SPH BASED FLUID ANIMATION USING CUDA ENABLED GPU

Uday A. Nuli[1] and P. J. Kulkarni[2]

[1]Textile and Engineering Institute, Ichalkaranji, Maharashtra(INDIA)
uanuli@yahoo.com
[2]Walchand College of Engineering, Sangli, Maharashtra(INDIA)
pjk_walchand@rediffmail.com

ABSTRACT

Realistic Fluid Animation is an inherent part of special effects in Film and Gaming Industry. These animations are created through the simulation of highly compute intensive fluid model. The computations involved in execution of fluid model emphasize the need of high performance parallel system to achieve the real time animation. This paper primarily devoted to the formalization of parallel algorithms for fluid animation employing Smoothed Particle Hydrodynamics (SPH) model on Compute Unified Device Architecture (CUDA). We have demonstrated a considerable execution speedup on CUDA as compare to CPU. The speedup is further improved by reducing complexity of SPH computations from O(N ^{2}) to O(N) by utilizing spatial grid based particle neighbour lookup.

KEYWORDS
Particle Animation, Smoothed Particle Hydrodynamics, CUDA, SPH, GPU.

1. INTRODUCTION

Animation is changing its trend from traditional key-framed, non-realistic and offline to realistic, model driven, and real time animation. Animation of natural phenomena such as flood can be realistically created if the underline physicals laws for movement of fluid are used. Physically based animation is a technique that incorporate physical behavioural model of the object to estimate motion of the object. Since the motion is based on physical laws, the motion produced in the animation is realistic. However such animation needs huge computational power to solve the equations governing the motion. Hence real-time physically based animation is possible through involvement of suitable parallel architecture such as multi-core or computer cluster. Smoothed Particle Hydrodynamics (SPH) is a physically based fluid Model that treats fluid continuum as collection of particles. Motion of particles is governed by set of equations defined by SPH technique. Hence particle based animations based on SPH is appropriate for implementation on SIMD parallel architecture.

Over past few years, Graphics Processing Unit (GPU) has evolved from a fixed function graphics pipeline to a general purpose, many-core SIMD architecture. Although GPU architecture is inherently parallel since its invention, it was specifically limited to graphics functionality till NVIDIA introduced it as a Compute Unified Device Architecture (CUDA)[1][2]. CUDA is highly parallel, multithreaded, many-core processor with tremendous computational horsepower and very high memory bandwidth. Although many parallel algorithms are already written for many platforms before, these can not be applied to new platforms in its original form. Existing algorithms needs certain level of reformation to execute on new platform efficiently. This paper is focused on reformation and optimization of parallel algorithm for implementation of SPH based particle animation on NVIDIA CUDA platform.

DOI : 10.5121/ijcga.2012.2404

2. RELATED WORK

Fluid animations have a very long history. In past it has been implemented with various physical models, and hardware platforms. Jos Stam[3] has developed a Mesh-oriented solution for fluid animation. Nick Foster and Ron Fedkiw[4] derived full 3D solution for Navier-Stokes equations that produces realistic animation results. In addition to the basic method, the Lagrange equations of motion areused to place buoyant dynamic objects into a scene, and track the position of spray and foam during the animation process. Jos Stam[5] extended the basic Eulerian approach by approximating the flow equations in order to achieve near real time performance. He has also demonstrated various special effects of fluid with simple "C" code at typical frame rate of 4 to 7 minutes per frame. Although Eulerian approach was a popular scheme for fluid animation, it has few important drawbacks such as; it needs global pressure correction and has poor scalability [6]. Due to these drawbacks such schemes are unable to take benefits of recent parallel architectures.

Particle based methods are free from these limitations and hence are becoming more popular in fluid animation. Reeves [7] introduced the particle system which is then widely used to model the deformable bodies, clothes and water. He has demonstrated animation of fire and multicoloured fireworks. Particle System based animations are created with two approaches, one with motion defined by certain physical model and other by simple use of Newton's basic laws. R. A. Gingold and J.J Monaghan proposed "Smoothed Particle Hydrodynamics", a particle based model to simulate astrophysical phenomena [8] and later extended to simulate free-surface incompressible fluid flows [9]. This model was created for scientific analysis of fluid flow, carried out with few particles. Matthias Müller extended the basic SPH method for fluid simulation for interactive application [10] and designed a new SPH kernel. The first implementation of the SPH method totally on GPU was realized by T. Harada [11] using OpenGL APIs. T. Harada has demonstrated 60,000 particles fluid animation at 17 frames per second which is much faster as compared to CPU based SPH fluid animation. These papers clearly highlight the aptness of Smoothed Particle Hydrodynamics technique for SIMD parallel architecture to achieve realistic Particle based fluid animation.

3. SMOOTHED PARTICLE HYDRODYNAMICS

Smoothed Particle Hydrodynamics [8][9] is a mesh-free, Lagrangian, particle method for modelling Fluid flow. This technique is introduced by Lucy and Monaghan in 1977 for modelling astrophysics phenomena and later on extended for modelling fluid phenomena.

SPH integrates the hydrodynamic equations of motion on each particle in the Lagrangian formalism. Relevant physical quantities are computed for each particle as an interpolation of the values of the nearest neighbouring particles, and then particles move according to those values. The basic Navier-Stokes equations are transformed to equivalent particle equations. According to SPH, a scalar quantity $A_s(r)$ is interpolated at location r by a weighted sum of contributions from all particles. The basic interpolation formula used is:

$$A_s(r) = \sum_j m_j \frac{A_j}{\rho_j} W(r - r_j, h) \tag{1}$$

Where, $A_s(r)$ is the scalar property of Particle at position r, m_j the mass of J[th] particle at distance r_j from particle at r, ρ_j the mass-density of particle at location r_j, A_j the scalar property of particle at location r_j and W the Kernel Function[10] or the smoothing kernel.

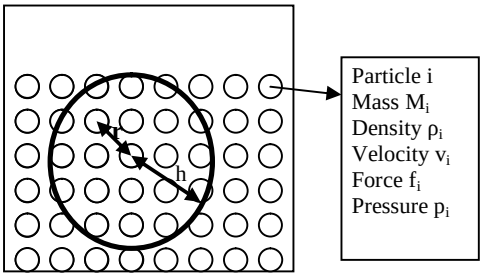

Figure 1. Representation of fluid as collection of particles.

The Mass-density of a particle is calculated by substituting density term in place of generic term $A_S(r)$ in equation 1. The equation for density $\rho_s(r)$ terms is as follows [[10]]:

$$\rho_s(r) = \sum_j m_j W(r - r_j, h) \tag{2}$$

The pressure exerted on a particle due to other particles is derived from ideal gas law. The pressure is computed using following equation:

$$P = k(\rho - \rho_0) \tag{3}$$

Where P is the pressure exerted on the particle, k the stiffness constant of gas, ρ the mass density of the particle at time t in simulation and ρ_0 the mass density of the particle at rest condition. Every particle is influenced by viscous and pressure forces. These forces are computed using SPH formulations as:

$$f_i^{pressure} = -\sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla W\left(r_i - r_j, h\right) \tag{4}$$

$$f_i^{viscosity} = \mu \sum_j m_j \frac{v_j - v_i}{\rho_j} \nabla^2 W\left(r_i - r_j, h\right) \tag{5}$$

Where $f_i^{pressure}$ is the force due to pressure and $f_i^{viscosity}$ the force due to viscosity on i^{th} particle exerted by other particles; p_i and p_j are the pressure, v_i and v_j the velocities, r_i and r_j the position of i^{th} and j^{th} particle; m_j is the mass and ρ_j is the density of the j^{th} particle.

Particle acceleration velocity and position updates are carried out using equations specified by the Matthias Müller[10].

4. COMPUTE UNIFIED DEVICE ARCHITECTURE

Over the past few years, Graphics Processing Unit (GPU) has evolved from a fixed function graphics pipeline to a general purpose, many-core SIMD architecture. GPU, itself has originated from the need of parallelism to render complex graphics scenes at real time. Even though the architecture of graphics card, since its inventions, was inherently parallel, the functionality of card was not programmable. Due to the need of realistic and customizable rendering requirement, GPU architecture was changed from fixed pipeline to limited programmable pipeline. The pipeline functionality is allowed to change through small shader programs. Although General Purpose programming on GPU using graphics language, termed as GPGPU is a complex task,

still many researcher has put enormous efforts in order to achieve performance due to parallelization.

In the year 2006, NVIDIA unveiled a new GPU architecture with general purpose parallel programming model and instruction set, called as CUDA – Compute Unified Device Architecture [1][2]. This architecture is a variant of basic SIMD model and ideally suitable for data parallel computation with high arithmetic intensity. CUDA is both hardware as well as programming platform. Basic component of CUDA program execution is a Thread. Threads are organised as one dimensional or two dimensional array in a block and blocks are organised in a grid. CUDA can launch thousands of thread simultaneously on available physical cores. A block of thread is typically launched on a multiprocessor in hardware. Along with thread, CUDA is characterised by its memory organization. Registers, shared memory and external global memory are the most important types of CUDA memory that has significant impact on execution performance. Registers and shared memory are local to each multiprocessor and its availability is limited as compared to global memory. Access latency of global memory and bank conflict in shared memory are the key limiting factors in execution performance of an algorithm on CUDA.

5. SYSTEM ARCHITECTURE

Primary task in any physically based particle animation model is to estimate the motion of each particle and comprises of computation of particle spatial position in every time step. Motion estimation based on SPH involves computation of every particle's mass density, pressure and force exerted due to neighbouring particle within a distance of smoothing radius. Hence determination of neighbour particles has complexity of $O(N^2)$ for all particles. This complexity can be reduced down to O(N) level, employing spatial grid based neighbour search technique. These huge periodic computations justify the need of CUDA parallel architecture in order to complete it in real time. However certain part of animation needs to be carried out on CPU due to execution constraints of GPU.

A simple one Thread per particle allocation strategy is used to decompose the total execution cost among CUDA threads. Threads are organized as lineaer array in a block. The total number of blocks is decided by the total threads and hence total particles used for animation. Structure of array data structure is employed to store particle data in GPU global memory. Data is cached in shared memory as per execution requirement to avoid access latency issue of global memory and to keep all executing threads busy. Unavailability of data to executing treads results into performance degradation due to stalling of thread execution. Each independent computation is carried out as separate kernel execution on thread. Basic algorithm for particle animation is as follows:

Basic Algorithm for SPH Based Particle System Animation

a) Initialize and setup particle system.
b) Render Particles.
c) Transfer data to GPU Memory.
d) Construct spatial grid for particles.
e) For each particle:
    i) Calculate density.
    ii) Calculate Pressure exerted on the particle due to its neighbours.
    iii) Calculate net Force on Particle due to inter-particle pressure.
    iv) Add external force to Particle.
    v) Find Particle Acceleration.

vi) Find next particle position.
vii) Update Particle position.

f) Transfer data from GPU to CPU Memory.
g) Invoke OpenGL Commands to Display the Particles at updated position.
h) Repeat from step d.

5.1. System Block Diagram

The animation process needs to be partitioned into two phases. During first phase the initialization of particle system, SPH parameters and GPU memory allocation is carried out. Second phase of animation loop is dedicated to particle motion estimation and comprise of density, pressure force and displacement computations. First phase execution entirely takes place on CPU and second phase on GPU co-ordinated by CPU. Separate parallel kernels are written for spatial grid construction, Density, pressure, force and displacement computation. These kernels are called sequentially from CPU as indicated in Figure 2. Particle rendering can be implemented on entirely CPU or GPU based on sophistication requirements of animated result.

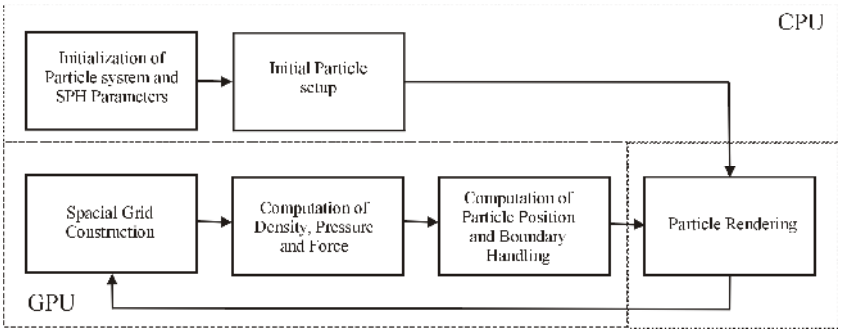

Figure 2. System Block Diagram

5.2. System Initialization

Particle based fluid animation starts with parameter setup for the physical model followed by the initial fluid particle setup. Physical parameters such as density of fluid, volume of fluid, number of particles constituting the fluid volume, fluid viscosity, fluid boundary specifications, gravitational force, stiffness constant and SPH smoothing radius required to set prior to start of animation loop. Particle density is computed based on fluid density, fluid volume, and Total number of particles used for animation. Initial particle setup decides fluid's physical appearance at the beginning of animation and involves setting of individual particle position. SPH parameters are more critical and require careful tuning to achieve visually plausible result from animation. SPH smoothing radius is set to 2.5 to 5 times the particle diameter and is decided by the total number of particles used for animation.

6. PARTICLE MOTION ESTIMATION USING CUDA

Motion estimation computations, described in section 5, can be significantly speedup through implementation on parallel architecture like CUDA. Following are the steps required to calculate particle position at next time step:

1. CUDA Initialization.
2. Spatial Grid Construction.

3. Particle density and force computation.
4. Particle Velocity and displacement computation.

6.1. CUDA Initialization

CUDA initialization involves memory allocation for GPU device memory, Thread count and layout estimation, Initial Data transfer from CPU to GPU memory. Since GPU can not allocate its own memory, it needs to be allocated from CPU. Memory allocation on GPU is necessary to store particle data such as particle position, velocity, physical parameters, and SPH parameters. Structure of Array is the most appropriate data structure to store all particle data as it causes coalesced memory access that reduces memory fetch cycles. Initialization ends with transfer of Particle data from CPU memory to GPU memory.

6.2. Spatial Grid Construction

Particles are organized in a 3-dimensional spatial grid to reduce particle neighbour search cost from $O(N^2)$ to O(N). Dynamic grid construction with variable number of particles per grid cells is the key requirements for grid construction approach. Both requirements can be satisfied by sorting based grid construction approach. The steps for spatial grid construction are as listed below [12][13][14].

1. Assign Cell ID to each particle based on its location.
2. Grouping Particles according to grid cell.
3. Prepare cell table consisting Cell ID, Particle index for cell header, and particle count.
4. Prepare cell neighbour table consisting 26 neighbours for each cell.

In the process of identification of cell ID of a particle, first structure of grid cell identifier is finalized. Size of grid cell ID is fixed to 32 bits and its structure is as shown below:

00	Y component	Z component	X component
2 bits	10 bits	10 bits	10 bits

Figure 3: Structure of grid cell identifier

This grid cell ID structure can accommodate a grid of size 1024 x 1024 x 1024. Each of cell ID components is calculated by dividing its respective location component by grid cell size. Assigning each particle to one CUDA thread cell ID computation can be carried out it O(1) step. Particle grouping according to cell ID is carried out by sorting all particles on cell ID. Parallel radix sort algorithm is used for sorting due to its best execution performance on CUDA as compared to other sorting techniques. Cell Table is used to store details of spatial grids such as the cell Id, total particles per cell, index of first particle in the cell. A modified Prefix sum algorithm is used to find total particles per cell and first particle in the cell. Although cell table stores information about all the cells in grid, it does not provide information about neighbour cells of a cell. Cell neighbour table is constructed to extract information about neighbour cell of a cell in O(1) step avoiding sequential search on cell table. Every entry in cell neighbour table consists of the cell ID and its 26 neighbour cells. A parallel binary search algorithm is used to construct cell neighbour table. Cell neighbour table reduces the complexity of particle neighbour search from $O(N^2)$ to O(N) since neighbours of a particle can present only within its 26 neighbour grid cell.

6.3. Particle Density, Force and Displacement computation

Density of a particle is computed using equation (2) with kernel specified by Müller [10]. This equation is evaluated on per particle basis and only neighbouring particle within a distance of smoothing radius considered for calculation. Since each particle is assigned to a CUDA thread, every thread performs same density computation.

Algorithm for density and pressure computation of a particle on a CUDA Thread

a) Get current position of the particle.
b) Set density to zero.
c) For each neighbour cell of current particle including its own cell.

i) Select a particle from the selected cell.
ii) Find the distance between the two particles.
iii) If distance is less than smoothing radius, calculate density and accumulate it in global density.
iv) Repeat steps from i for all particles in selected cell.

d) Compute pressure exerted on particle using equation (3).

Above algorithm can be extended to evaluate per particle force computations using equations described in section 3. Force is resolved in three dimensions to calculate resultant motion of a particle in 3D space. Velocities calculated in previous iteration are used to compute the viscous forces on particle. Aggregate force on particle is summation of force due to Pressure, viscosity and gravitation. This aggregate force is used to compute acceleration and velocity of a particle along with resultant direction. Euler's integration technique is used to compute displacement of a particle using its velocity.

7. FLUID RENDERING

Particles of fluid are rendered using OpenGL functions. A simplest approach is to render each particle as solid sphere. This approach is suitable for demonstration of fluid animation technique and not for professional animation. For Professional animation fluid surface is reconstructed from particle using surface construction algorithm such as Marching Cube.

8. RESULTS

Main motive behind employment of CUDA enable GPU for fluid animation is to achieve real-time performance. The result show here clearly demonstrates considerable speedup in execution of every stage of animation. This particle animation has been carried out on a computer with Intel® Core™2 Duo CPU E7500 @ 2.93GHz with 2GB of RAM and NVIDIA GTX 280 GPU card. Timings are measured with NVIDIA CUDA Profiler and averaged.

Table 1: Computation Time and Speedup for 10000 Particles (time in ms):

Operations	CPU	GPU	Optimized GPU	Speed up	Optimized speedup
Pressure Computation	578.43	7.40	1.457	78.16	397.00
Force Computation	674.08	12.37	1.976	54.49	341.13
Displacement	0.27	1.12	1.115	0.24	0.24

Computation					
Point Sprite Rendering	0.35	0.35	0.35	-	-
Total Time and Speedup	1253.13	21.24	4.898	59.00	255.84

Table 2: Computation Time and Speedup for 50000 Particles (time in ms):

Operations	**CPU**	**GPU**	**Optimized GPU**	**Speed up**	**Optimized speedup**
Pressure Computation	12725.74	71.26	2.653	178.58	4796.74
Force Computation	14672.61	190.66	5.092	76.96	2881.50
Displacement Computation	1.41	1.09	1.091	1.29	1.29
Point Sprite Rendering	1.33	1.30	1.301	-	-
Total Time and Speedup	27401.09	264.31	10.137	103.67	2703.08

Table 3: Computation Time and Speedup for 100000 Particles (time in ms):

Operations	**CPU**	**GPU**	**Optimized GPU**	**Speed up**	**Optimized speedup**
Pressure Computation	58249.08	269.22	4.825	216.36	12072.35
Force Computation	65184.26	939.86	12.743	69.36	5115.29
Displacement Computation	2.94	1.14	1.132	2.58	2.59
Point Sprite Rendering	2.60	2.54	2.541	-	-
Total Time and Speedup	123438.9	1212.76	21.241	101.78	5811.35

9. CONCLUSION AND FUTURE WORK

This Paper is focused on Smoothed Particle Hydrodynamics (SPH), a relatively new Fluid Dynamics technique to simulate motion of fluid particles. SPH is a particle based parallelizable technique hence more suitable for GPU based architecture. Equations of SPH expose more arithmetic intensity in computation; hence the animation's computational part can be executed in real time. However the major hurdle in SPH based animation is the tuning of SPH parameters like smoothing radius and rest density. A small change in any of these parameter results into almost explosion of fluid. Since no proper guidelines are available for tuning SPH, considerable amount of efforts are required to find these parameters using brute force approach.

Parallel Algorithm Design is the most critical issue in any application development for CUDA. Even though most of the algorithms discussed in the paper are already designed for some of earlier parallel architectures, it is not possible to adopt them without modification for CUDA. Also the typical asymptotic complexity equations are not appropriate for indicating performance of algorithm on CUDA. The asymptotic definition of complexity comments on the number of discrete steps executed by algorithm and not on the arithmetic intensity of computations per step. Hence asymptotic complexity equations do not represent true performance of an algorithm on

CUDA. Besides complexity of algorithm, other factors that contribute significantly in deciding execution performance of any parallel algorithm are Memory access latency, Memory access pattern, inter-processor communication overheads, synchronization overheads, and the degree of parallelism.

This work can be extended for huge quantity fluid animation that demands more memory space than available on present GPU. For animation of large quantity of fluid, the employment of alone GPU is also insufficient. Hence development of framework for animation on cluster of GPU can be considered as next possible approach. Also Surface quality of fluid can still be improved by optimizing ray tracing for interactive rate rendering on GPU.

ACKNOWLEDGEMENTS

We extend our sincere thanks to NVIDIA Corporation, USA, for sponsoring a GTX 280 GPU card to us.

REFERENCES

[1] NVIDIA, "NVIDIA CUDA C, Programming Guide, version 4.2", http://developer.download. nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf, visited on 31 July 2012.

[2] NVIDIA, "NVIDIA CUDA C Best Practices Guide, version 4.1", http://developer.download. nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Best_Practices_Guide.pdf, visited on 31 July 2012.

[3] Stam Jos. "Stable fluids", ACM SIGGRAPH 99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co., New York, NY, pp121-128, 1999.

[4] Nick Foster, Ronald Fedkiw, "Practical Animation of Liquids", ACM SIGGRAPH 2001, pp21-30, 2001.

[5] Stam Jos. "Real-Time Fluid Dynamics for Games", Proceedings of the Game Developer Conference, 2003.

[6] Jie Tan and XuBo Yang, "Physically-based fluid animation: A survey", Science in China Series F: Information Sciences, Science China Press, co-published with SpringerLink, Volume 52, pp723-740, May 2009.

[7] Reeves W.T. "Particle Systems-A technique for modeling a class of fuzzy objects", ACM Transactions on Graphics, Vol. 2, Issue No. 2, pp91-108, April 1983.

[8] R. A. Gingold and J. J. Monaghan, "Smoothed particle hydrodynamics: theory and application to non-spherical stars",Monthly Notices of the Royal Astronomical Society, 181:375 - 398, 1977.

[9] J. J. Monaghan, "Simulating free surface flows with SPH", Journal of Computational Physics, Volume 110, Issue 2, pp399-406, 1994.

[10] Matthias Müller, David Charypar and Markus Gross, "Particle-Based Fluid Simulation for Interactive Applications", Proceedings of the 2003 ACM SIGGRAPH, pp154 – 159, 2003.

[11] Takahiro Harada, Seiichi Koshizuka, Yoichiro Kawaguchi, "Smoothed Particle Hydrodynamics on GPUs", Proceedings of Computer Graphics International, pp63-70, 2007.

[12] Simon Green, "Particle Simulation using CUDA", http://www.dps.uibk.ac.at/~cosenza/ teaching/ gpu/nv_particles.pdf, visited on 31 July 2012.

[13] Nadathur Satish, Mark Harris Michael Garland, "Designing Efficient Sorting Algorithms for Manycore GPUs", Proceedings of 23rd IEEE International Parallel and Distributed Processing Symposium, pp1-10, 2009.

[14] M. Harris, S. Sengupta, and J. D. Owens. "Parallel prefix sum (scan) with CUDA",GPU Gems 3, chapter 39, pp851-876. Addison Wesley, Aug. 2007.