PRACTICAL PATIENT-SPECIFIC CARDIAC BLOOD FLOW SIMULATIONS USING SPH

Scott Kulp[1]*, Mingchen Gao*[1]*, Shaoting Zhang*[1]*, Zhen Qian*[2]*, Szilard Voros*[2]*, Dimitris Metaxas*[1]*, Leon Axel*[3]

[1]CBIM Center, Rutgers University, Piscataway, NJ 08854, USA
[2]Piedmont Heart Institute, Atlanta, GA 30309, USA
[3]New York University, 660 First Avenue, New York, NY, 10016, USA

ABSTRACT

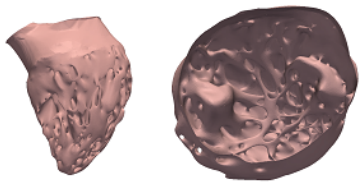
While recent developments in the field of ventricular blood flow simulations have pushed modeling to increasingly high levels of accuracy, there has been a steep cost in computation time. Current state-of-the-art simulators take days to run, which is impractical for use in a clinical setting. In this paper, we describe novel adaptations of the SPH algorithm to this problem to achieve an order of magnitude faster performance, while maintaining accuracy in the flow. By constructing appropriate boundary particles and wall motion and adding a fast collision detection component to an existing SPH architecture, our system is able to simulate a cardiac cycle in as little as 30 minutes. This breakthrough will, in the near future, allow the useful simulation of blood flow and its related characterization for clinically useful applications.

Index Terms— Blood flow, CT, cardiac, SPH

1. INTRODUCTION AND RELATED WORK

Simulating patient-specific blood flow has recently become an area of great interest to doctors. In patients who experience a heart attack, or in those who suffer from other various cardiovascular diseases, the motion of the heart walls and valves can become disturbed, leading to an abnormal blood flow pattern. If the blood is not being fully circulated within the heart and becomes stagnant, these patients are at high risk of thrombus, leading to stroke. Thus, it is very important for doctors to be able to visualize and understand a patient's cardiac blood flow. While it is possible to acquire flow data from MRI or Doppler ultrasound imaging, the relatively low quality and resolution of this data severely limits its usefulness to doctors. In recent years, though, as cardiac blood flow simulations have demonstrated accurate results and the potential of visualizing problems in the flow, this field of research has become increasingly active.

Early work in ventricular simulations used highly simplified models of the heart walls and structure. Jones et al. [1] were the first to use MRI data to perform patient-specific cardiac blood flow simulations, applying fluid velocity boundary conditions at the valves. More recently, in 2010, Mihalef et al. [2] used CT data to simulate left ventricular blood flow, and were able to compare the computed flow fields in healthy and diseased hearts. In [3], a more sophisticated method of extracting the heart and its motion from CT was used to capture the geometry of the trabeculae and show evidence of interactions between these structures and the blood flow.



(a) (b)

Fig. 1. Meshes reconstructed from CT data. (a) Outside heart (b) Apex.

However, as the geometry and motion of the heart wall models become more realistic and complex, the computation time for running these simulations becomes extremely high; for example, [3] states that a single simulation takes nearly a full week to complete. In a clinical setting, these long waits for one result would be unacceptable. However, in recent years, there has been increasing interest in meshless methods, such as Smoothed Particle Hydrodynamics (SPH), due to their improved running times. However, while these algorithms are fast, working with complex boundary conditions is a notoriously difficult unsolved problem. While SPH has been successfully used for blood flow simulations before [4], these studies have focused entirely on the flow through blood vessels, which is far simpler than that of the heart. In this paper, we present three major contributions: 1) A method to manage the highly complex boundary conditions of the heart using SPH by thickening the walls and treating the boundaries as particles, 2) A very fast and effective collision detection method optimized for a GPU implementation of SPH, and 3) Analysis of the SPH results that clearly show that these methods are practical and accurate enough in a clinical setting.

2. DATA AQUISITION

The CT images we used to generate the 3D heart mesh data were created on a Toshiba Aquilion ONE 320-MSCT scan-

ner, which produces 10 images of the whole heart in a single cardiac cycle at a volumetric resolution of 0.3mm, and an in-plane resolution of 512x512.

To build the mesh animation, we first apply a smoothing filter to the 3D image that corresponds to the beginning of diastole, and then extract a mesh through isosurfacing. Following manual cleanup with a 3D modeling tool, we transfer motion data, acquired from the same CT scan, to the mesh to generate a total of 10 frames. Since the valve motion is difficult to extract through CT imagery, we add models of the mitral and aortic valves generated from ultrasound data. Finally, we use cubic spline interpolation to create a final, smooth animation of 50 3D meshes, appropriate for use by the simulator. As can be seen in Figure 1, the reconstructed results are highly detailed, clearly showing the papillary muscles and trabeculae.

3. SIMULATION SYSTEM

The motion of an incompressible fluid is governed by the laws of conservation of momentum and mass. These two laws are modeled by the Navier-Stokes equations:

$$\rho(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u}) = -\nabla P + \mu \nabla^2 \mathbf{u} \quad (1)$$

$$\nabla \cdot \mathbf{u} = 0 \quad (2)$$

where $\mathbf{u}$ is the velocity field, P is pressure, ρ is the density, and μ is the coefficient of viscosity. The first equation balances the forces within the fluid and enforces conservation of momentum, while the second maintains conservation of mass.

3.1. SMOOTHED PARTICLE HYDRODYNAMICS

SPH [5] is a meshless method for solving fluid flow, where we seek to explicitly solve the equations of motion at unconnected points, or "particles," within the domain, each storing its own mass, density, pressure, position, and velocity. Unlike Eulerian-based methods, such as FDM and FEM, no computational mesh is required, and particles are free to move in the flow. At the beginning of a time step, for each particle i, we first search for all neighboring particles within some distance h. The particle's density is computed as follows:

$$\rho_i = \sum_{j=1}^{N} m_j W(\mathbf{r}_i - \mathbf{r}_j, h), \quad (3)$$

where N is the number of neighboring particles, m_j is the mass of particle j, r is a particle's position, and $W(r, h)$ is a smoothing kernel of radius h. We use the same choices for smoothing kernels as in [6]. In SPH, fluid is actually assumed to be semi-compressible, and so to find we pressure, we use the constitutive equation

$$P_i = c^2(\rho_i - \rho_0), \quad (4)$$

where c is the speed of sound and ρ_0 is the rest density, which we set to $1050kg/m^3$[2]. Higher values for c represent greater incompressibility, but will cause the simulation to become unstable if δt is too high. Once density and pressure are computed, we can compute the forces as follows:

$$\mathbf{f}_i^{pressure} = -\sum_{j=1}^{N} \frac{m_j}{\rho_j} \frac{P_i + P_j}{2} \nabla W(\mathbf{r}_i - \mathbf{r}_j, h), \quad (5)$$

$$\mathbf{f}_i^{viscosity} = \mu \sum_{j=1}^{N} \mathbf{u}_j \frac{m_j}{\rho_j} \nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h). \quad (6)$$

3.2. BOUNDARY MANAGEMENT

The enforcement of boundary conditions is one of the most challenging problems in SPH. To the best of our knowledge, no other group has attempted to adapt SPH to a problem of such complex geometry and movement as the left ventricle. A variety of methods have been proposed to prevent fluid particles from passing through solid boundaries. Most techniques either use fluid particles to model the solid boundaries, or use ghost particles [7]. Ghost particles generally perform quite accurately, but they are not well-suited for problems in which the solid is thin and complex. In these difficult problems, fluid particles on each side of the thin surface will produce their own ghost particles, which the fluid particles on the other side will include in its list of neighbors during density/force computation, generating instability.

Most techniques that use fluid particles as boundaries either keep the boundary pressure constant, or raise it slightly to discourage particles from entering. However, we found that in our problem, these methods causes significant instabilities in the flow, due to the complex nature of the geometry and wall movement. We note that as the left ventricle expands during diastole, the pressure within the left ventricle drops, which allows fluid from the left atrium to enter. If the boundary particles at the walls maintain a constant pressure as the fluid particles within encounter an lower pressure, the fluid will unrealistically be repelled from the wall and cause instabilities. Similarly, we found that low pressure at the walls during systole will also become unstable. To overcome this problem, we used a technique that allows the boundary particles to naturally change in pressure with the rest of the fluid.

First, to generate the boundary particles, an implicit function computing the distances to the mesh is rasterized onto a 100^3 grid. Then, at each grid point where the value of the implicit function is less than some distance ϵ, we label the particle generated at this position as a boundary. All other particles at a distance greater than ϵ are labeled as normal fluid particles. We then perform a search for the k closest mesh vertices, and the boundary particle's velocity is set to the inverse-distance weighted average of the velocities of its neighbors. Note that ϵ must be thick enough to prevent particles within the heart near the boundaries from including par-

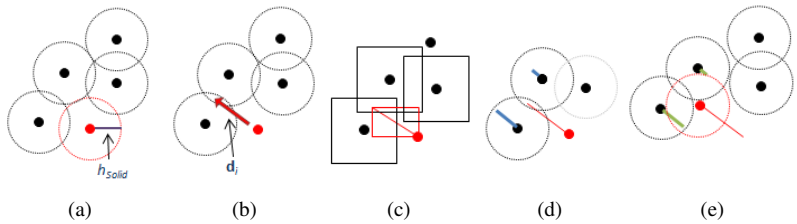


Fig. 2. Collision detection on CUDA. (a) Initial State - Boundary (red) and fluid (black), sphere of radius h_{solid} surrounds each particle; (b) Boundary moves, new location too close to fluid; (c) Pass 1: Bounding box collision to detect danger pairs; (d) Pass 2: For all danger pairs, determine if line segment intersects sphere. If so, push fluid particles forward in the same direction as the boundary; (e) New positions: Boundary is no longer too close to fluid particles

ticles outside the heart during the neighborhood search. We found that setting $\epsilon \geq h/2$ and $k = 5$ produces the most stable results.

At each time step, each boundary particle's density and pressure is computed in the same manner as a normal fluid particle, but its velocity is forced to match its corresponding heart mesh vertices' velocities. As such, during diastole, the pressure at boundary particles drops as they move slightly farther apart, and the opposite occurs in systole. We found this method to be remarkably effective, and consistently produced stable and accurate results, as we discuss later.

4. CUDA IMPLEMENTATION

To further improve performance, we implemented and optimized our simulator using CUDA, allowing it to take advantage of highly-parallelizable GPUs. A framework for implementing SPH on CUDA, including the neighborhood search, density/pressure gradient computation, etc, is described in [8].

Collision detection is another open problem in SPH. Most methods focus on polygon-sphere collision, such as [9], who recently developed a method for continuous collision detection optimized for GPUs. Our problem requires sphere-sphere collision detection, so we devised a new GPU-optimized algorithm for this task. First, we set h_{solid} to be the minimum distance a fluid particle must be from a boundary particle (Figure 2 (a)). At the beginning of each time step, the boundary particles will advance forward in time. Let d_i be the line segment connecting particle i's starting and end positions (Figure 2 (b)). In the first pass, we make a list of all fluid-boundary particle pairs that are in danger of colliding by performing a bounding box test between d_i and each neighbor (Figure 2 (c)). Each time a potential "danger pair" is encountered, the particle pair indices are saved in global RAM. When done, we have a full list of all potential collisions. We then execute a second CUDA kernel, where each thread performs a sphere-line segment collision test on a single pair (Figure 2 (d)). If a collision is detected, we know where on the line segment the intersection took place, and move the fluid particle in the direction of the boundary's motion such that the collision is resolved. The reverse procedure is done after the fluid particles move, to prevent them from moving through the boundary.

5. RESULTS

As mentioned previously, the models used in this simulation were generated from CT imagery from a healthy patient's heart. The simulation was run three times, with different settings of c and Δt for each run. Each experiment was initialized with 100^3 particles evenly-spaced throughout the domain, and the smoothing radius h was set to 2.5x the initial distance between particles. In run 1, we set Δt = 0:001s, and c = 10m/s. In run 2, we set Δt = 0:0005s, and c = 20m/s. Finally, in run 3, Δt = 0:00025s, and c = 30m/s. As c increases, the fluid becomes less compressible, and so we expect accuracy to improve. All simulations were performed on an Nvidia Geforce GTX 590. The running time of the simulations scaled linearly as Δt dropped. The total computation time for run 1 was 30 minutes, the time for run 2 was 62 minutes, and the time for run 3 was 126 minutes. All of these running times are orders of magnitude better than those described in other methods. Each time step took 2.5-3 seconds to complete. The force computation was the most expensive step, taking an average of 1.5 seconds per iteration. The density computation took, on average, 0.5 seconds each per time step. The rest of the time in each iteration was spread across the remaining CUDA kernels, including the collisions; compared to the density/force/velocity correction functions, the others' individual running times were negligible.

Visualizations of the blood flow can be seen in Figure 3. Columns 1, 2, and 3 correspond to Runs 1, 2, and 3, respectively. Frames in row 1 were taken during mid-diastole, and frames in row 2 were taking during mid-systole. All frames in a row were taken at equivalent time steps. The direction of the flow is seen in the direction of the embedded cones, and the velocity magnitude is shown by color, where blue regions represent velocities approaching zero, and red regions represent velocities approaching 1 m/s. We can see that as Δt decreases and compressibility goes down, the computed velocities within the heart go up and approach more accurate

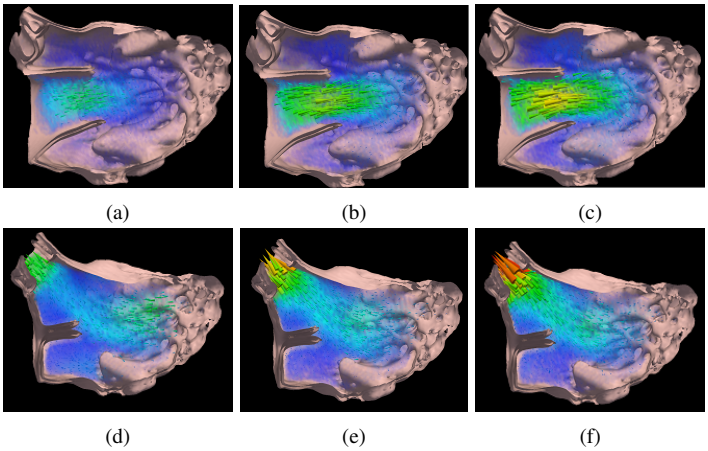


Fig. 3. Velocity fields for simulations with different Δt. Top row: Diastole, Bottom row: Systole, Left column: Δt=0.001s, Middle column: Δt=0.0005s, Right column: Δt=0.00025s.

values. We note, however, that by thickening the walls as described in Section 3.2, interactions between blood flow and trabeculae are not clearly visible.

Validation of cardiac blood flow simulations is difficult. In the future, we plan to acquire both CT and MRI images of the patient's heart, and use the MRI flow data to compare and validate. Here, we compute the ejection fraction by counting the number of particles within the heart at the end of systole and the end of diastole. We found that for run 1, the ejection fraction was about 0.42, for run 2, the ejection fraction was about 0.48, and for run 3, the ejection fraction was 0.50. Again, the increase in accuracy as Δt is decreased is expected, and gives the doctor a scalable option.

6. CONCLUSION

In this paper, we have described an adaptation of SPH to simulate blood flow through the left ventricle quickly and accurately. By simply scaling Δt and c, doctors can choose an appropriate level of accuracy, while maintaining faster speeds than previous methods allowed. To the best of our knowledge, this is one of the most complex problems successfully attempted with SPH, and is the fastest that patient-specific cardiac blood flow simulations has been solved.

7. ACKNOWLEDGEMENTS

This work is supported by the Multiscale Quantification of 3D LV Geometry from CT project, sponsored by NHLBI under Grant Award Number 5R21HL088354-02.

8. REFERENCES

[1] Timothy Jones, Timothy N. Jones, and Dimitris N. Metaxas, "Patient-specific analysis of left ventricular blood flow," in *MICCAI*, William M. Wells, Alan C. F. Colchester, and Scott L. Delp, Eds., 1998, pp. 156–166.

[2] Viorel Mihalef, Razvan Ionasec, Yang Wang, Yefeng Zheng, Bogdan Georgescu, and Dorin Comaniciu, "Patient-specific modeling of left heart anatomy, dynamics and hemodynamics from high resolution 4d CT," in *ISBI*, Wiro Niessen and Erik Meijering, Eds., 2010, pp. 504–507.

[3] Scott Kulp, Mingchen Gao, Shaoting Zhang, Zhen Qian, Szilard Voros, Dimitris N. Metaxas, and Leon Axel, "Using high resolution cardiac ct data to model and visualize patient-specific interactions between trabeculae and blood flow.," in *MICCAI 2011*, 2011, pp. 468–475.

[4] Matthias Müller, Simon Schirm, and Matthias Teschner, "Interactive blood simulation for virtual surgery based on smoothed particle hydrodynamics," *Technol. Health Care*, vol. 12, no. 1, pp. 25–31, Feb. 2004.

[5] J. J. Monaghan, "Smoothed particle hydrodynamics," *Annual Review of Astronomy and Astrophysics*, vol. 30, pp. 546 – 574, 1992.

[6] Matthias Müller, David Charypar, and Markus Gross, "Particle-based fluid simulation for interactive applications," in *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Aire-la-Ville, Switzerland, Switzerland, 2003, SCA '03, pp. 154–159, Eurographics Association.

[7] Andrea Colagrossi and Maurizio Landrini, "Numerical simulation of interfacial flows by smoothed particle hydrodynamics," *Journal of Computational Physics*, vol. 191, no. 2, pp. 448 – 475, 2003.

[8] Prashant Goswami, Philipp Schlegel, Barbara Solenthaler, and Renato Pajarola, "Interactive sph simulation and rendering on the gpu," in *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Aire-la-Ville, Switzerland, Switzerland, 2010, SCA '10, pp. 55–64, Eurographics Association.

[9] Min Tang, Dinesh Manocha, Jiang Lin, and Ruofeng Tong, "Collision-streams: Fast GPU-based collision detection for deformable models," in *I3D '11: Proceedings of the 2011 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, 2011, pp. 63–70.