GPU Methods for Real-Time Haptic Interaction with 3D Fluids

Meng Yang[*][†], Jingwan Lu[‡][†], Alla Safonova[†], and Katherine J. Kuchenbecker[§]
[*]Microsoft Corporation, Redmond, WA, USA
[†]Computer and Information Science, University of Pennsylvania, Philadelphia, PA, USA
[‡]Computer Science and Mathematics, Hong Kong University of Science and Technology, Kowloon, Hong Kong
[§]Mechanical Engineering and Applied Mechanics, University of Pennsylvania, Philadelphia, PA, USA

***Abstract*—Real-time haptic rendering of three-dimensional fluid flow will improve the interactivity and realism of applications ranging from video games to surgical simulators, but it remains a challenging undertaking due to its high computational cost. Humans are very familiar with the look and feel of real fluids, so successful interactive simulations need to obey the mathematical relationships of fluid dynamics with high spatial resolution and fast temporal response. In this work we propose an innovative GPU-based approach that enables real-time haptic rendering of high-resolution 3D Navier-Stokes fluids. We show that moving the vast majority of the computation to the GPU allows for the simulation of touchable fluids at resolutions and frame rates that are significantly higher than any other recent real-time methods without a need for pre-computations. Based on our proposed approach, we build a haptic and graphic rendering system that allows users to interact with 3D virtual smoke in real time through the Novint Falcon, a commercial haptic interface.**

I. Introduction

The computer graphics community has long been interested in creating virtual worlds where users can interact with synthetic entities as though they were real. To produce graphical feedback that looks authentic, interactive applications for entertainment and education increasingly rely on physically-based simulation to create sophisticated virtual worlds with realistic physical phenomena. Such simulations can go beyond visual feedback by also providing haptic (touch-based) feedback to the user, typically by monitoring user movements, detecting virtual object interactions, and computing and applying the appropriate haptic responses in real time [1]. Note that here we are concerned with kinesthetic (force) feedback, as opposed to tactile (cutaneous) feedback, though both are possible; our focus stems primarily from the wide availability of commercial force feedback devices, which allow for point-force interactions between the user and the computer and are adaptable to a variety of applications. The ability to touch virtual objects and experience interaction forces directly is known to increase the user's sense of presence and also improves the user's ability to perform manual tasks in virtual environments, e.g., [2].

Within the domain of physically-based simulation, we believe there is a great need for accurate real-time graphic and haptic rendering of three-dimensional fluid flow. A system that lets users see and feel the response of liquids and gases will find application in video games as well as high-fidelity surgical simulators; effects like smoke and blood flow will enhance the quality of the simulation and allow virtual environments to deliver more realistic feedback to the user. However, such simulations are usually computationally expensive, and therefore it is important for a physics engine to strike a balance between precision and performance. In the case of interactive applications with haptic feedback, performance becomes even more critical as haptic rendering requires a significantly higher frame rate than real-time graphical display to generate smooth force output [3].

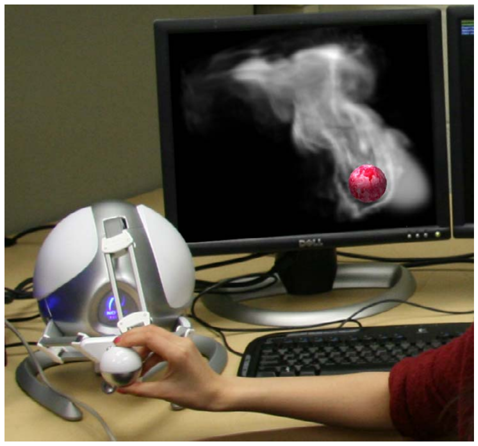
Fig. 1. The user grasps the handle of the Novint Falcon to move the spherical object and feel the forces it experiences in the fluid.

Real-time haptic rendering of physically-based 3D fluids remains a challenging problem today primarily because of limitations on Central Processing Unit (CPU) hardware. The highly parallel structure of the Graphics Processing Unit (GPU) has demonstrated much more powerful performance than general-purpose CPUs for a range of complex algorithms. Inspired by Crane's GPU-based implementation of real-time 3D Navier-Stokes fluids [4], we propose an innovative GPU-based parallel computing model for fluid interaction forces, and we build a system that enables real-time haptic rendering of high resolution 3D Navier-Stokes fluids. We show that moving both the fluid simulation and the interaction force computation to the GPU allows us to simulate touchable fluids at resolutions and frame rates that are significantly higher than any other recent real-time haptics methods [5]–[7], without requiring any pre-computations. Our sample simulation lets the user move a virtual sphere inside a 3D fluid flow by maneuvering the handle of a commercial haptic device, as shown in Figure 1; the user feels an amplified version of the force one would feel when touching real smoke in this way. The major contributions of this research are as follows:

- We demonstrate the feasibility of haptic interaction with 3D fluid flow for high resolution grids (up to $100\times100\times100$) at interactive rates without involving pre-computation.
- We devise a novel technique for computing the 3D forces of interaction between a mobile virtual object and a high resolution fluid simulation on the GPU.
- We establish an efficient pipeline that combines previous work on GPU-based fluid simulation with our hardware-accelerated method for haptic force computation.

II. Related Work

Many research teams have previously studied the topic of real-time fluid simulation for graphics, culminating in Stam's 2D real-time fluid simulation method [8]. Baxter and Lin were the first to augment this type of interactive fluid simulation with force feedback that enables the user to feel the fluid's response to the imposed object motion [5]. Their force computation method is derived from the fundamental mechanics principles of an incompressible Navier-Stokes fluid to ensure physical correctness of the rendered result. Baxter and Lin adapted their haptic display approach to build a drawing application that allows users to feel the forces that occur between the virtual brush and the liquid paint. However, their application is based on 2D fluid simulation and haptic rendering, which limits its broad usefulness. They state that extension to 3D is prohibitively difficult because "the computational cost of the fluid simulation places an especially restrictive limitation on 3D grid size" [5]. Our GPU-based 3D simulation system addresses this bottleneck by significantly reducing the time required for fluid simulation and force computation, providing the possibility of realistic haptic interaction in 3D space.

More recently, Dobashi et al. used partial offline pre-computation to achieve haptic simulation of a fishing rod and a kayak paddle each interacting with water [6]; for example, they pre-calculate the feedback force that the user should feel when the kayak paddle moves into different parts of the water at different velocities. This approach can be used effectively for large-scale simulation, but it has high memory requirements. Furthermore, this approach cannot be quickly reconfigured to provide real-time haptic rendering of a variety of high-resolution 3D fluids interacting with a variety of different virtual object at high frame rates. In contrast to this prior work, we aim for general-purpose real-time graphic and haptic rendering of high-resolution fluids on a personal computer.

Mora and Lee extend Baxter and Lin's 2D fluid simulation work by adding a spring-mass deformable surface to achieve a 3D look and feel [7]. Although their method achieves haptic rendering of 3D fluids at interactive rates, the fluid force computation is still done on the CPU in 2D (with a maximum spatial resolution of 15×15), so the fluid itself cannot generate forces in the depth direction. It is also important to consider whether treating the fluid surface as a spring-mass system is physically accurate. In contrast, our technique conducts fluid simulation and force computation in 3D space with acceleration from parallel reduction on the GPU; this combination enables a physically accurate fluid to be presented to the user at interactive rates both on the screen and through a haptic interface.

Crane et al. [4] provide a GPU-based 3D implementation of Stam's semi-Lagrangian implicit solver for the Navier-Stokes equation [9]. By taking advantage of the data-level parallelism of the GPU [10], we adapt this architecture's efficient framework for real-time interactive simulation of 3D fluids at the high spatial and temporal resolutions that are required for compelling graphic and haptic feedback. Compared with other GPU-based simulations of 3D fluids, such as [11], [12], ours is unique in its focus on interactivity and force computation. The following sections explain our approach in detail and present the performance results we have been able to achieve by applying it to the simulation of physics-based smoke.

III. Pipeline Overview

We target a simulation system that allows users to not only see but also feel virtual fluids. In particular, the user moves an impedance-type haptic device (which can measure hand motion and apply force in response) to control the position of a virtual object in the simulation environment. This object interacts with 3D fluids by displacing volume and applying an external force to the fluid, as seen in Figure 2. The haptic device simultaneously sends the force experienced by the virtual object in the fluid back to the user. Throughout the interaction, the user can feel the fluid's response as forces from the device handle and can simultaneously see the fluid's overall motion on the screen. Figure 1 demonstrates how a user

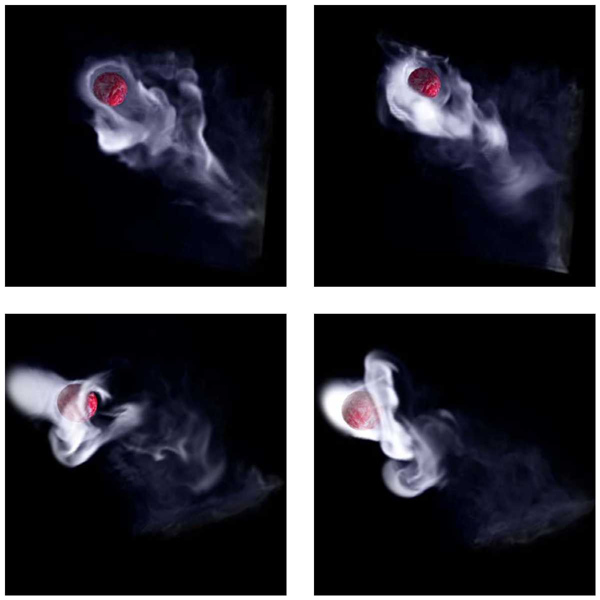

Fig. 2. Captured sequence of our simulation. Notice how the solid object interacts with the 3D smoke by displacing volume and applying an external force to the fluid.

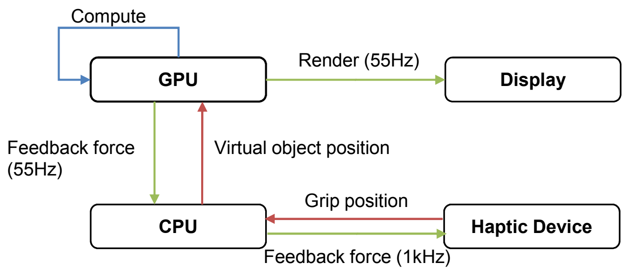

Fig. 3. System Diagram. The input data flow is marked with arrows in red, and the output data flow is in green. The listed output data rates were achieved for a fluid simulation with a resolution of 70 × 70 × 70.

grasps the handle of the Novint Falcon to move the spherical object and feel the forces it experiences in the fluid.

Figure 3 illustrates a diagram of our system. A high-speed thread on the CPU reads positions from the haptic device, transforms them to virtual object positions, and passes them to the GPU. The GPU is responsible for handling the fluid simulation update and the force computation, and for rendering the results of the simulation graphically to the display. Upon completion of each computational cycle (frame), the CPU copies the feedback force from the GPU and stores it to a shared memory location, from which the haptic thread constantly reads in order to deliver a smoothed feedback force to the user.

Our fluid simulation model is extended from Crane's GPU-based implementation of Stam's semi-Lagrangian method for solving the Navier-Stokes equation [4]. In particular, we treat the fluid grid as a set of 3D textures and rasterize them through Vertex and Geometry shaders for quick access to neighboring cells. Density and velocity advection, pressure computation, vorticity confinement and external force update are all handled through the Pixel Shader.

IV. Algorithm Details

The motion of the grip of the haptic device is scaled into our virtual work space and directly mapped onto the movement of the virtual object. As it moves, this object interacts with the 3D fluid by displacing volume and applying an external force to the fluid. The surrounding fluid, in return, exerts a net force and torque onto the virtual object. A smoothed version of this interaction force and torque is rendered by the haptic device and experienced by the user.

The algorithm generally can be separated into four parts: 1) texture preparation, 2) fluid simulation, 3) force computation and read-back, and 4) force smoothing. The following sections will explain these major steps in detail.

A. Texture Preparation

The CPU manages communication with the haptic device, reading its position at a rate of 1000 Hz and steadily passing this three-element vector to the GPU. When ready, the GPU takes the most recent position measurement and object dimension information to prepare a 3D texture that marks the type of each of the grid cells. This 3D texture will be used later for identifying and extracting the quantities that contribute to the force exerted on the virtual object.

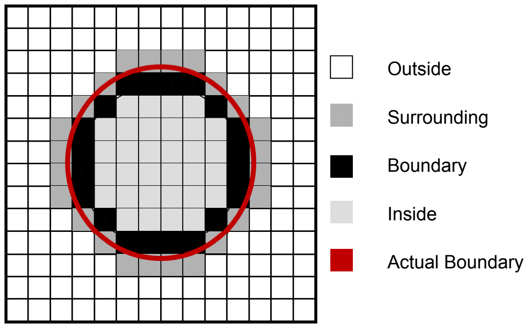

Fig. 4. Illustration of the texture preparation step for a spherical object. The example shown here represents a single slice of the inside-outside 3D texture that differentiates the four types of cells. The surrounding cells directly contribute to the force exerted on the virtual object by the fluid.

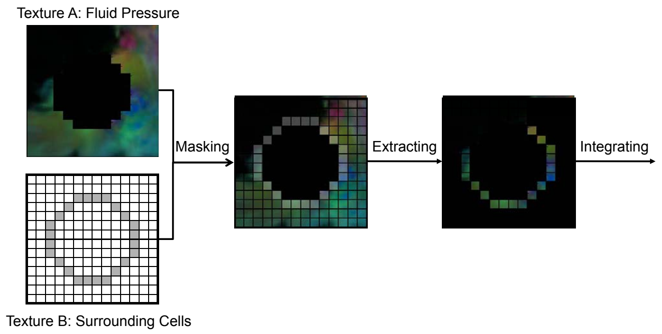

Fig. 5. Use of the inside-outside texture as a mask to identify surrounding cells and extract their pressures, which contribute to the force exerted on the virtual object. Each of the textures shown here represents a single slice of the corresponding 3D texture.

We follow Crane's steps [4] to create the inside-outside 3D texture that describes whether a grid cell is 1) inside, 2) on the boundary of, or 3) outside the obstacle. In addition, we introduce a new type of 4) surrounding cells which lie immediately outside the obstacle boundary by one grid unit. Because they share at least one face with a boundary cell, these surrounding cells are the ones that directly interact with the object. Figure 4 illustrates a single slice of the 3D texture that differentiates the four cell types defined above. The new type of surrounding cells will serve as a mask that lets us identify the layer of fluid cells with direct contribution to the force exerted on the virtual object, as depicted in Figure 5. We can categorize all the cells in a single pass on the pixel shader. Individual cell-type values are assigned and stored in the alpha channel of the inside-outside texture.

B. Fluid Simulation

The GPU then updates the voxelized fluid simulation using a semi-Lagrangian approach to implicitly solving the Navier-Stokes equation, largely following Crane's GPU-based implementation [4]. This process includes density and velocity advection, pressure computation, vorticity confinement, and displacement of fluid by the manipulated object.

C. Force Computation

After updating the simulation, the GPU estimates the force of the interaction between the virtual object and the fluid. As in [5], the net force and torque acting on a closed object inside incompressible viscous fluid can be expressed by the following equations:

$$\mathbf{F} = \int_S \boldsymbol{\sigma} \cdot \mathbf{n} dA \tag{1}$$

$$\boldsymbol{\tau} = \int_S \mathbf{r} \times \boldsymbol{\sigma} \cdot \mathbf{n} dA \tag{2}$$

where $\boldsymbol{\sigma}$ is the stress tensor and $\mathbf{r}$ is the vector from the object's center to the point of contact with the fluid.

Let $\mathbf{P} = \boldsymbol{\sigma}(x) \cdot \mathbf{n}$ be the force per unit area acting on a given point with normal $\mathbf{n}$ in the fluid. For 3D fluids, the vector components of $\mathbf{P}$ can be expressed as:

$$\mathbf{P}_x = -p\mathbf{n}_x + \mu \left(2\frac{\partial \mathbf{u}_x}{\partial x}\mathbf{n}_x + \left(\frac{\partial \mathbf{u}_x}{\partial y} + \frac{\partial \mathbf{u}_y}{\partial x} \right) \mathbf{n}_y + \left(\frac{\partial \mathbf{u}_x}{\partial z} + \frac{\partial \mathbf{u}_z}{\partial x} \right) \mathbf{n}_z \right) \tag{3}$$

$$\mathbf{P}_y = -p\mathbf{n}_y + \mu \left(2\frac{\partial \mathbf{u}_y}{\partial y}\mathbf{n}_y + \left(\frac{\partial \mathbf{u}_y}{\partial x} + \frac{\partial \mathbf{u}_x}{\partial y} \right) \mathbf{n}_x + \left(\frac{\partial \mathbf{u}_y}{\partial z} + \frac{\partial \mathbf{u}_z}{\partial y} \right) \mathbf{n}_z \right) \tag{4}$$

$$\mathbf{P}_z = -p\mathbf{n}_z + \mu \left(2\frac{\partial \mathbf{u}_z}{\partial z}\mathbf{n}_z + \left(\frac{\partial \mathbf{u}_z}{\partial x} + \frac{\partial \mathbf{u}_x}{\partial z} \right) \mathbf{n}_x + \left(\frac{\partial \mathbf{u}_z}{\partial y} + \frac{\partial \mathbf{u}_y}{\partial z} \right) \mathbf{n}_y \right) \tag{5}$$

where p is the fluid pressure at a given point and μ is the fluid's dynamic viscosity. For our grid-based simulation, we need to express equations (3), (4) and (5) in discretized form. Our simulation grid is uniformly sampled (with Δx increment) along all three axes. After approximating $\frac{\partial \mathbf{u}}{\partial x}$ at grid point i as $\frac{\partial \mathbf{u}}{\partial x} \approx \frac{\mathbf{u}_{i+1/2} - \mathbf{u}_{i-1/2}}{\Delta x}$ [13] and grouping similar terms we get:

$$\begin{aligned}\mathbf{P}_i = -p\mathbf{n}_i + \frac{\mu}{\Delta x}(2(\mathbf{u}_{i+1/2} - \mathbf{u}_{i-1/2})\mathbf{n}_i + \\ (\mathbf{u}_{i+1/2} - \mathbf{u}_{i-1/2} + \mathbf{u}_{j+1/2} - \mathbf{u}_{j-1/2})\mathbf{n}_j + \\ (\mathbf{u}_{i+1/2} - \mathbf{u}_{i-1/2} + \mathbf{u}_{k+1/2} - \mathbf{u}_{k-1/2})\mathbf{n}_k)\end{aligned} \tag{6}$$

$$\begin{aligned}\mathbf{P}_j = -p\mathbf{n}_j + \frac{\mu}{\Delta x}(2(\mathbf{u}_{j+1/2} - \mathbf{u}_{j-1/2})\mathbf{n}_j + \\ (\mathbf{u}_{j+1/2} - \mathbf{u}_{j-1/2} + \mathbf{u}_{i+1/2} - \mathbf{u}_{i-1/2})\mathbf{n}_i + \\ (\mathbf{u}_{j+1/2} - \mathbf{u}_{j-1/2} + \mathbf{u}_{k+1/2} - \mathbf{u}_{k-1/2})\mathbf{n}_k)\end{aligned} \tag{7}$$

$$\begin{aligned}\mathbf{P}_k = -p\mathbf{n}_k + \frac{\mu}{\Delta x}(2(\mathbf{u}_{k+1/2} - \mathbf{u}_{k-1/2})\mathbf{n}_k + \\ (\mathbf{u}_{k+1/2} - \mathbf{u}_{k-1/2} + \mathbf{u}_{i+1/2} - \mathbf{u}_{i-1/2})\mathbf{n}_i + \\ (\mathbf{u}_{k+1/2} - \mathbf{u}_{k-1/2} + \mathbf{u}_{j+1/2} - \mathbf{u}_{j-1/2})\mathbf{n}_j)\end{aligned} \tag{8}$$

If the fluid is inviscid (not viscous), we can drop the terms that are multiplied by the dynamic viscosity constant μ. Equations (6), (7) and (8) are then simplified to:

$$(\mathbf{P}_i, \mathbf{P}_j, \mathbf{P}_k) = -p(\mathbf{n}_i, \mathbf{n}_j, \mathbf{n}_c) \tag{9}$$

For torque computation, we can derive the discretized form of Equation 2 similarly to the above. We can now compute the total force acting on an object by summing the pressures acting on all cells on the object's boundary:

$$\mathbf{F}_{i,j,k} = \Delta x^2 (\mathbf{P}_i, \mathbf{P}_j, \mathbf{P}_k) \tag{10}$$

$$\mathbf{F} = \sum_{i,j,k} \mathbf{F}_{i,j,k} \tag{11}$$

After the advection step of the Navier-Stokes solver, all the fluid properties have been advected and stored in their respective textures, as described in [4]. Given the inside-outside texture prepared and the pressure texture $\mathbb{T}_P$ properly updated by the advection step, we can locate the surrounding fluid cells (the outer gray strip in Figure 4) and retrieve each cell's corresponding pressure p, as demonstrated in Figure 5. The inside-outside texture $\mathbb{T}_{IO}$ serves as a mask to identify and extract pressure from surrounding cells.

We apply Equations (9) through (11) to compute the accumulated force $\mathbf{F}$ exerted on the object. For all surrounding fluid cells, the exerted force $\mathbf{F_{i,j,k}}$ can be computed in parallel on a pixel shader in a single rendering pass. The individual $\mathbf{F_{i,j,k}}$ is then written in a new 3D texture $\mathbb{T}_F$ of the same dimension as the grid at the corresponding position (i, j, k). The resulting texture $\mathbb{T}_F$ contains force vectors $\mathbf{F_{i,j,k}}$ of all surrounding cells, representing all the individual force components being applied to the object simultaneously. Computing the integral force $\mathbf{F}$ exerted along the boundary surface involves only summing up all values currently stored in the newly created texture $\mathbb{T}_F$ as in Equation (11).

In each simulation cycle, to integrate all the force values stored in the 3D texture $\mathbb{T}_F$, a CPU-based linear scan approach would have to read back the entire 3D texture from the GPU. Although $\mathbb{T}_F$ contains only a fraction of non-zero values if the volume of the obstacle is small relative to the entire grid, the CPU-based approach still needs to loop through the entire texture to add up the values one by one, hence requiring $O(N^3)$ running time for an $N \times N \times N$ grid. The texture read-back and the brute-force linear integration are the potential bottleneck for performance in this CPU-based approach to force computation.

A GPU-based approach, however, can be optimized to $O(logN)$ passes via parallel reduction for 3D textures on the GPU. Figure 6 illustrates the basic idea of 3D parallel reduction [10]. Starting from a $4 \times 4 \times 4$ three-dimensional texture, the first pass will calculate the sum over eight $2 \times 2 \times 2$ subgroups of elements in $2 \times 2 \times 2$ individual kernels which reduces the problem from $4 \times 4 \times 4$ to $2 \times 2 \times 2$. This is recursively repeated until the problem is reduced to the final scalar texture, yielding a logarithmic number of iterations. Moreover, upon completion of parallel reduction, the integral force is stored at position $(0, 0, 0)$ of texture $\mathbb{T}_F$ and only this single value needs to be read back from the GPU, which is a potential speed-up for the simulation. The CPU copies this force vector into shared memory that is accessible by the haptic thread. In general, such memory read-back operations can cause significant delays and slow down the entire pipeline [14]. Although our approach already significantly reduces the GPU read-back through parallel reduction during the force integration process, it is still crucial to choose the most optimal type of memory based on the specific nature of each individual operation. A performance comparison of these CPU-based and GPU-based force computation approaches is provided in Section V.

The overall force computation process is as follows:

1) Use the inside/outside texture as mask to extract the

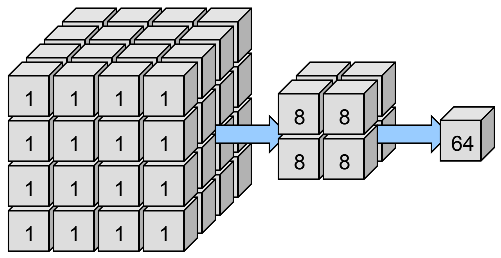

Fig. 6. The concept of GPU-based parallel reduction on 3D textures.

pressure of surrounding cells along the obstacle boundary in parallel.
2) Calculate the force component for each individual surrounding cell and store it into a 3D texture $\mathbb{T}_F$ at corresponding position in parallel.
3) Sum up all the values in texture $\mathbb{T}_F$ to compute the integral force via parallel reduction on 3D textures.
4) Have the CPU perform a single read-back from the GPU to acquire the resulting integral force.

Because Step 1 and 2 above can be processed together on the pixel shader in a single pass in parallel for all grid cells, the combined running time for force computation is $O(1) + O(\log N)$ passes. An additional benefit of this approach is that it requires just one read-back, thereby minimizing the effect of the data transfer delay from the GPU to the CPU.

D. *Force Smoothing*

Since real-time fluid simulation is quite computationally expensive (even on the GPU), we are able to achieve interactive update rates within the range of 20 to 75 frames per second (FPS), with computational grids generally between $30 \times 30 \times 30$ and $100 \times 100 \times 100$ cells. These rates are slower than the 1 kHz rate that is typically desired for smooth and natural haptic rendering, but the force information can still be used to create an interactive haptic experience for the user. Generally, it is undesirable to use a force filter for smoothing the haptic interaction between rigid objects, due to the high energy change at the transition between contact and non-contact states. However, as also found by Baxter and Lin [5], a force filter is acceptable and necessary for fluid haptic interactions, because such smoothing is able to eliminate the vibrations and force artifacts caused by the relatively low frequency of the haptic force update.

We use a discrete-time low-pass filter on the raw force values to send smoothed haptic feedback commands to the haptic interface at a rate of 1 kHz. To achieve a better user experience, we further modify the filter to eliminate high impulse forces that exceed a certain threshold and also to allow its bandwidth to be adjustable based on the simulation's current frame rate.

V. Performance and Discussion

Our simulation is implemented in DirectX10, and its performance is benchmarked on an nVidia 8800 GT graphics card with 512MB DDR3 memory and an Intel Q6600 CPU with 2GB DDR2 memory. The motions and forces for a sample interaction with a 3D smoke simulation are shown in Figure 7.

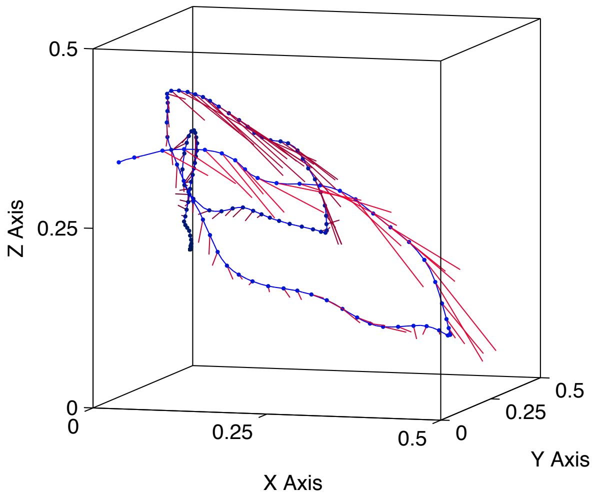

Fig. 7. Sample user motion (blue dots) with the computed interaction force (red lines). The smoke is issuing from a source at the top left corner with an initial velocity toward the bottom right corner; this directed fluid flow causes strong haptic forces that pull the user toward the bottom right corner as well.

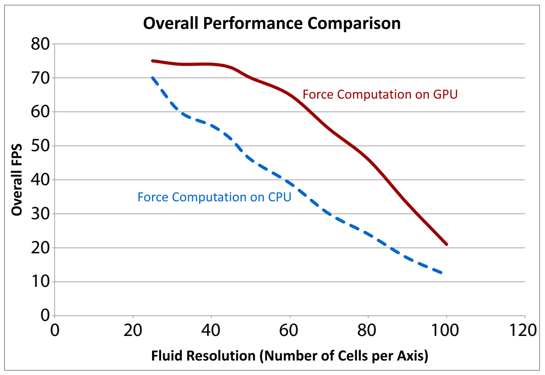

Fig. 8. A comparison on overall performance between CPU and GPU based methods for force computation.

Our GPU-based simulation achieves 20 FPS to 75 FPS for simulation grids that range from $25\times25\times25$ to $100\times100\times100$. The overall trade-off between update rate and grid resolution is presented in Figure 8. These results indicate that force computation is significantly faster on the GPU than it is on the CPU.

We are interested not only in the simulation's overall performance but also particularly in the speedup we obtain performing the force calculation on the GPU. We apply the same fluid simulation and alter the way the force is computed in order to achieve an equitable comparison. Figure 9 illustrates this aspect of the CPU-GPU comparison, breaking down in more details the individual costs of fluid simulation and force computation respectively. Computing the integral force on the CPU is nearly as expensive as the GPU-based fluid simulation itself. By moving the force computation to the GPU, we distribute the majority of the computational cost on fluid simulation. This result matches our theoretical analysis, where the GPU-based method requires $O(1)+O(\log N)$ passes via parallel reduction on 3D textures, while the CPU-based brute-force method runs in $O(N^3)$ for an $N\times N\times N$ grid. Notice in Figure 9 that as the

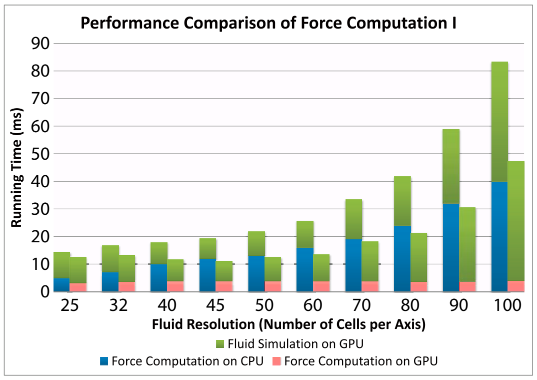

Fig. 9. A comparison on overall performance between CPU- and GPU-based methods for force computation with detailed breakdowns.

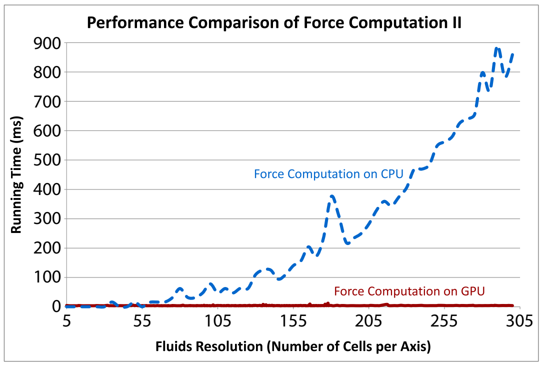

Fig. 10. A comparison between CPU-and GPU-based methods of computing the integral force alone. As the size of the simulation grid increases, computing the force integral on the CPU becomes a more severe bottleneck.

size of the simulation grid increases, computing the integral force on the CPU becomes an increasingly severe bottleneck. We further compare the performance of computing the integral force alone in Figure 10.

A comparison of overall performance between our GPU-based method and other recent real-time methods is shown in Table 1; these data reflect differences in algorithmic efficiency as well as variations in the computational hardware on which each was implemented. When considered in this context, the first benefit of our GPU-based approach is that it allows real-time haptic rendering of *three-dimensional* fluids with high physical accuracy. Second, it allows the haptic interaction to run at significantly *higher resolutions* than previous methods, while still maintaining an adequate frame rate.

VI. Conclusion and Future Work

Physically based fluid simulation by solving the Navier-Stokes equations requires a heavy computational workload. At present, most work on haptic fluid simulation is done on the CPU, which either is slow, is limited to low-resolution fluid grids, or requires substantive pre-computation. In this paper, we proposed techniques that move the vast majority of the time-consuming computation involved in real-time haptic rendering of 3D fluids from the CPU onto the GPU, and we managed to achieve a desirable performance at high resolution. We combined the 3D Navier-Stokes fluid simulation with 3D force computation to achieve real-time graphical rendering and

TABLE I
Performance Comparison

Implementation	Fluid Size	FPS
Baxter's 2D [5]	64×64	60
Mora's 2D$^+$ [7]	$15 \times 15 (\times 15)$	30
Our GPU-Based [this paper]	$70 \times 70 \times 70$	55

haptic rendering without pre-computation.

However, our current model for maintaining physical accuracy is still through an implicit method, which has a limited number of steps to converge when computing the distribution of pressure across the fluid. Furthermore, our voxelization-based method discretizes and maps the solid virtual object onto grid cells. Therefore it is not suitable for handling objects that are small relative to the fluid resolution, or that have fine details. Moreover, our current haptic rendering quality, similar to the previous work of [5], is limited by the fluid simulation update rate; though the grid sizes demonstrated here feel very good, exceptionally high fluid grid resolution may result in force feedback that feels rough and unnatural. In the future, in addition to addressing the issues listed above, we will seek to further optimize the GPU-based computations, further investigate and improve our force computation model, and apply this multi-modal simulation method to a wide variety of fluids and virtual objects.

References

[1] S. J. Biggs and M. A. Srinivasan, "Haptic interfaces," in *Handbook of Virtual Environments: Design, Implementation, and Applications*, ser. Human Factors and Ergonomics, K. Stanney, Ed. Lawrence Erlbaum Associates, 2002, ch. 5, pp. 93–115.

[2] C. R. Wagner and R. D. Howe, "Mechanisms of performance enhancement with force feedback," in *Proc. IEEE Joint Eurohaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, Mar. 2005, pp. 21–29.

[3] V. Hayward, O. R. Astley, M. Cruz-Hernandez, D. Grant, and G. Robles-De-La-Torre, "Haptic interfaces and devices," *Sensor Review*, vol. 24, no. 1, pp. 16–29, 2004.

[4] K. Crane, I. Llamas, and S. Tariq, "Real-time simulation and rendering of 3D fluids," in *GPU Gems 3*, August 2007.

[5] W. Baxter and M. C. Lin, "Haptic interaction with fluid media," in *GI '04: Proceedings of Graphics Interface 2004*, 2004.

[6] Y. Dobashi, M. Sato, S. Hasegawa, T. Yamamoto, M. Kato, and T. Nishita, "A fluid resistance map method for real-time haptic interaction with fluids," in *VRST '06*, 2006.

[7] J. Mora and W.-S. Lee, "Real-time 3D fluid interaction with a haptic user interface," in *Proceedings of IEEE Symposium on 3D User Interfaces 2008*, March 2008.

[8] J. Stam, "Stable fluids," in *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999, pp. 121–128.

[9] R. Fedkiw, J. Stam, and H. W. Jensen, "Visual simulation of smoke," in *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 2001, pp. 15–22.

[10] D. Goddeke, "GPGPU reduction tutorial," *GPGPU.org*, 2006.

[11] W. Li, X. Wei, and A. Kaufman, "Implementing Lattice Boltzmann computation on graphics hardware," *The Visual Computer*, vol. 19, no. 7–8, pp. 444–456, December 2003.

[12] Y. Liu, X. Liu, and E. Wu, "Real-time 3D fluid simulation on GPU with complex obstacles," in *Proc. IEEE Pacific Conference on Computer Graphics and Applications*, October 2004, pp. 247–256.

[13] R. Bridson, R. Fedkiw, and M. Muller-Fischer, "Fluid simulation: Siggraph 2006 course notes," in *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*. New York, NY, USA: ACM, 2006, pp. 1–87.

[14] D. Blythe, "The Direct3D 10 system," in *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*. New York, NY, USA: ACM, 2006, pp. 724–734.