

DETC2012-71236

**USING GPUS FOR REALTIME PREDICTION OF OPTICAL FORCES ON
MICROSPHERE ENSEMBLES**

Sujal Bista

Graduate Student
Inst. for Advanced Computer Studies
Department of Computer Science
University of Maryland
College Park, Maryland 20742
Email: sujal@cs.umd.edu

Sagar Chowdhury

Research Assistant
Dept. of Mechanical Engineering
University of Maryland
College Park, Maryland 20742
Email: sagar353@umd.edu

Satyandra K. Gupta

Professor, Fellow of ASME
Inst. for Systems Research
Department of Mechanical Engineering
University of Maryland
College Park, Maryland 20742
Email: skgupta@umd.edu

Amitabh Varshney

Professor
Inst. for Advanced Computer Studies
Department of Computer Science
University of Maryland
College Park, Maryland 20742
Email: varshney@cs.umd.edu

ABSTRACT

Laser beams can be used to create optical traps that can hold and transport small particles. Optical trapping has been used in a number of applications ranging from prototyping at the microscale to biological cell manipulation. Successfully using optical tweezers requires predicting optical forces on the particle being trapped and transported. Reasonably accurate theory and computational models exist for predicting optical forces on a single particle in the close vicinity of a Gaussian laser beam. However, in practice the workspace includes multiple particles that are manipulated using individual optical traps. It has been experimentally shown that the presence of a particle can cast a shadow on a nearby particle and hence affect the optical forces acting on it. Computing optical forces in the presence of shadows in real-time is not feasible on CPUs. In this paper, we introduce a ray-tracing-based application optimized for GPUs to calculate forces exerted by the laser beams on microparticle ensembles in

an optical tweezers system. When evaluating the force exerted by a laser beam on 32 interacting particles, our GPU-based application is able to get a 66-fold speed up compared to a single core CPU implementation of traditional Ashkin's approach and a 10-fold speedup over its single core CPU-based counterpart.

1 Introduction

An optical tweezers system is a scientific instrument that uses light to manipulate micron-sized particles. Ashkin first introduced the system in 1986 [1]. Since then scientists have been using this system to manipulate and study microparticles such as dielectric spheres, cells, DNA, bacteria, and virus. They are often used in creating assembly of micro- and nano-scaled components to make a functional device due to the extensive range of positioning and orienting capabilities of the system [2]. Additionally, optical tweezers systems can be used to manipulate cells in a controlled manner without causing them any damage [3,4,5].

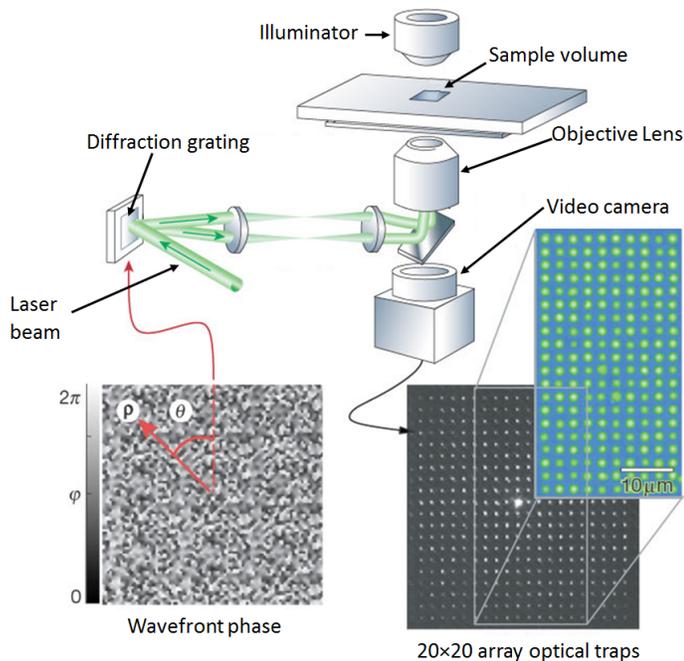


FIGURE 1. In an optical tweezer setup, a Gaussian laser beam is converged by a convex lens (objective lens of a microscope) to a focal point which is used for trapping microparticles. To create multiple optical traps, the laser beam is split into multiple-beams using a diffraction grating. Diagram courtesy of [6].

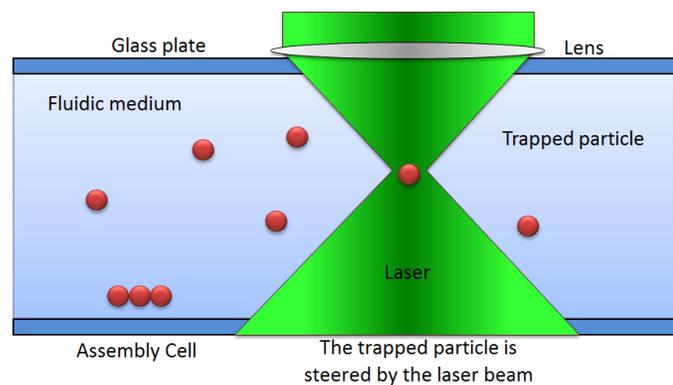


FIGURE 2. An illustration of the optical tweezers system. A laser beam with a Gaussian-based intensity distribution is converged into a focal point with the help of a convex lens. The figure shows laser beam trapping microparticles at the focal point.

The optical tweezers system is composed of a very powerful laser beam that has a Gaussian-based intensity distribution and a convex lens that focuses the laser beam onto the focal point as shown in Figure 1. This focused laser beam is used to move microparticles that are submerged in the fluid. When the micropar-

ticles are bigger than the wavelength of the light used in the laser beam, the ray optics model is used to define the behavior of the optical tweezers system [1]. The laser beam is decomposed into a bundle of rays, each carrying a photon. When these rays interact with the microparticles, they get reflected and refracted. As each ray consists of a photon, the change in the momentum gives rise to the optical force that is exerted on the microparticles. This force is used by the optical tweezers system to trap and move the microparticles. Figure 2 shows an illustration of a microparticle getting trapped. Figure 3 shows a series of images captured through the imaging device in the optical tweezers system in our lab showing a microparticle (silicon bead) getting trapped. An optical trap is placed close to the microparticle which exerts a strong gradient force that pulls the particle towards the focal point.

Simulation plays an important role in understanding optical tweezers system. To manipulate microparticles precisely, the force exerted by the laser has to be known; this is studied by performing simulations. The force calculation is a very computationally intensive task due to the Brownian motion of the microparticles suspended in the fluid which requires the time step of the simulation to be smaller than 10^{-6} sec. The popular approach to overcome the timing constraint is to use a pre-computed force look-up table to study simulation as done by Banerjee *et al.* [2]. Reasonably accurate theory and computational models exist for predicting optical forces on a single particle in the close vicinity of the Gaussian laser beam. However, in practice the workspace includes multiple particles that are manipulated using individual optical traps. Experiments have shown that the presence of a particle can cast a shadow on a neighboring particle and hence affect the optical forces acting on it. When microparticles are closely placed under several laser beams, the rays get reflected and refracted which introduces secondary forces that affect the trapping. This behavior is often referred as shadowing phenomenon. It occasionally causes trapped microparticles to escape or cause an unwanted microparticle to jump into the trap. Studying this phenomenon is vital for scientists who are using optical tweezers system for micro assembly or path planning [7, 8, 9].

In this paper, we present an optimized GPU-based ray tracing application to calculate the force exerted by the laser beams on the microparticles to study the shadowing phenomenon. Our program is capable of computing the forces exerted by laser beams on multiple microparticles (up to 32) at more than 100Hz which is a higher rate than that of a typical optical device used for imaging/monitoring the microparticles. We are able to calculate the interaction between the lasers and several microparticles and make it possible to study the shadowing phenomenon vital for understanding optical trapping. When evaluating the force exerted by a laser beam on 32 interacting particles, our GPU-based application is able to get approximately a 66 times speed up compared to a single core CPU implementation of traditional Ashkin's approach and a 10 times faster than its single

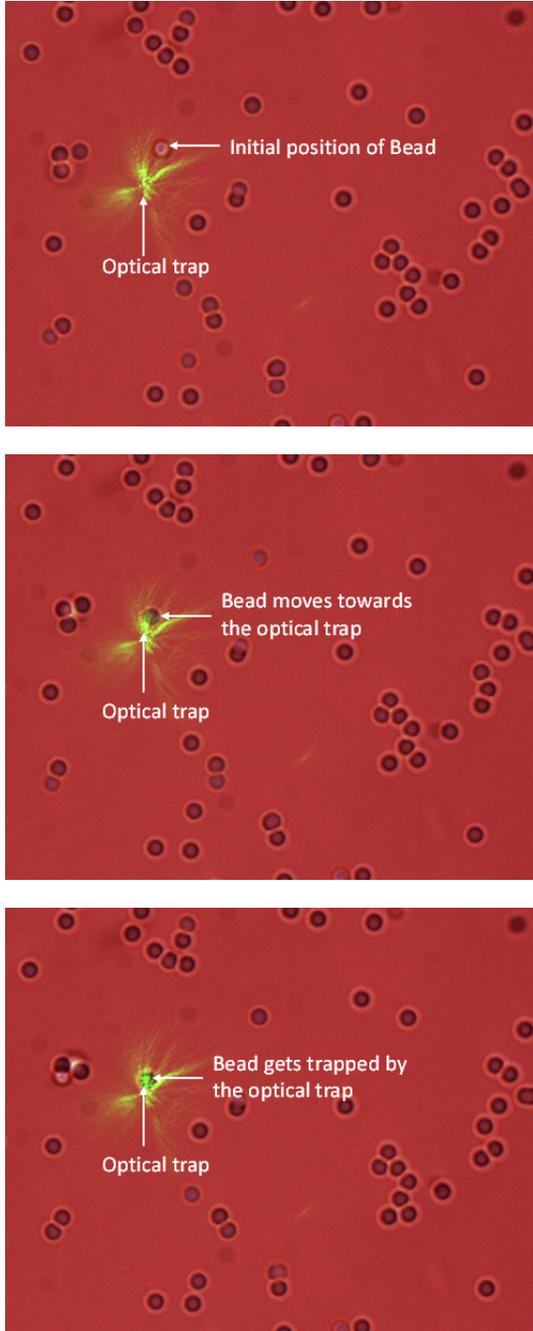


FIGURE 3. When an optical trap is placed close to a microparticle, it pulls the particle towards the focal point. The images above captured using the imaging device in the optical tweezers system show a microparticle moving into a trap.

core CPU-based counterpart. In this paper, we also talk about several choices we made while developing this application and compare them in terms of performance and precision. We also

present an alternative way to calculate force exerted by the laser that exploits coherence of the mapping from incident ray to the x, y, z components of force and the transmitted ray by using non-negative matrix factorization (NMF). This method can be useful when computation of the path a ray travels within the microparticle cannot be easily computed by simple sphere-object intersections (possibly caused by uneven density of the microparticle). We also present an instance where the shadowing effect drastically changes the amount of force applied on a microparticle.

2 Related Work

Powerful lasers are used to manipulate microparticles in an optical tweezers system. This was first introduced by Ashkin *et al.* [10] where they used a single-beam gradient force to trap micro- and nano-sized dielectric particles. Ashkin later introduced a geometric ray-optic model that is used to compute trapping forces created by a laser acting on microparticles much larger than the wavelength of light [1]. Though the equation Ashkin used is fairly optimized as it computes scattering and gradient forces based only on the incident angle and the radial position of the ray, it only works with rigid spherical objects and cannot be used directly to study interaction between several beams and microparticles. Our work is focused in calculating forces using our GPU-based ray tracing algorithm which provides both speed and flexibility needed to study shadowing phenomenon.

One of the biggest challenges in simulating optical tweezers system is performing calculation of force exerted by the laser quickly. Since the microparticles are influenced by the Brownian motion, simulations have to be done at a time scale much smaller than a microsecond. Banerjee *et al.* [2] introduced a framework where offline simulation is used to pre-compute data at discrete points and is later used to perform fast and accurate calculation of dynamic trapping probability estimates at any arbitrary point in 3D. This approach cannot accurately compute the effect when laser interacts with several nanoparticles. We focus on calculating the force quickly on dynamic microparticles so that the interactions of laser with several microparticles can be accurately simulated to study the shadowing phenomenon.

Bianchi and Leonardo [11] use GPUs to perform optical manipulation using holograms in real-time. They achieved speedups of 45X and 350X over CPU on their super position algorithm (SR) and Gerchberg-Saxton weighted algorithm (GSW) respectively. The speedup helped them to perform interactive micromanipulation. Balijepalli *et al.* [12] and Patro *et al.* [13] have used GPUs to compute trapping probabilities and have gotten significant speedups. We also carry out our calculation on the GPU. However our work is focused on computing the force exerted by the laser beams and we perform ray tracing to compute the force as the laser interacts with several microparticles.

Sraj *et al.* [14] used dynamic ray tracing to induce optical force on the surface of the deformable cell from which they cal-

culate stress distribution. Rather than using the rigid spheres as an approximated shape of the cell, they perform force calculation on the actual cell. They show that the shape of the cell strongly influences how the optical force stretches and deforms them. They also highlight that the applied optical forces change drastically when the cells are deformed. We focus our study on reducing the amount of time required to compute the exerted force. We perform our calculation on rigid microparticles and study how optical forces change when laser interacts with several nanoparticles that are closely interacting with each other.

Zhou *et al.* [15] have introduced a force calculating model that uses ray tracing based on spatial analytic geometry. Some of our ray tracing is based on their work but we perform GPU-based optimization and calculate interaction of laser beams with multiple particles quickly which is needed to study the shadowing phenomenon. We also provide an alternative way of computing the forces using NMF.

Using GPUs to accelerate computationally expensive algorithms is gaining a strong interest in the scientific and gaming community. Early work done by Harris *et al.* [16] used GPUs to perform visual simulation of fluids, clouds, and smoke. They mapped some basic operators (like heat and Laplace) on the GPU and used these operators to accelerate the simulation. They performed their calculation on the GPU using programmable shaders before general languages for GPU like CUDA, DirectX Compute, and OpenCL became prominent. Considerable advancements in physically-based simulation have been made recently due to their application in games and graphics [17]. In particular, fluid simulations on GPUs have gained significant momentum [18, 19, 20, 21]. Recently, Phillips *et al.* [22] used a cluster of GPUs to accelerate solver for 2D compressible Euler equation and MBFLO solvers. Using a cluster of 16 GPUs they achieve speedups of 496X and 88X on their Euler and MBFLO solvers. The trend of using GPUs to accelerate existing algorithms is growing. In our work, we use GPU-based acceleration of ray tracing to compute the force exerted by the optical tweezers on the microparticles.

Carr *et al.* [23] made a very persuasive case for the use of GPUs for computing ray-triangle intersections fairly efficiently by using pixel shaders. Purcell *et al.* [24] mapped the complete ray tracing algorithm to the GPUs, using different pixel shaders for creating rays, traversing rays, intersecting rays with triangles, and illumination calculations. Another well-known ray tracing engine Optix from NVIDIA uses GPUs for speed. Our ray tracing program is similar in nature but rather than calculating color for each pixel we compute the force exerted by the laser beam on the microparticles and perform integration. Also, the density of rays and the paths taken by the rays used in our calculation are different from the ones used by typical ray tracing program that uses a pinhole camera model.

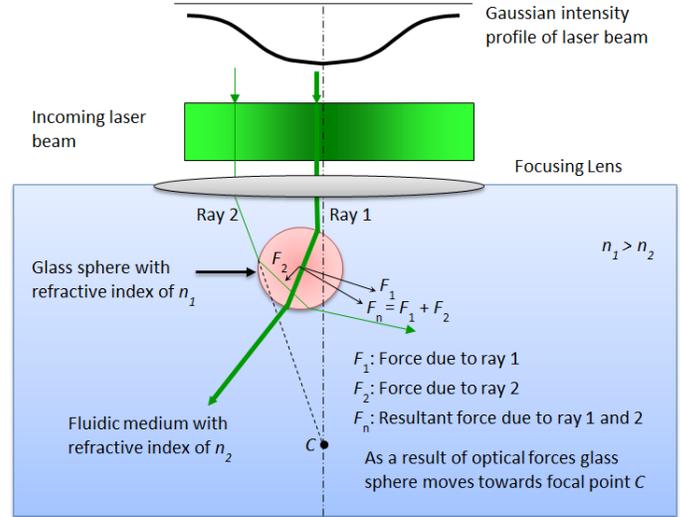


FIGURE 4. A diagram showing the simplified ray-optics model for calculating the force. The incident ray is diverted from its original path when it interacts with the microparticle. This causes the ray to change its momentum. When the ray changes momentum due to the microparticle, equal and opposite force is applied to the microparticle.

3 Approach

A simplified ray-optics model for calculating the force is shown in Figure 4. Due to the change in the index of refraction between the fluid and the microparticle, the incident ray is diverted from its original path as it goes through the microparticle. This causes the ray to change its momentum. When the ray changes momentum, equal and opposite force is applied to the microparticle. We calculate force contributed by each ray for each particle. After the contribution of each ray is calculated, integration is done to find the total force. We divide up the entire force calculation process into several steps described below.

3.1 Ray-Object Intersection

The first step is to compute fast ray-object intersection. We compute ray-object intersection on the GPU using a 3D-grid-based data structure. We choose uniform grid-based data structure over BSP, kDTree, and Octree because creating, updating, and ray traversing operation is faster when a uniform 3D grid is used as it allows constant time access to the cells and ray-traversal can be carried out using the efficient 3D-DDA algorithm [25, 24]. In our application, the grid-based data structure is created on the CPU and sent to the GPU memory every frame. In the optical tweezers system, the number of particles monitored in the experiments is often less than 64, so we create and update the data structure on the CPU. Once the grid data is transferred to the GPU, we perform ray-object intersection using a GPU Kernel. The ray-object intersection is highly parallelizable and a huge

performance gain is achieved by using many cores of a GPU as compared to a single core of a CPU. At first we considered using Optix for ray tracing. However we soon realized that we needed a ray tracer that was more flexible to meet our memory mapping needs, easily integrable with the remaining steps in the force calculating pipeline, and incurred less overhead. As Optix is a general-purpose ray tracing software made for rendering, we decided to develop our own dedicated GPU-based program that is highly specialized for force calculation.

The laser beams are decomposed into R_N rays. Each ray is mapped to a thread in the CUDA kernel and all R_N threads are launched at the same time. We save the attributes (such as position, radius) of the microparticles and the 3D grid data in the global GPU memory whereas the properties of the 3D grid and the laser beam are saved in the constant GPU memory as shown in Figure 5. Every thread traces the path of a ray independently.

3.2 Force Calculation

When a ray intersects a microparticle, we compute the reflected, refracted, and the final transmitted ray by performing basic intersections. Using these rays and the properties of the microparticle, we first compute only the magnitude of the scattering and the gradient force using the equation described by Ashkin [1].

$$F_s = \frac{n_1 P}{c} \left\{ 1 + R \cos(2\theta) - \frac{T^2 [\cos(2\theta - 2r) + R \cos(2\theta)]}{1 + R^2 + 2R \cos(2r)} \right\}$$

$$F_g = \frac{n_1 P}{c} \left\{ R \sin(2\theta) - \frac{T^2 [\sin(2\theta - 2r) + R \sin(2\theta)]}{1 + R^2 + 2R \cos(2r)} \right\}$$

where n_1 is the index of refraction of the incident medium, c is the speed of light, P is the incident power of the ray, R is the Fresnel reflection coefficient, T is the Fresnel transmission coefficient, θ is the angle of incidence, and r is the angle of refraction. Then we compute the direction of the scattering and the gradient forces directly from the vectors obtained from ray tracing. The direction of the ray is the same as the scattering direction. For the gradient direction, we use the scattering direction's orthogonal component that lies on the plane formed by the center of the particle, the point where the ray intersects the particle, and the intersection of the ray with the horizontal plane as described in [1, 15]. Now using the computed magnitude and the direction, we find the scattering and gradient forces. These forces are combined to calculate the total force exerted by the ray. The total force is then saved in the GPU memory. The transmitted ray is further traced to find the intersection of the ray with other particles and the steps described above are repeated as needed.

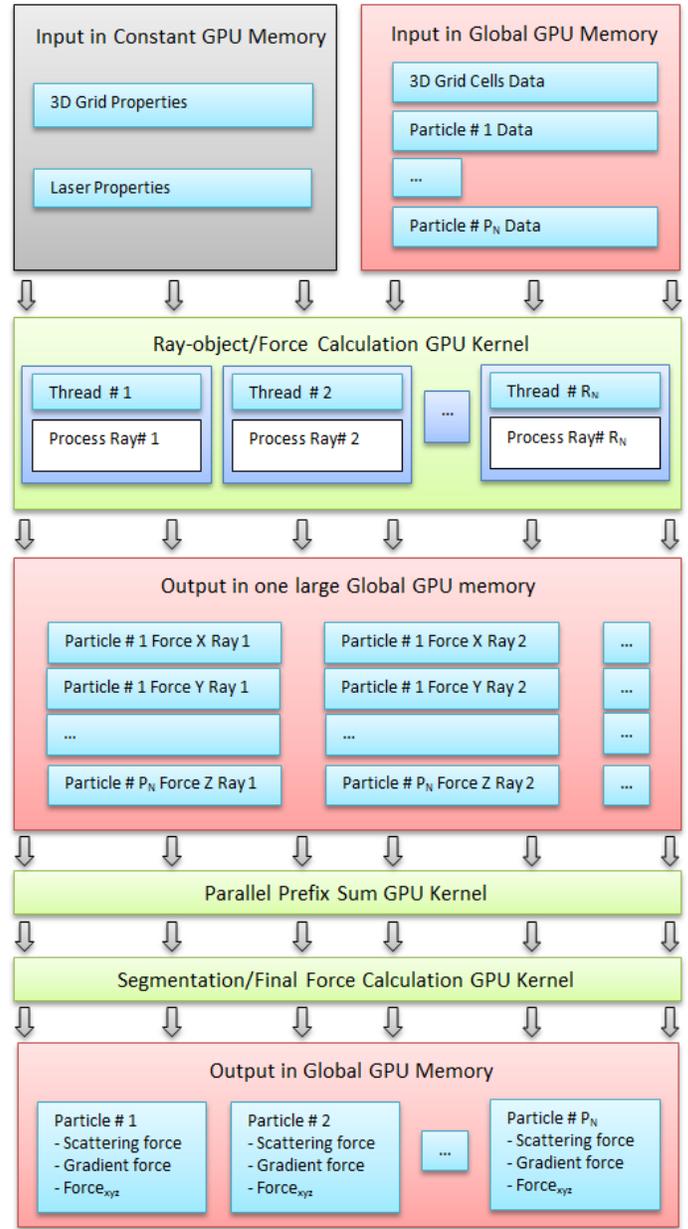


FIGURE 5. An overview of the GPU pipeline. The properties of the laser and the 3D grid are saved into the constant GPU memory whereas the properties of the particles and the 3D grid cells are saved in the global GPU memory. These data are used by the first GPU kernel that performs ray-object intersection and force per ray calculation. The output is written to a large global memory array. We then perform a parallel-prefix sum at the output. As the parallel-prefix sum adds up all the components together, Segmentation/Final force calculating kernel finds the proper segment boundaries for each component and subtracts necessary amount from the boundaries to compute the final result.

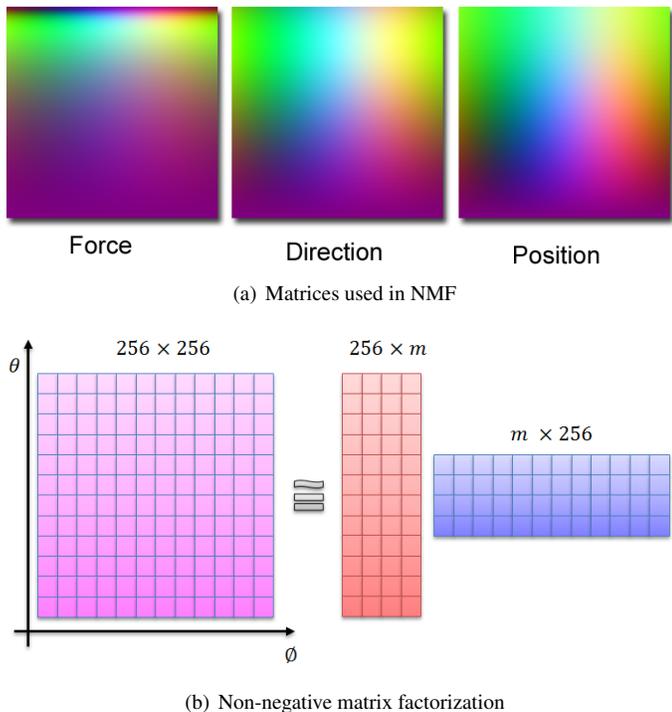


FIGURE 6. Pictorial view of the matrices that map discretized representation of incident ray angles to the force applied to the microparticle, the direction of the transmitted ray, and the position of the transmitted ray. The mapping is highly coherent which allows NMF to efficiently factorize each component of the matrix into two compact sized outer product matrices. Value of m used in our experiments is 4.

3.3 Force Calculation using Non-Negative Matrix Factorization

As an alternative to calculating the force independently for each ray, we also use non-negative matrix factorization (NMF) to take advantage of the coherence between the input rays, the force exerted, and the transmitted rays. The idea is to rotate the ray-object intersection point so that it lines up with the pole of the microparticle. Then we represent the ray by its spherical coordinates. We discretize the angles and compute the force exerted and the outgoing transmitted ray for each discrete set of angles. By doing this, we have a look-up map that is used to convert the incident ray angle to the exerted force, outgoing rays position, and direction directly. This look-up map can be very large and is made for each component of force and the outgoing ray. When we created the mapping, we observed that the mapping of the exerted force and the transmitted rays are all very coherent and have mostly low-frequency data as shown in Figure 6(a). To take advantage of this coherence, we use non-negative matrix factorization to factorize each mapping matrix into two compact matrices. Lawrence *et al.* [26] have applied NMF to decompose

the re-parameterized BRDF matrix that uses the half angle formulation. Here, we use NMF to compact the mapping matrix as shown in Figure 6(b) and save the dominant representation.

Once the matrices are saved, computing the force exerted by a ray is straightforward. When a ray intersects a sphere, we first compute the intersection position and the incident direction. We rotate the ray so the intersection position aligns with the pole of the microparticles. We then convert the incident ray into spherical coordinates. Using these angles, we multiply the appropriate rows and columns of the matrices generated by the NMF to get the mapping for each component of the force and the transmitted ray. Once we have the components, we apply inverse of the rotation that we applied to the incident ray to then calculate final values.

By using this alternative method to compute the force, we can take advantage of the coherence of the mapping. However, this method suffers from performance and precision issues compared to the ray tracing approach. The performance loss is caused by the matrix multiplication done to rotate the vectors and the cost of multiplying a row and a column to get values for each component of the force and the transmitted ray. The performance results are shown in Table 1. The precision loss is created by discretization of the input angles and the compact factorization created by NMF. Due to these reasons, we mainly use regular Ashkin's equation to calculate force using ray tracing when micro-sphere is used. However we believe that the NMF method can be useful when the microparticle has an uneven density which makes computation of the path a ray travels within the particle both difficult and computationally expensive. Such cases arise when computing the force applied to cells or micro-spheres with uneven density.

3.4 Force Integration

Once the CUDA kernel to compute the force for each ray is completed, we compute the net force applied on each particle by performing integration of the force field over the surface of the microparticle as done by Banerjee *et al.* [2]. We also perform this calculation on the GPU to save data transfer latency. At this stage of the pipeline, we have the force exerted by each ray in the GPU memory. Each component of the force for each particle is grouped and saved in a different part of a single large memory array. For example, all the scattering force exerted on a particle is saved in the first block followed by the gradient force and then followed by the entries of the second particle as shown in Figure 5. We perform a parallel-prefix sum on this large array once. This will scan all the components for the force for all particles together. Since the number of particles and the individual values of the force components used in the simulation are not very high, we do not suffer overflow error while performing the parallel-prefix sum of all the components together. We then execute another CUDA kernel that segments and outputs the final

Method	Number of Rays					
	8^2	16^2	32^2	64^2	128^2	256^2
Ashkin (Float)	0.0759	0.3558	1.2708	5.0548	20.2793	81.7446
Ashkin (Double)	0.0762	0.3705	1.3399	5.3316	21.5276	86.5138
CPU Ray (Float)	0.0807	0.3389	1.4369	5.4946	22.1243	88.9347
CPU Ray (Double)	0.0852	0.3529	1.4268	5.7644	22.8563	92.5199
GPU NMF (Float)	0.9592	0.9589	0.9826	1.1923	2.0615	5.4888
GPU Ray (Float)	0.7132	0.8745	0.8337	0.9007	1.2058	2.3813

TABLE 1. The time in seconds taken by the various methods to compute total force exerted on a single microparticle performed 5000 times at different locations.

force contribution for each particle by subtracting appropriate entries from the segment boundaries of each component as shown in Figure 7. This final step performs extremely well on the GPU because the output of the previous step is very big so transferring the data and calculating the final result on the CPU will cause a delay. By calculating the final force contribution on the GPU, we only need to read back the several components per particle.

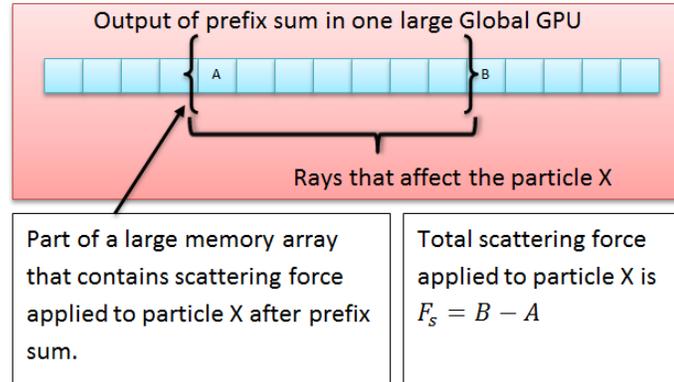


FIGURE 7. The final force contribution for each particle is calculated by subtracting values from the segment boundaries of an array that contains the result of the parallel-prefix sum. In this figure, we show how the final value of the scattering force is computed for a particle.

4 Results and Discussion

We have implemented our system in C++. We use the CUDA API for the GPU-based ray tracing. For all of our experiments, we use Windows 7 64-bit machine with Intel I5-750 2.66 GHz processor, NVIDIA GeForce 470 GTX GPU, and 8 GB of RAM.

4.1 Performance Comparison

We first show the performance gains achieved by using our GPU-based method. In our first set of experiments, we use rigid microparticles and record the amount of time it takes to compute the force. Our performance results are shown in Table 1 and Table 3. We compare the timings of various methods: Ashkin’s traditional, CPU-based ray tracing, GPU-based method that uses NMF, and GPU-based ray tracing methods using single and double precision floating-point arithmetic. For the first experiment, we perform force calculations on a single microparticle 5000 times placed in different locations around the focal point of the laser beam. We also vary the number of rays that are used to describe the laser beam. As shown in Table 1, when only one microparticle and 256^2 rays are used to represent the laser beam, the GPU-based force calculation is about 34 times faster than Ashkin’s method.

Some of the methods we use perform computation on the CPU and some use the GPU. We compare precision between various methods against CPU-based Ashkin’s method computed using equal number of rays and double precision floating-point arithmetic. We perform several comparisons by varying the number of rays to represent the laser beam. The results are shown in Table 2. In general, the relative error decreases as the number of rays increases. For NMF based computation, the relative error decreases at first and then fluctuates slightly as we increase the number of rays. This is because we discretized input angles while creating the mapping table. Due to this, increasing the number of rays while keeping the size of the mapping table constant, can increase the amount of error. For regular computations, 32^2 is an ideal number of rays to use to represent the laser as both the relative error and the computation cost are low.

For the second experiment, we performed force calculations using a laser beam and 32 interacting microparticles computed 5000 times placed in different locations. The ray tracing methods can capture the interaction of a laser with multiple particles while Ashkin’s traditional method can only capture interaction of the

laser with a particle at a time ignoring the shadowing effects. For the second experiment, we also show the performance difference triggered by the use of a data structure while doing ray tracing. As shown in Table 3, GPU-based force calculation that uses grid based data structure is about a 66 times faster than traditional Ashkin’s method and about 10 times faster than its CPU-based ray tracing analog when 256^2 rays are used to represent the laser beam.

When inspected carefully, using a 3D grid causes a slight delay when the numbers of rays or particles are low. As shown in Figure 8, when the numbers of rays or particles increases, the 3D grid performs better than the brute-force ray tracing method. This is generally because of the overhead of creating, updating, and transferring the grid to the GPU. As the performance depends on the number of particles, we allow the user to select any desired method.

4.2 Shadowing Phenomenon

We perform simple experiments to show the shadowing phenomenon. In the traditional ray-tracing community, the phenomenon we are simulating would be called as the second and higher-order refractions. However, since this is referred to as the *shadowing phenomenon* by the optical tweezers community, this is the term we shall use here.

We use two microparticles for these experiments. The first microparticle is moving in a path. The second microparticle is stationary and is being gripped by two laser beams with one focal point exactly above and the other below the microparticle. The rays that are incident on the second microparticle get refracted which can influence the number of rays that interact with the first microparticle. Thus the presence of the second microparticle causes a change in the amount of force being applied to the first microparticle. We show this change by performing simulations.

In the first experiment, we first perform the simulation using a single silicon bead of size 5 microns. Three Gaussian laser

Method	Number of Rays						
	8^2	16^2	32^2	64^2	128^2	256^2	512^2
GPU NMF (Float)	0.0068	0.0047	0.0034	0.0028	0.0035	0.0025	0.0032
CPU Ray (Double)	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
CPU Ray (Float)	0.0005	0.0001	0.0001	0.0001	0.0002	0.0001	0.0001
GPU Ray (Float)	0.0005	0.0006	0.0005	0.0005	0.0005	0.0005	0.0005

TABLE 2. Here we show the comparison of precision between various methods rounded up to the nearest 4 digits. We take Ashkin’s method as the reference and compute the relative error to compare other methods with an equal number of rays. As the number of rays increase, the relative error decreases in general but the computational cost increases.

Method	Number of Rays						
	8^2	16^2	32^2	64^2	128^2	256^2	
Ashkin (Float)	1.8877	7.7762	31.5119	128.1370	515.1390	2081.6000	
Ashkin (Double)	1.7971	7.7572	32.0977	129.2160	519.8880	2101.7000	
CPU Ray (Float)	0.2951	1.2400	5.1456	21.4900	86.4165	346.2620	
CPU Ray (Double)	0.3103	1.3025	5.9534	23.8178	95.1179	379.2980	
CPU Ray with 3D Grid (Double)	0.3831	1.3404	5.7862	22.8523	90.7224	360.8080	
GPU NMF (Float)	1.3050	2.0442	3.5816	9.1022	30.7739	116.5450	
GPU Ray (Float)	1.2649	1.6148	1.9824	3.7576	9.9187	33.3911	
GPU Ray with 3D Grid (Float)	1.8859	1.8621	2.2662	3.6953	9.4589	31.5070	

TABLE 3. The time taken (in seconds) by the various methods to compute total force exerted by a laser beam on 32 interacting microparticles computed 5000 times at different locations. It is interesting to note that when the number of rays is low, brute-force ray tracing is faster than the ray tracing method that uses a 3D grid data structure. This is due to the cost of creating and maintaining the data structure.

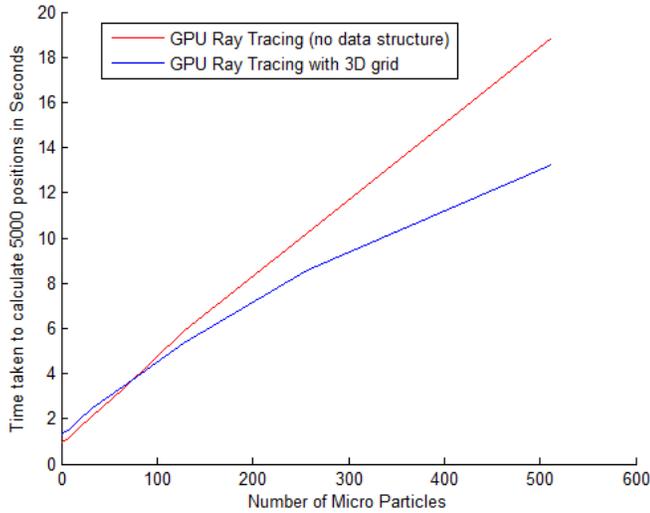


FIGURE 8. Here we show the time taken to compute the force exerted by a laser beam containing 32 rays 5000 times on a varying number of particles. We compare brute-force GPU ray tracing against GPU ray tracing with a 3D grid. As the number of particles increases, the use of a 3D grid data structure shows a clear advantage.

beams each focused at locations $(0.0, 0.0, 0.0)$, $(-1.0, 7.5, 0.0)$, and $(-1.0, 2.5, 0.0)$ are used. A bead is placed at $(0.0, -4.0, 0.0)$ and it slowly moves to $(0.0, 0.0, 0.0)$. In Figure 9, we show the force experienced by the bead as it goes from $(0.0, -4.0, 0.0)$ to $(0.0, 0.0, 0.0)$.

Now to show the shadowing phenomenon, we add an extra bead at location $(-1.0, 5.5, 0.0)$ in the same setup described above. This bead acts like a lens and changes the direction of the rays from the lasers. This causes the first bead to experience force from secondary rays. We compute the force experienced by the bead as it goes from $(0.0, -4.0, 0.0)$ to $(0.0, 0.0, 0.0)$ when the shadowing phenomenon is occurring. In Figure 9, we show the difference in the amount of force experienced by the first microparticle. This change adds instability and weakens the traps. Now again, we do the similar experiment but this time move the bead from $(-4.0, 0.0, 0.0)$ to $(0.0, 0.0, 0.0)$. Figure 10 shows the result of force calculation with and without the shadow phenomenon.

In both experiments, shadowing effects changes force applied significantly. This can change the behavior of the optical traps. Experimentally validating the results of simulations is challenging. There is no direct way to measure force. The force needs to be inferred from the observed motion. This requires a high speed image capture, accounting for the Brownian motion, and accounting for image blurring due to motion in the z-direction. We are currently in the process of designing experiments to record particle trajectories in the presence and absence

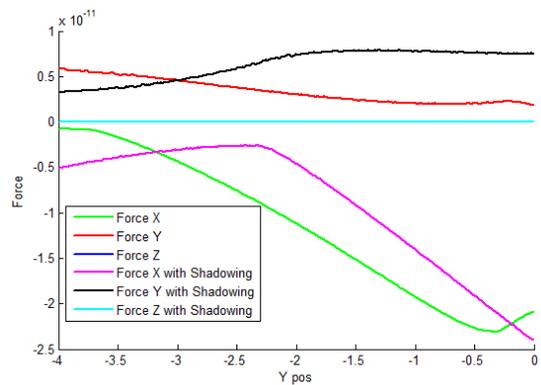
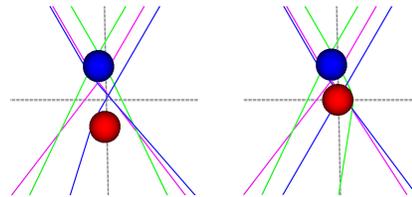
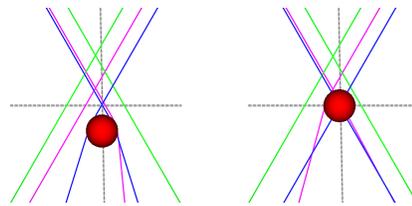
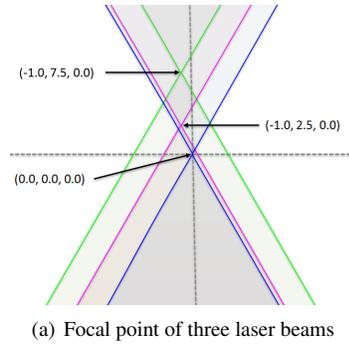


FIGURE 9. An illustration of the shadowing phenomenon. Figure (a) shows the focal point of three laser beams at location $(0.0, 0.0, 0.0)$, $(-1.0, 7.5, 0.0)$, and $(-1.0, 2.5, 0.0)$. Figure (b) shows the movement of a single particle from $(0.0, -4.0, 0.0)$ to $(0.0, 0.0, 0.0)$. The Figure (c) shows the movement of same particle when second particle is present at location $(-1.0, 5.5, 0.0)$. Finally, Figure (d) shows the difference in force experienced by the first bead caused by the shadowing phenomenon.

of shadowing phenomena.

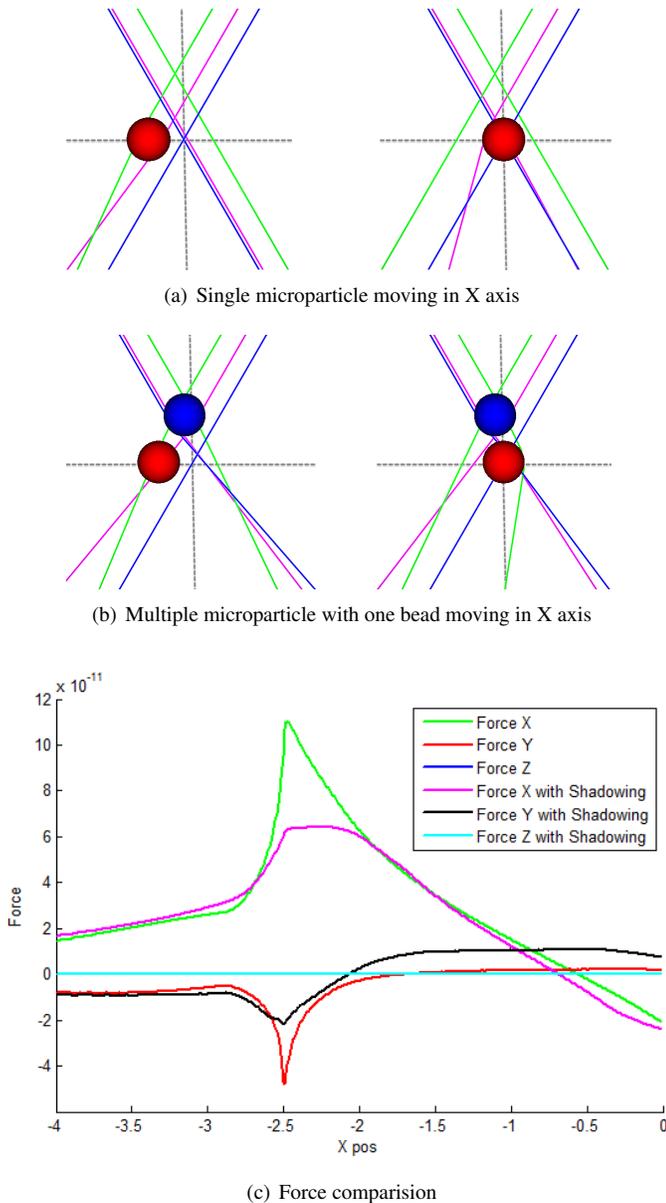


FIGURE 10. An illustration of the shadowing phenomenon similar to the previous figure. Figure (a) shows the movement of a single particle from $(-4.0, 0.0, 0.0)$ to $(0.0, 0.0, 0.0)$. The Figure (b) shows the movement of same particle when second particle is present at location $(-1.0, 5.5, 0.0)$. Finally, Figure (c) shows the difference in force experienced by the first bead caused by the shadowing phenomenon.

5 Conclusion and Future Work

The GPU-based application we presented in this paper computes the forces when laser beams interact with multiple microparticles and allow a scientist to study the shadowing phenomenon. Studying these phenomenon in real-time is vital as it allows efficient planning required for trapping and manipulating microparticles. When evaluating the force exerted by a laser beam on 32 interacting particles, our GPU-based application is able to get approximately a 66-fold speed up compared to the single core CPU implementation of traditional Ashkin’s approach and 10-fold speedup over its single core CPU-based counterpart. We also present an alternative way to calculate the force exerted by the laser that exploits coherence of the mapping from incident ray to the components of force and the transmitted ray by using NMF.

In future we plan to perform experimental investigation to validate our computational model by performing tests on scenarios that can be validated experimentally. Currently every time step is computed independently. Computing the force over a few time steps by taking account of changes might provide further speedup.

6 Acknowledgment

This work has been supported in part by the NSF grants: CCF 04-29753, CNS 04-03313, CCF 05-41120 and CMMI 08-35572. We also gratefully acknowledge the support provided by the NVIDIA CUDA Center of Excellence award to the University of Maryland. Any opinions, findings, conclusions, or recommendations expressed in this article are those of the authors and do not necessarily reflect the views of the research sponsors.

REFERENCES

- [1] Ashkin, A., 1992. “Forces of a single-beam gradient laser trap on a dielectric sphere in the ray optics regime”. *Bio-physical Journal*, **61**, Feb., pp. 569–582.
- [2] Banerjee, A. G., Balijepalli, A., Gupta, S. K., and LeBrun, T. W., 2009. “Generating Simplified Trapping Probability Models From Simulation of Optical Tweezers System”. *Journal of Computing and Information Science in Engineering*, **9**, p. 021003.
- [3] Koss, B., Chowdhury, S., Aabo, T., Losert, W., and Gupta, S. K., 2011. “Indirect optical gripping with triplet traps”. *J. Opt. Soc. America B*, **28**(5), Apr., pp. 982–985.
- [4] Banerjee, A. G., Chowdhury, S., Losert, W., and Gupta, S. K., 2011. “Survey on indirect optical manipulation of cells, nucleic acids, and motor proteins”. *J. Biomed. Opt.*, **16**(5), May.
- [5] Chowdhury, S., Svec, P., Wang, C., Losert, W., and Gupta, S. K., 2012. “Robust gripper synthesis for indirect manipu-

- lation of cells using holographic optical tweezers”. In IEEE Int. Conf. Intell. Robot. Autom.
- [6] Grier, D. G., 2003. “A revolution in optical manipulation”. *Nature*, **424**, Aug., pp. 810–816.
- [7] Banerjee, A. G., Pomerance, A., Losert, W., and Gupta, S. K., 2010. “Developing a Stochastic Dynamic Programming Framework for Optical Tweezer-Based Automated Particle Transport Operations”. *IEEE Trans. Autom. Sci. Eng.*, **7**(2), Apr., pp. 218–227.
- [8] Banerjee, A. G., Chowdhury, S., Losert, W., and Gupta, S. K., 2011. “Real-time path planning for coordinated transport of multiple particles using optical tweezers”. *IEEE Trans. Autom. Sci. Eng.* Accepted for publication.
- [9] Chowdhury, S., Svec, P., Wang, C., Seale, K., Wikswo, J. P., Losert, W., and Gupta, S. K., 2011. “Investigation of automated cell manipulation in optical tweezers-assisted microfluidic chamber using simulations”. In Proc. ASME Int. Des. Eng. Tech. Conf. & Comp. Inf. Eng. Conf.
- [10] Ashkin, A., Dziedzic, J. M., Bjorkholm, J. E., and Chu, S., 1986. “Observation of a single-beam gradient force optical trap for dielectric particles”. *Optics Letters*, **11**(5), May, pp. 288–290.
- [11] Bianchi, S., and Leonardo, R. D., 2010. “Real-time optical micro-manipulation using optimized holograms generated on the GPU”. *Computer Physics Communications*, **181**(8), pp. 1444–1448.
- [12] Baliyepalli, A., LeBrun, T., and Gupta, S. K., 2010. “Stochastic Simulations With Graphics Hardware: Characterization of Accuracy and Performance”. *Journal of Computing and Information Science in Engineering*, **10**, p. 011010.
- [13] Patro, R., Dickerson, J. P., Bista, S., Gupta, S. K., and Varshney, A., 2012. “Speeding Up Particle Trajectory Simulations under Moving Force Fields using GPUs”. *ASME Journal of Computing and Information Science in Engineering*. Accepted for publication.
- [14] Sraj, I., Szatmary, A. C., Marr, D. W. M., and Eggleton, C. D., 2010. “Dynamic ray tracing for modeling optical cell manipulation”. *Opt. Express*, **18**(16), Aug, pp. 16702–16714.
- [15] Zhou, J.-H., Ren, H.-L., Cai, J., and Li, Y.-M., 2008. “Ray-tracing methodology: application of spatial analytic geometry in the ray-optic model of optical tweezers”. *Applied Optics*, **47**.
- [16] Harris, M. J., Coombe, G., Scheuermann, T., and Lasstra, A., 2002. “Physically-based visual simulation on graphics hardware”. In Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, HWWS '02, Eurographics Association, pp. 109–118.
- [17] Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A. E., and Purcell, T., 2007. “A survey of general-purpose computation on graphics hardware”. *Computer Graphics Forum*, **26**(1), pp. 80–113.
- [18] Harris, M., 2005. “Fast fluid dynamics simulation on the GPU”. In SIGGRAPH '05: ACM SIGGRAPH 2005 Courses, ACM, p. 220.
- [19] Li, W., Wei, X., and Kaufman, A. E., 2003. “Implementing lattice boltzmann computation on graphics hardware”. *The Visual Computer*, **19**(7-8), pp. 444–456.
- [20] Liu, Y., Liu, X., and Wu, E., 2004. “Real-time 3D fluid simulation on GPU with complex obstacles”. In Pacific Conference on Computer Graphics and Applications, IEEE Computer Society, pp. 247–256.
- [21] Wei, X., Zhao, Y., Fan, Z., Li, W., Qiu, F., Yoakum-Stover, S., and Kaufman, A. E., 2004. “Lattice-based flow field modeling”. *IEEE Transactions on Visualization and Computer Graphics*, **10**(6), pp. 719–729.
- [22] Phillips, E. H., Zhang, Y., Davis, R. L., and Owens, J. D., 2009. “Rapid aerodynamic performance prediction on a cluster of graphics processing units”. In AIAA Aerospace Sciences Meeting, no. AIAA 2009-565.
- [23] Carr, N. A., Hoberock, J., Crane, K., and Hart, J. C., 2006. “Fast GPU ray tracing of dynamic meshes using geometry images”. In Graphics Interface, Canadian Human-Computer Communications Society, pp. 203–209.
- [24] Purcell, T. J., Buck, I., Mark, W. R., and Hanrahan, P., 2002. “Ray tracing on programmable graphics hardware”. *ACM Transactions on Graphics*, **21**(3), July, pp. 703–712.
- [25] Fujimoto, A., Tanaka, T., and Iwata, K., 1986. “Arts: Accelerated ray-tracing system”. *IEEE Computer Graphics and Applications*, **6**, pp. 16–26.
- [26] Lawrence, J., Rusinkiewicz, S., and Ramamoorthi, R., 2004. “Efficient BRDF importance sampling using a factored representation”. *ACM Transactions on Graphics*, **23**, pp. 496–505.
- [27] James, G., 2001. *Operations for hardware-accelerated procedural texture animation*. Charles River Media, pp. 497–509.
- [28] Hagen, T. R., Lie, K.-A., and Natvig, J. R., 2006. “Solving the Euler equations on graphics processing units.”. In International Conference on Computational Science (4), pp. 220–227.
- [29] Juba, D., and Varshney, A., 2008. “Parallel stochastic measurement of molecular surface area”. *Journal of Molecular Graphics and Modelling*, **27 No. 1**, August, pp. 82 – 87.
- [30] Heidrich, W., Westermann, R., Seidel, H.-P., and Ertl, T., 1999. “Applications of pixel textures in visualization and realistic image synthesis”. In Symposium on Interactive 3D graphics, ACM, pp. 127–134.