve an immersive display of three dimensional scenery, the graphics cards manufacturers compete in making their products as fast as possible. This lead to GPUs becoming highly parallel high performance computing hardware.

While CPUs are designed for running operating systems and doing multi purpose (and comparably mildly parallel) operations, GPUs were designed with accelerating graphics instead. Since in the field of graphics processing, most operations are embarassingly parallel, current GPUs have hundreds of arithmetic logical units to do their computations in parallel. In that way GPUs offer a multiple of the peak computing power of CPUs while operating at a fraction of the clock speed.

One more thing that is important to mention is that the memory technology for performance oriented GPUs differ from that used in CPUs. Due to the improved electrical properties of omitting the concept of ram sockets and modules but soldering the chips directly onto the GPU board, the memory is capable of operating at a much higher clock speed. As a consequence, the memory performance of GPUs is again a multiple of that of CPUs and reaches easily the performance of the chaches of modern CPUs in terms of bandwidth.

3 GPU Computing Today

Besides the possibilities to do general purpose computing with GPUs through the typical graphics interfaces, a set of interfaces for non-experts was developed by different parties in the past years. All interfaces have the following concepts in common; they introduce a new datatype called *streams* and operations on streams (so-called *kernels*). Please note that streams basically are just a renaming of textures.

For an in-detail explanation of the programming languages / interfaces, I have to refer to the respective official documentation given in the references section.

3.1 BrookGPU

BrookGPU[3] was the first GPGPU framework. It was developed at Stanford and was published in the year 2003. It is basically an extension to the standard C programming language.

When working with kernels, it features customized reduction, scatter and gather operations.

3.2 OEM SDKs

Both AMD/ATI Stream SDK[4] and NVIDIA CUDA[5] more or less adopt the concept pioneered by BrookGPU.

Since AMD is also manufacturing processors, they have a slight advantage because what they offer is not only a C-style programming interface to GPUs but also math libraries that optimally use as many GPUs and CPUs as they detect in the system[6]. This is useful because one can utilize the same interface for tasks that are either more CPU or more GPU optimizable.

When it comes to comparing hardware, performance leadership and architectural designs can change every few months, so I will not discuss the current situation in this paper.

4 GPU Computing Example

As a live demo of GPU Computing, I showed a 2D Fluid Simulation developed by Dr. Jens Krüger as part of his PhD thesis[7]. The reason for this is that my notebook hardware is pretty old and the aforementioned SDKs do not work for it. The demo is written within a linear algebra framework based on GPU computing which was developed at the chair for computer graphics at the Technische Universität München. It uses a conjugate gradient solver to solve a sparse linear system of equations.

4.1 Formulization of the Water Surface Simulation

Denote u to be the height of the water surface at a certain point in time and space and let c be the wave speed. The following differential equation describes the evolution of the water surface height over time:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right)$$

Applying finite difference methods in discretizing space and time each equidistantally by Δt and h yields:

$$\frac{u_{i,j}^{t+1} - 2u_{i,j}^{t} + u_{i,j}^{t-1}}{\Delta t^2} = c^2 \left(\frac{u_{i+1,j}^{t} + u_{i-1,j}^{t} + u_{i,j+1}^{t} + u_{i,j-1}^{t} - 4u_{i,j}^{t}}{h^2} \right)$$

It can be shown that applying $u_{i,j}^{t} := 0.5 \cdot \left(u_{i,j}^{t+1} + u_{i,j}^{t} \right)$ to the right side of the equation leads to a solution of the problem that is always stable. Solving the problem is done by solving a system of linear equations which is given through:

$$(4\alpha + 2)\, u_{i,j}^{t+1} - \alpha \left(u_{i+1,j}^{t+1} + u_{i-1,j}^{t+1} + u_{i,j+1}^{t+1} + u_{i,j-1}^{t+1} \right) = d_{i,j}^{t}$$

where

$$d_{i,j}^{t} := \alpha \left(u_{i+1,j}^{t} + u_{i-1,j}^{t+1} + u_{i,j+1}^{t+1} + u_{i,j-1}^{t+1} \right) + (4 - 4\alpha)\, u_{i,j}^{t} - 2u_{i,j}^{t-1}$$

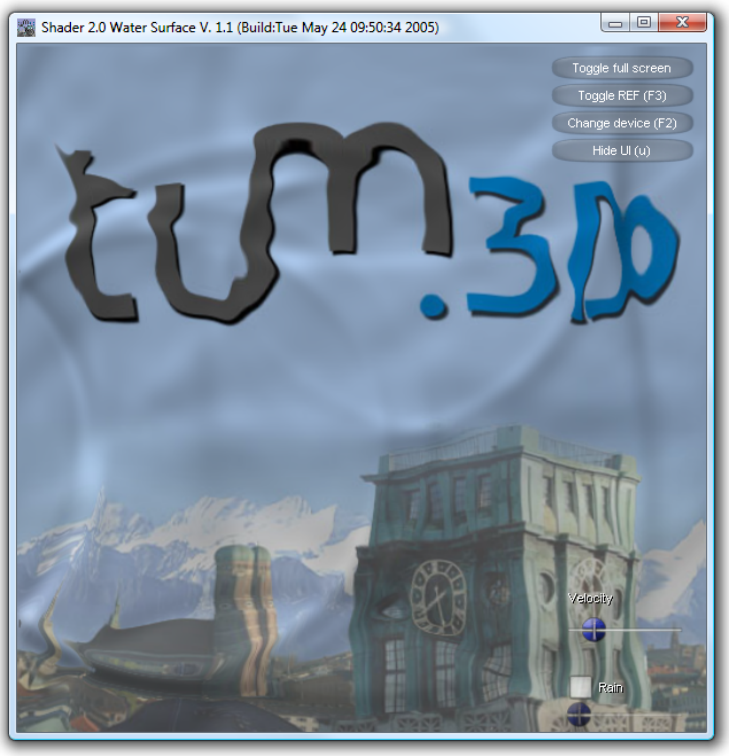

Figure 1: Water Surface Simulation by Dr. Jens Krüger

and

$$\alpha := \frac{\Delta t^2 \cdot c^2}{h^2}$$

The update step for a 3×3 system can be written as the following Matrix vector product:

$$\begin{pmatrix}
4\alpha+2 & -\alpha & & -\alpha & & & & & \\
-\alpha & 4\alpha+2 & -\alpha & & -\alpha & & & & \\
 & -\alpha & 4\alpha+2 & & & -\alpha & & & \\
-\alpha & & & 4\alpha+2 & -\alpha & & -\alpha & & \\
 & -\alpha & & -\alpha & 4\alpha+2 & -\alpha & & -\alpha & \\
 & & -\alpha & & -\alpha & 4\alpha+2 & & & -\alpha \\
 & & & -\alpha & & & 4\alpha+2 & -\alpha & \\
 & & & & -\alpha & & -\alpha & 4\alpha+2 & -\alpha \\
 & & & & & -\alpha & & -\alpha & 4\alpha+2
\end{pmatrix}
\begin{pmatrix} u_{0,0}^{t+1} \\ u_{1,0}^{t+1} \\ u_{2,0}^{t+1} \\ u_{0,1}^{t+1} \\ u_{1,1}^{t+1} \\ u_{2,1}^{t+1} \\ u_{0,2}^{t+1} \\ u_{1,2}^{t+1} \\ u_{2,2}^{t+1} \end{pmatrix}
=
\begin{pmatrix} d_{0,0}^{t} \\ d_{1,0}^{t} \\ d_{2,0}^{t} \\ d_{0,1}^{t} \\ d_{1,1}^{t} \\ d_{2,1}^{t} \\ d_{0,2}^{t} \\ d_{1,2}^{t} \\ d_{2,2}^{t} \end{pmatrix}$$

Please note that the boundary condition of zero waveheight around the modelled grid was used here.

For an in-detail discussion of the conjugate gradient solver and the employed sparse matrix storage schemes, I have to refer to the dissertation of Dr. Jens Krüger[7].

5 Outlook

As I already outlined in the introduction, the two most interesting upcoming things are DirectX11 and OpenCL.

DirectX is very interesting because, since it is only driven by a single entity, namely Microsoft, it is being pushed forward much faster than a standard that is agreed on in a huge working group like OpenCL with the Khronos group. Because of this, DirectX is what the hardware manufacturers focus on making their hardware compliant to.

Concerning GPGPU, DirectX 11 introduces what is called the "Compute Shader" (*CS*). In terms of flexibility and native hardware support the CS is pushing the GPGPU frontier a lot further. One feature of the CS is, e.g., object oriented programming natively supported by the GPU.

The advantage of OpenCL is that it is defined for heterogeneous computing (the same code runs on many hardware architectures). Besides that, OpenCL is just a flat C extension, like BrookGPU and the current OEM SDKs are.

6 Summary

GPUs feature high performance for little money. They are capable not only of doing numerical calculations in, e.g. simulations, but they are also very well suited for immediate visualization of the results. Both DirectX 11 and OpenCL are very promising, one in flexibility for using the GPU, the other for having a truely heterogeneous stream computing language that is supposed to run on a multitude of hardware archtitectures.

References

[1] The Khronos Group. Open Computing Language. www.khronos.org, 2008.

[2] Microsoft Developer Network. Direct3D 11 Technical Preview. www.microsoft.com, 2008.

[3] Ian Buck, Tim Foley, Daniel Horn, Jeremy Sugerman, Pat Hanrahan, Mike Houston, and Kayvon Fatahalian. Brook for GPUs. graphics.stanford.edu, 2003.

[4] AMD/ATI. ATI Stream SDK. developer.amd.com, 2009.

[5] NVIDIA. Compute Unified Device Architecture. www.nvidia.com/cuda, 2009.

[6] AMD/ATI. AMD Core Math Library for Graphic Processors. developer.amd.com, 2009.

[7] Jens Harald Krüger. *A GPU Framework for Interactive Simulation and Rendering of Fluid Effects*. Dissertation, Technische Universität München, ub.tum.de, 2006.