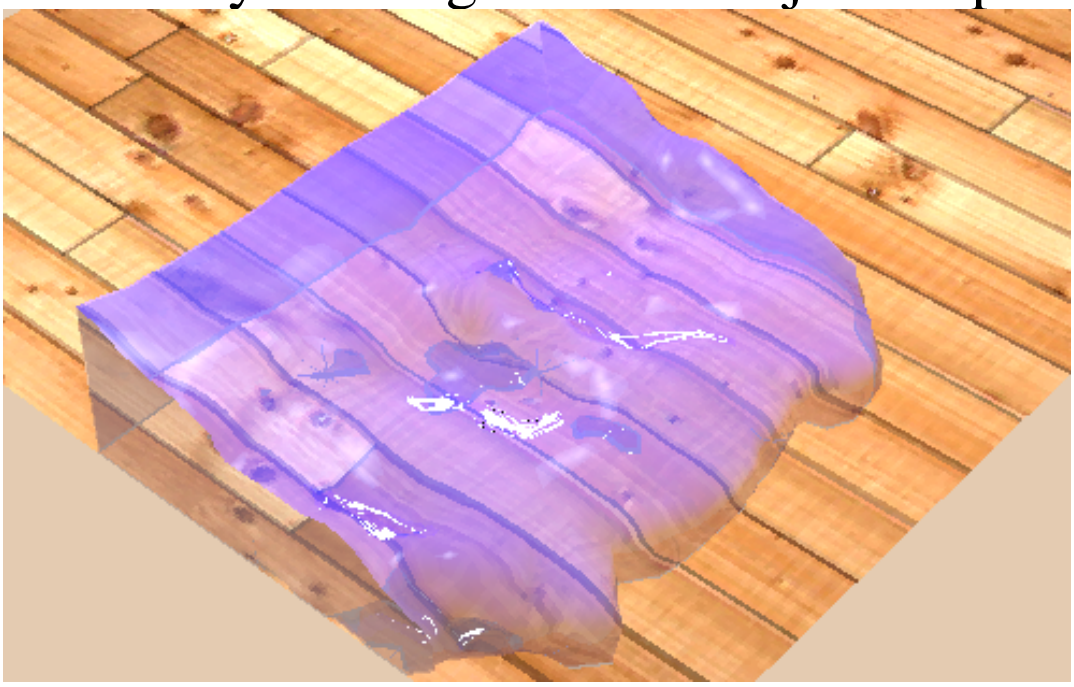
Fluid Ray Tracing - Final Project Report



An Zhang
Rensselaer Polytechnic Institute
zhanga11@rpi.edu

James Sours
Rensselaer Polytechnic Institute
soursj@rpi.edu

***Abstract*—This document presents the final project report for a fluid ray tracing project by An Zhang and James Sours. The motivation for the project is to produce convincing renders of liquids by creating a smooth and reasonable simulation of the liquid surface in 3D and animating fluid dynamics. The project aims to model phenomena such as refraction and participating media and improve the quality of liquid simulation by solving more complex phenomena such as water splash and explosion. Additionally, the project seeks to consider caustics of liquid surface using photons map and extend the photon mapping solution to store photons in 3D space. The paper discusses related works such as the introduction of full and surface cell and the accurate description of the velocity of fluid surface. The methodology employed includes the Navier-Stokes equations, Bounding Volume Hierarchy (BVH) acceleration for ray tracing. The results of the project include a comparison of BVH's and various effects. The conclusion of the project emphasizes the importance of fluid ray tracing in producing realistic renders of liquids.**

***Index Terms*—fluid simulation, ray tracing, photon maps**

I. Introduction

In recent years, the development of video games and movies has been characterized by a rapid pace of innovation and advancement. To create realistic visual effects in these media, it is necessary to use advanced algorithms for simulation and rendering. Among the many challenges involved in computer graphics, one of the most complex and intriguing is the simulation and rendering of fluids.

Simulating and rendering a fluid presents a challenging problem because it requires the accurate representation of complex physical phenomena such as surface tension, turbulence, and refraction. Even a simple image of a "broken straw" in a cup of water involves multiple physical effects that must be accurately simulated, such as the way the light bends as it passes through the fluid. Despite these challenges, the ability to simulate and render fluids with high accuracy has become increasingly important in a wide range of applications, from the creation of stunning visual effects in movies to the development of realistic simulations for scientific research. Therefore, investigating advanced techniques for ray tracing on fluids is a crucial area of research for computer graphics and scientific visualization.

This paper presents a method for simulating and rendering fluids in three dimensions, with a particular focus on the accurate representation of the fluid surface and the simulation of wave motion. To achieve this goal, we employ mathematical techniques based on the Navier-Stokes equation to calculate the velocity of fluid surface cells, which enables us to create a smooth and realistic simulation of the fluid surface. Once we have obtained a basic fluid simulation, we apply the ray tracing algorithm to render the fluid, including accurate refraction

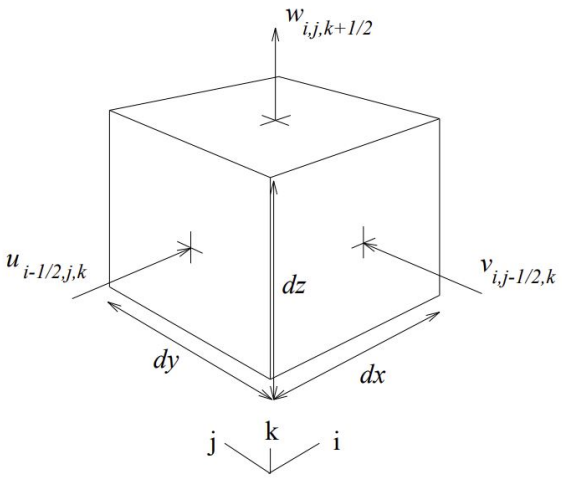


Fig. 1: velocity components on a typical cell(i, j, k). Graph cited from [1]

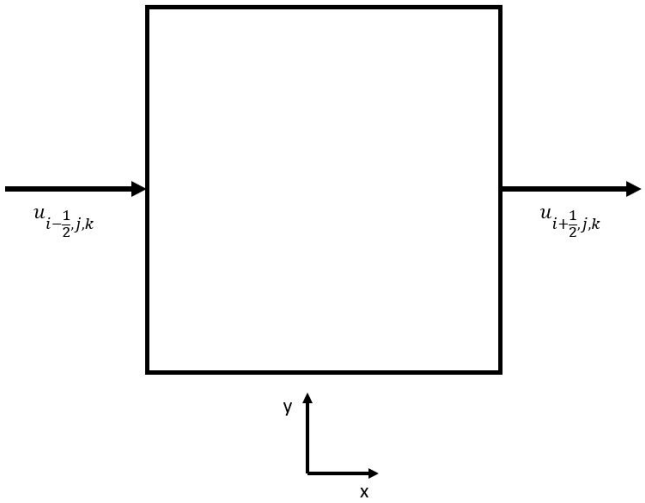


Fig. 2: Special case for surface cell in 2D

effects. To further enhance the realism of the simulation, we also use photon mapping to generate caustic effects, which add to the complexity and beauty of the rendered image.

To support our approach, we introduce a data structure for storing fluid primitives that enables faster access and more efficient computation, while also improving the accuracy and realism of the simulation. Overall, this paper aims to provide a detailed and comprehensive overview of our method for simulating and rendering fluids, highlighting the challenges involved in this complex and challenging task, and presenting novel approaches for achieving high-quality results.

II. Related Works

In recent years, a number of researchers have explored various approaches to simulating and rendering fluids using computer graphics techniques. One promising direction in this area is the introduction of full and surface cell methods [1], which enable the separation of fluid surface and inner parts, allowing for more efficient and accurate calculation of the velocity gradient.

Another important development in fluid simulation is the use of particle interpolation techniques, which have been shown to be effective in creating realistic fluid simulations [1]. By modeling the fluid as a collection of particles, and using interpolation methods to calculate the properties of the fluid at each point in space, these approaches have achieved impressive results in simulating the behavior of fluids in a wide range of scenarios.

However, more complex and realistic fluid simulations require additional considerations. For example, a more accurate approach for describing the velocity of fluid surface has been proposed [2], which involves adjusting implicit equation errors and solving for the behavior of breaking waves. These techniques can significantly improve the performance and accuracy of fluid simulations, and have been shown to be effective in a variety of contexts, including real-time rendering and scientific simulation.

There is a highly intriguing paper in the related work that introduces an innovative algorithm for producing water refraction without the use of ray tracing [3]. The approach involves pre-computing the refraction angle and mapping the calculation to a transformation matrix, which is subsequently utilized to distort the viewing and produce the desired refraction effect. This paper provides valuable insight into the techniques that may be employed to implement refraction and the expected outcomes that may be obtained.

III. Fluid Simulation

To simulate the behavior of fluids, we employ the full and surface cell method proposed by previous researchers [1]. This approach enables us to separate the fluid surface from the inner part, allowing for more efficient and accurate calculation of the velocity gradient. Additionally, we use marker particles to interpolate the velocity at specified positions [1], which provides a more accurate and realistic representation of the surrounding fluid behavior.

However, while previous researchers have made important contributions to the field of fluid simulation and rendering, the method for updating surface cell velocity in some special cases has not been clearly described. To address this limitation, we propose a novel approach for calculating the behavior of the fluid surface, which builds on previous work and introduces new mathematical techniques for modeling and simulating fluid behavior.

A. Update Surface Cell

Each full cell velocity will obey the continuity equation,

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \tag{1}$$

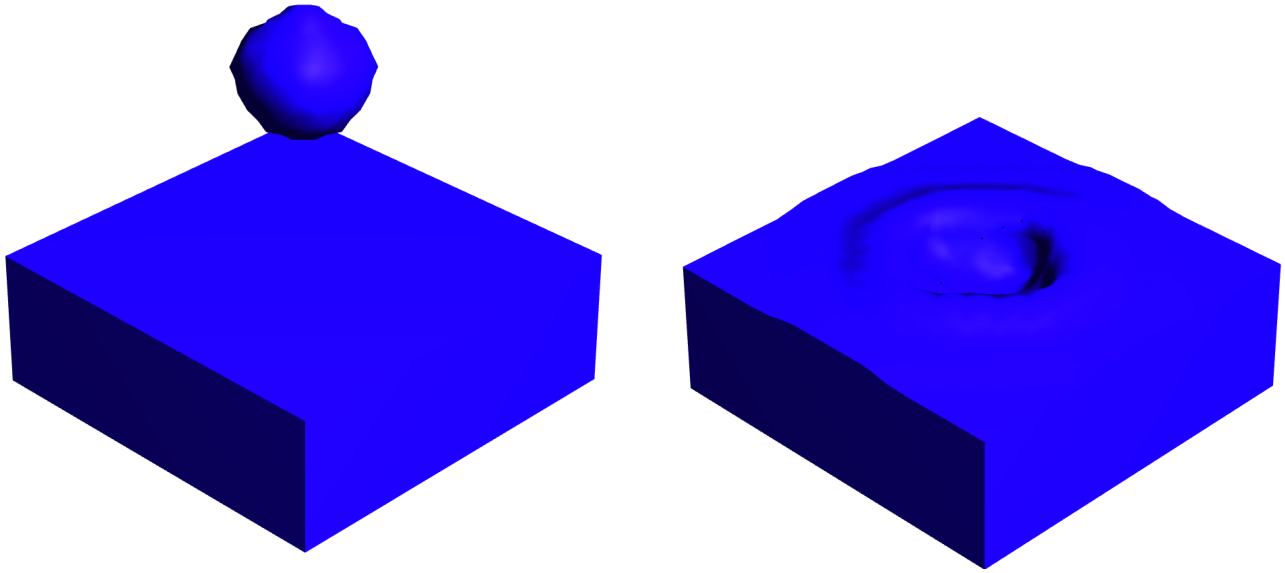


Fig. 3: Simulation for fluid drop

where after discretization, ∂u stands for the derivative or the change rate if u which is $u_{i+1/2,j,k} - u_{i-1/2,j,k}$ and ∂x stands for the length of cell in x-axis. Thus $\frac{\partial u}{\partial x}$ stands for the ratio of change of u over x-length of the cell. Same for $\frac{\partial v}{\partial y}$ and $\frac{\partial w}{\partial z}$.

In this project, we assumed not only full cell obey this equation but also the surface cell. One special case in 2D which did not talked by Foster is when two side in opposite direction is not open and other two sides are open, shown as figure 2.

In this case, we decide to vanish the velocity in x and y separately. Because we have both open sides in y-axis which have zero velocity, we can modify the equation 1 to:

$$\frac{\partial u}{\partial x} = \frac{u_{i+1/2,j,k} - u_{i-1/2,j,k}}{dx} = 0 \tag{2}$$

In order to satisfy this equation 2, we decide to set both $u_{i+1/2,j,k}$ and $u_{i-1/2,j,k}$ to their average which is $\frac{u_{i+1/2,j,k}+u_{i-1/2,j,k}}{2}$. In this case, we have numerator of the equation 2 equal to 0. After solving this case, we can generalize the solution for each case in 2D to 3D simulation.

In 3D simulation, when surface cell has more than 1 open side, we will vanish velocity of x, y and z separately which means we only consider each opposite pair of velocity,

- When one side is closed and the other is open, we take set the velocity of the open side to the closed side.
- When both side is closed we set both side equal their average value.

When surface cell has only 1 open side, we choose to vanish velocity by set the open side equal to the sum the amount of velocity comes into the cell from other 5 closed sides. For example, if we have the side $w_{i,j,k+1/2}$ is open and the other 5 sides are closed. We will update the top side by:

$$\begin{aligned} w_{i,j,k+1/2} = w_{i,j,k-1/2} &- \frac{dz}{dx}(u_{i+1/2,j,k} - u_{i-1/2,j,k}) \\ &- \frac{dz}{dy}(v_{i,j+1/2,k} - v_{i,j-1/2,k}) \end{aligned}$$

Moreover, equation above also satisfy the equation 1.

B. Marching Cubes

To achieve this, we utilize the marching cubes algorithm, which is an effective method for rendering fluid surfaces [4]. We generate an isosurface of a scalar field defined by the fluid marker particles. The algorithm partitions the scalar field into small cubes and determines the configuration of the isosurface within each cube. Depending on whether the corners of each cube are inside or outside the surface, the algorithm can use any of 16 possible configurations of triangle surfaces per cube.

To define the scalar field, we employ a distance function that returns a value of 0 in empty cells, a value of 2 in full cells, and a value that depends on the number of particles over density in cells containing a surface. Although this approach is not strictly accurate, it yields a heuristic approximation of the surface's shape that suffices for our purposes.

The output of the marching cubes algorithm consists of a set of triangles, and we compute normals for each vertex of the triangles by evaluating the gradient of the scalar field. These normals are utilized in our illumination model to create realistic lighting effects that produce visually appealing renderings of the fluid surface.

C. Stability

Moreover, the stability of the fluid system heavily depends on the chosen time step. If the time step is too large, the fluid particles may traverse several cells using the local velocity, and the cells in their path may have different velocities, which can result in instability. Therefore, we have designed an auto-adjustment algorithm for the time step. Whenever the time step is large enough to make the maximum speed particles move a distance greater than the size of a fluid cell, we will decrease the time step. This approach prevents phenomena such as velocity explosion and ensures the stability of the simulation.

IV. Ray Tracing and Rendering

A. Refraction

Refraction occurs when light travels from one medium to another with a different refractive index, causing a change in direction. The degree of bending that occurs depends on the difference in refractive indices between the two media, which is described by Snell's Law:

$$\eta_0 \sin\theta_0 = \eta_1 \sin\theta_1 \tag{3}$$

Where η_0 is the refractive index of the origin material, η_1 is the refractive index of the destination material, θ_0 is the angle of incidence for the incoming ray and θ_1 for the outgoing.

The input format was extended to include transparency and refractive index fields for materials, allowing for proper transmission of light rays through transparent objects in distributed ray tracing. To achieve this, a feature was added to keep track of exiting and entering fluid surfaces. Currently, the system identifies the exit point by detecting when the same material is hit twice. In addition to these modifications, we track the original material during ray tracing to calculate the ratio of optical densities.

In Figure 4, we experimented with different refraction indices to render the bunny underwater, and observed that the viewing angle was greatly affected by the refraction index. We experimented with different refraction indices to render the bunny underwater, and observed that the viewing angle was greatly affected by the refraction index. The effect of the refraction index on the viewing angle emphasizes its importance in accurate physical modeling of the liquids.

B. Liquid Surface Illumination

We expanded the material input format to include fields for transparency, refractive index, and specular exponent. We tagged the fluid model with a surface material, we simply apply these to the surface polygons.

Recall that the marching cubes algorithm outputs a set of triangles approximating the isosurface of a particular scalar field. However, rendering these directly was not acceptable since it produced gem-like artefacts as picture in the left image of Figure 5.

Since the gradient of the isosurface can be approximated at triangle vertices, we hoped to use these to approximate the normal at any point p in the triangular face.

Barycentric coordinates express a point p in the triangle as a linear combination of the vertices a,b,c such that the following hold

$$\alpha a + \beta b + \gamma c = p \tag{4}$$

$$\alpha + \beta + \gamma = 1 \tag{5}$$

$$\alpha, \beta, \gamma \geq 0 \tag{6}$$

Informally, the barycentric coordinates give a measure of how much the point "belongs" to each vertex.

$$\alpha N(a) + \beta N(b) + \gamma N(c) = N(p) \tag{7}$$

Formally, since the barycentric coordinates vary continuously as a function of p and take on value 1 when the p is equal to the corresponding vertex, we found it appropriate to replace the vertex positions in Equation 4 with their corresponding normals to obtain Equation 7.

These interpolated normals were used in the ray tracer to apply the Phong model at the fluid surface, to obtain the right image in Figure 5. Although it results in a smoother droplet when viewed from straight on, it does not remove the sharp corners when viewed from the profile.

One future improvement might include a surface reconstruction method, such as the one implemented by Hoppe et al. in [5]. This would be ideal for our purposes since it subdivides the surface while preserving sharp creases like the one formed when a fluid meets a wall.

We found that our initial specular exponent for the fluid material of 100 produced images with unreasonable artefacts (see Figure 6).

Figure 7 shows the effects of varying the Phong model exponent on the highlights. Large exponents emphasize highlights, which sometimes lead to unreasonable noise. Unlike common static materials, the Phong model exponent cannot be fixed and must be carefully chosen to render a realistic model, especially in complex shapes where the same exponent could result in white noise on the surface.

C. Raytracing Acceleration

We used a binary, axis aligned bounding box volume hierarchy to accelerate our ray tracing, which is effective at reducing computation [6].

Every leaf node held at most two primitives, and every internal node holds two children. We also assign an axis aligned bounding box to every node which contains its children and/or primitives. This guarantee means that if a ray does intersect one of the primitives, then it must also intersect the bounding box. Contrapositively, if the ray does not intersect the bounds, there is no need to test the ray against any of the children, as show in Figure 8.

Constructing a BVH in practice means sorting the list of primitives on some axis and splitting this at some point.

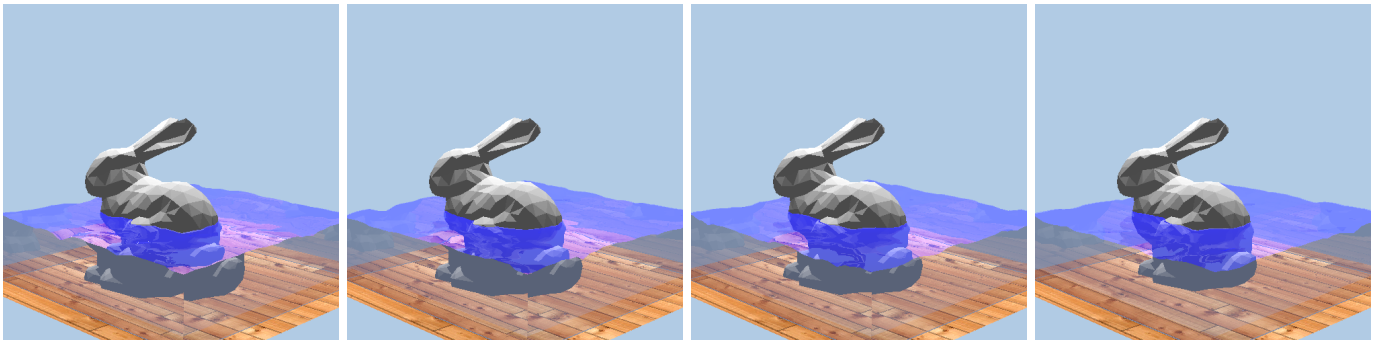


Fig. 4: Renderings of the bunny scene with varying refractive indices of the liquid material. From left to right they are: 1.5, 1.3, 1.1, and 1.0. Note how the apparent size of the bunny's hind quarters in the "wall" sides of the liquid is magnified by the refraction. At 1.0, there is no distortion present whatsoever. The physically accurate refractive index of water is 1.33, although we most commonly set it 1.5 to exaggerate the effect.

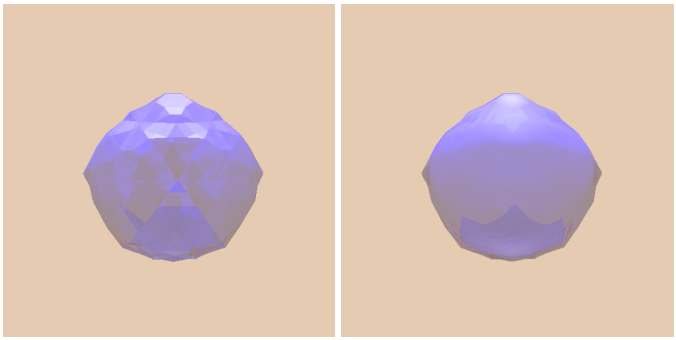


Fig. 5: A fluid drop, floating in space. On the left, the normals are constant across each triangle, resulting in a jagged, "gem-like" appearance. Interpolating the normals (right) by the barycentric coordinates results in a smoother image.

Ultimately, since constructing a perfectly balanced BVH is impractically, we evaluated several heuristics. First, we simply alternated the splitting axis, splitting at the median each time (called "alternate" in Table I). Second, we split on the axis on which the bounding box at the current level has the longest dimension, again at the median ("longest"). Third, we employed an estimate of the raytracing cost for a particular axis and splitting point, and take the split that minimizes the cost.

The cost of casting a ray into a BVH node U with children L and R depends on the conditional probability of a ray entering L given that it intersected U [7]. The relative conditional probabilities are related to the surface areas S_L and S_R of the children's bounding boxes. The number of primitives n_L and n_R give us the heuristic:

$$\text{Cost}(U) \approx S_L n_L + S_R n_R \tag{8}$$

Sample BVH's are pictured in Figure 9. Although visually the surface area heuristic has produced a closer fitting hierarchy, we performed a comparitive experiment to verify this intuition.

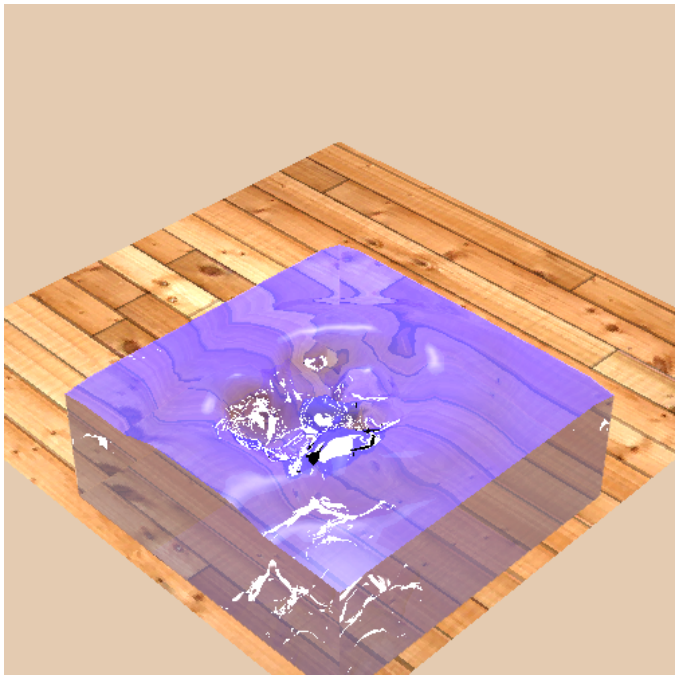


Fig. 6: Areas of high distortion produced unreasonable white patches, which were replicated through out the liquid volume due to reflection and refraction effects. The Phong exponent is 100.

To evaluate the effectiveness of the three heuristics ("longest", "alternate", "surface area", and "control", where we did not use a BVH) we rendered four scenes with each of the four heuristics and recorded the times in Table I. All times are system time measured on a 0:02.0 VGA compatible controller: Intel Corporation WhiskeyLake-U GT2 [UHD Graphics 620].

Observe that the surface area heuristic from [7] consistently rendered quicker than the other methods, thus validating our initial belief. Despite this, the new heuristic did not drastically out perform the others, although the gains would be more

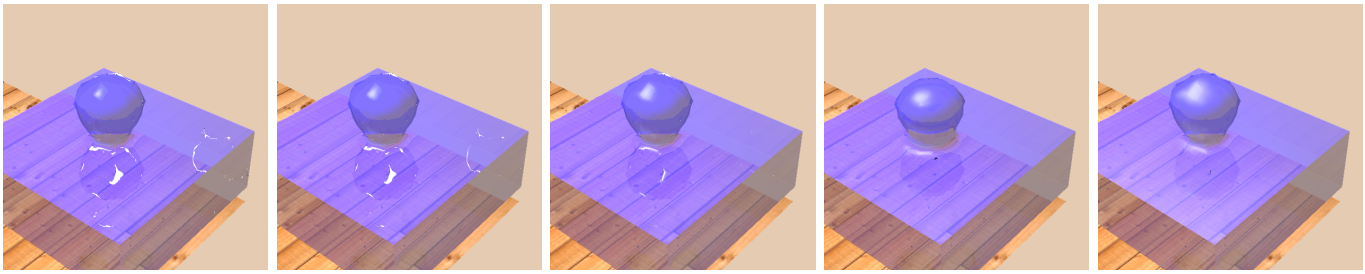


Fig. 7: The fluid drop scene, just as the fluid drop joins the water rendered with varying specular exponents. From left to right, the specular exponents are 100, 50, 20, 10, and 5. Note how the sharp white highlights in the leftmost images smear out as the specular exponent decreases.

.

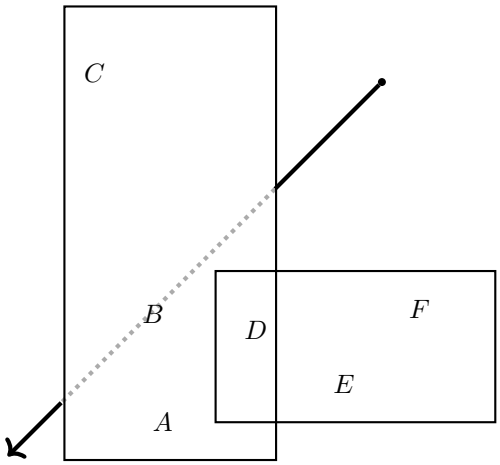


Fig. 8: Bounding Volume Hierarchies are effective at pruning calls to the primitive intersection routine. By testing against the bounding box for the right child, we eliminate the need to consider its primitives.

noticeable when rendering a sequence of images.

The freedom to evaluate any possible split does improves performance; however, the program spent considerably *more* time constructing the BVH using this heuristic. While this trade off is justified by Table I, it still represents a considerable opportunity for performance improvement. While this could be mitigated by restricting the search space of splits to choose from, the emphasis on construction time remains important due to the requirement us to rebuild the scene every time the geometry changes.

D. Dynamic Scenes Limitation

The construction cost of the BVH was negligible compared to the cost of rendering, regardless the heuristic used (see Table I). Still, when rendering a sequence of closely related scenes (such as those produced by our fluid animator!), we would prefer to have the better rendering performance of the surface area heuristic without paying the construction cost every frame. In this subsection, we examine the root cause of this restriction and propose potential solutions.

The marching cubes surface generation algorithm produces a new set of triangles each time the scene geometry is updated. Since the triangle data is not persistent, it is unsuitable to keep them in a BVH without reconstructing it anew for each scene. A possible solution would be to modify the marching cubes algorithm to compare the preexisting generation of triangles against the desired outcome.

To modify the marching cubes algorithm to compare pre-existing triangles with the desired outcome, we would need to implement a process for updating the positions of the existing triangles based on changes in the scene geometry. This could be done by tracking changes in the location of fluid marker particles and recalculating the isosurface accordingly. The updated surface geometry could then be compared with the previous generation of triangles to determine which triangles can be reused and which need to be modified or replaced. By updating the positions of the existing triangles, we could avoid the need to reconstruct the BVH each time the scene geometry is updated, which would reduce the memory allocation and computation time required for rendering. This method could be particularly beneficial for rendering sequences of animated scenes, as the same BVH could be used for multiple frames, further reducing the overall computational cost of the process.

V. **Results**

We have presented a method for solving Navier-Stokes equations for a specified situation, generating the mesh for the liquid surface and rendering the result with a BVH-accelerated ray tracer. See Figures 10 and 11 for raytraced snap shots of the fluid animation. However, there are still some challenges to overcome in our project.

VI. **Future Work**

One of the major challenges is the instability of the fluid system over time, which makes it difficult to achieve the desired results. Moreover, the current fluid algorithm has difficulty in eliminating the divergence of system.

In Figure 12, we rendered the bunny using photon maps, which looked visually appealing, although the liquid shape did

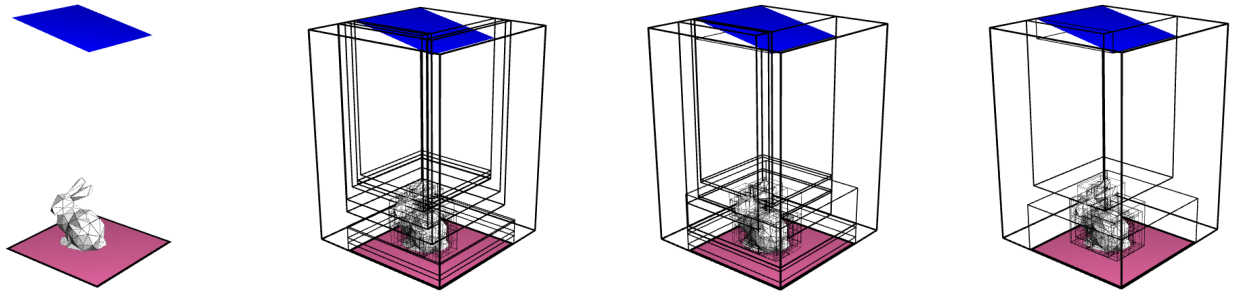


Fig. 9: The bunny model with 200 triangles. Pictured from left to right, the bounding boxes constructed by the longest axis heuristic, the alternate axes heuristic, and the surface area heuristic. Notice that the "surface area" bounding boxes differentiate between the light (blue, on top) and the bunny far quicker than the other two. In general, it has the freedom to choose various split points which creates bounding boxes more closely fit around the mesh.

not allow for caustic effects. Furthermore, we have not yet achieved the expected caustic effect, such as the highlights at the bottom of a swimming pool, in the photon mapping process.

To address these challenges, future work will focus on improving the stability of the fluid system, as well as refining the algorithm to smooth the fluid surface. Additionally, we will explore more advanced techniques to achieve realistic caustic effects and other properties of liquid, such as the reflection and absorption of light. We believe that these advancements will significantly enhance the realism of the liquid simulation and renderings, and thus contribute to the broader field of computer graphics and simulation.

VII. **Conclusion**

In this project, both An and James collaborated on merging the ray tracing code and fluid code, which took approximately one week to achieve OpenGL rendering, which we used to prototype scenes. Over the next three weeks, they focused on writing the algorithm, with both team members contributing significantly to the integration of the fluid base and ray tracing base. For the Navier-Stokes simulation, An played a critical role in extending a 2D solution to 3D. Meanwhile, James devoted considerable effort to the implementation of BVH for rendering fluid. After the successful implementation of the algorithm, both An and James spent a significant amount of time on debugging, creating various scenes, and writing the paper.

REFERENCES

[1] Dimitri Metaxas Nick Foster. Realistic animation of liquids. *Graphical Models and Image Processing*, 58:471–483, September 1996.

[2] Douglas Enright, Stephen Marschner, and Ronald Fedkiw. Animation and rendering of complex water surfaces. *ACM Trans. Graph.*, 21(3):736–744, jul 2002.

[3] Hongli Liu, Honglei Han, and Guangzheng Fei. Two-phase real-time rendering method for realistic water refraction. *Virtual Reality & Intelligent Hardware*, 2(2):132–141, 2020. Special issue on Visual interaction and its application.

[4] Harvey E. Cline William E. Lorensen. Marching cubes: A high resolution 3d surface construction algorithm. *ACM SIGGRAPH Computer Graphics*, 21(4):163–169, 01 August 1987.

[5] Hugues Hoppe, Tony DeRose, Tom Duchamp, Mark Halstead, Hubert Jin, John McDonald, Jean Schweitzer, and Werner Stuetzle. Piecewise smooth surface reconstruction. *Proceedings of the 21st annual conference on Computer graphics and interactive techniques - SIGGRAPH '94*, 1994.

[6] MATTHEW PHARR. *Physically based rendering: From theory to implementation*. MIT PRESS, 2023.

[7] J. David MacDonald and Kellogg S. Booth. Heuristics for ray tracing using space subdivision. *The Visual Computer*, 6(3):153–166, May 1990.

Scene Name	Primitive Count (thousands)	Heuristic	Construction Time (s)	Tree Height (nodes)	Rendering Time (s)	Percent Increase vs. Control
bunny-pool	5.2	control	0.0030	0	460	–
bunny-pool	5.2	longest	0.025	12	51	89%
bunny-pool	5.2	alternate	0.026	12	46	90%
bunny-pool	5.2	surface area	1.2	17	40	91%
bunny-1k	1.0	control	0.0024	0	160	–
bunny-1k	1.0	longest	0.074	9	14	91%
bunny-1k	1.0	alternate	0.080	9	13	92%
bunny-1k	1.0	surface area	0.27	14	11	93%
bunny-200	0.2	control	0.00050	0	37	–
bunny-200	0.2	longest	0.027	7	8.2	77%
bunny-200	0.2	alternate	0.029	7	7.5	79%
bunny-200	0.2	surface area	0.066	11	6.7	82%

TABLE I: Performance gains due to various BVH construction heuristics on relevant scenes.

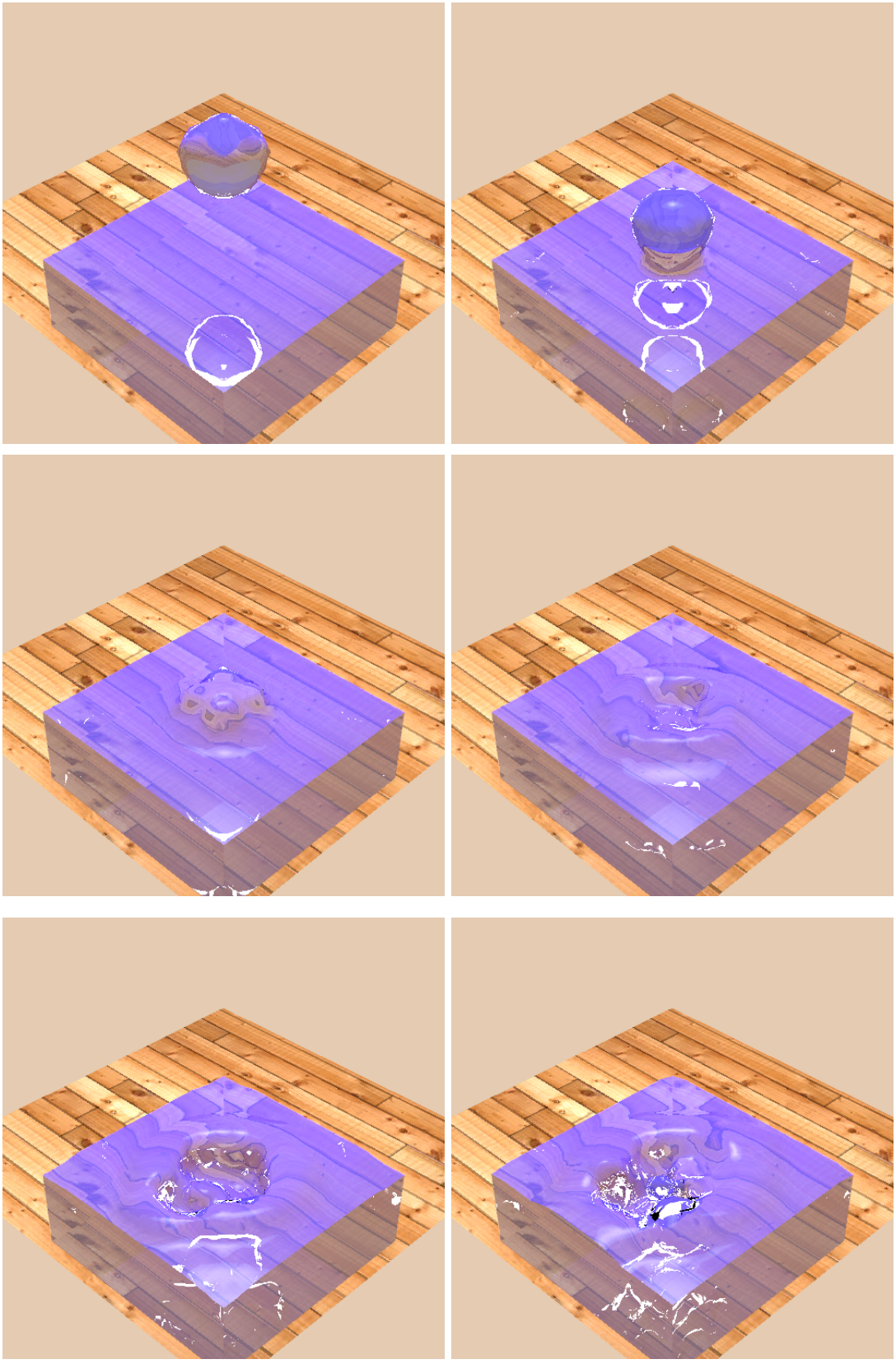


Fig. 10: Snapshots of the fluid drop animation. Note the predominance the unreasonable specular highlights, discussed in Section IV-B).

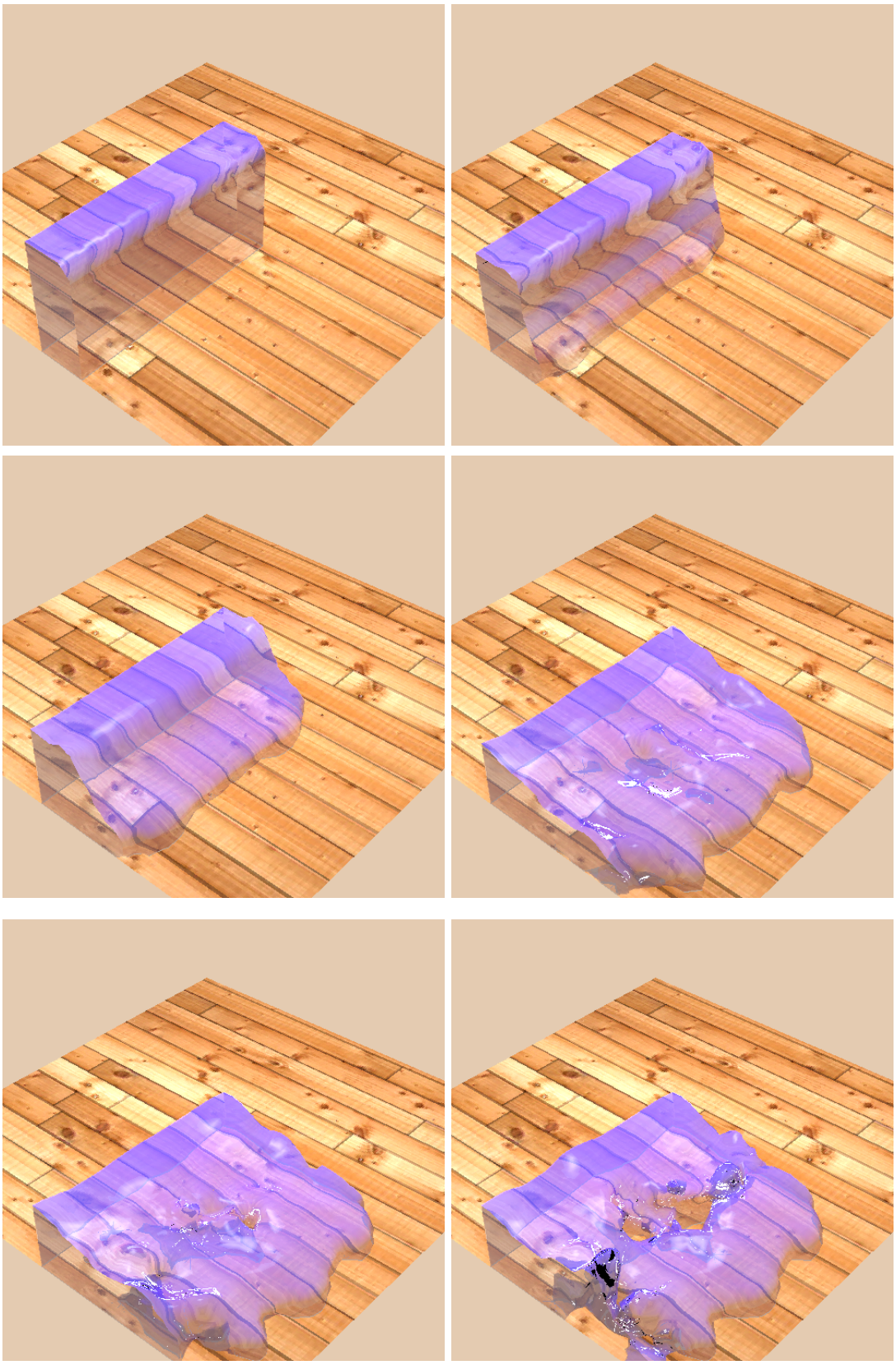


Fig. 11: Snapshots of the fluid dam animation. Note the instability discussed in the paper.

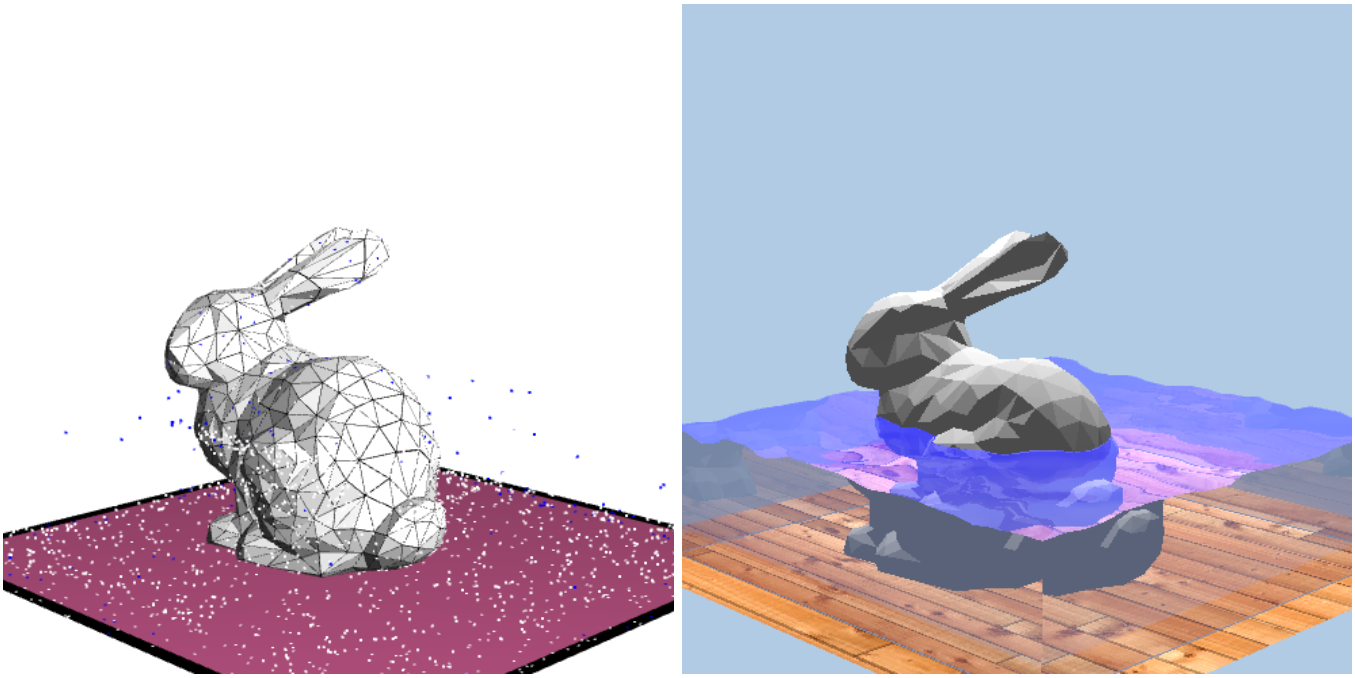


Fig. 12: Photons absorbed by the bunny pool scene (left) and the rendered image (right). The first stage shot 10,000 photons. Note the relatively uniform scattering of photons on the floor, as well as a few on the surface of the water. This results in an image of comparable quality, although no observable caustic effects despite the curvature of the liquid surface.