ISART DIGITAL

Real-Time Rendering of Molten Glass with Raymarching Technique on Point Clouds

Félix Chevalier
Manon Méhalin
Maxence Retier

Game Programming
Option GPU

Septembre 2024 - Juin 2025

Tuteur(s) ISART DIGITAL:
Maël Addoum
m.addoum@isartdigital.com

Tuteur entreprise :
Sylvain
CONTASSOT-VIVIER
sylvain.contassotvivier@loria.fr

Résumé

Table of contents

1 Introduction

Glassblowing is an art form with deep cultural and historical roots, especially in France, where it represents centuries of craftsmanship and tradition. However, preserving this heritage poses modern challenges, such as the environmental impact of energy-intensive training methods and adapting these artisanal techniques to new technologies.

The overarching goal of the LORIA (Laboratoire Lorrain de Recherche Informatique et ses Applications) project is to design a fully immersive augmented-reality experience that allows users to blow and shape their own molten glass to create various forms, such as vases or sculptures. Our challenge is to balance the realism of the interactions and the technical performance, paving the way for new possibilities in digital creation and virtual craftsmanship.

Beyond its technical innovations, this project contributes to the preservation of glassblowing while addressing sustainability. By providing a virtual training tool, our project could reduce the reliance on furnaces for practice, significantly lowering the ecological footprint of the learning process.

We are creating a real-time simulation of molten glass using raymarching techniques on 3D point clouds. Raymarching, a rendering method in which the renderer progressively marches along rays to detect surfaces, offers a highly flexible approach to representing complex physical deformations. In our case, raymarching allows us to dynamically model the surface of molten glass.

This work departs from traditional mesh-based rendering. Instead, we employ implicit surfaces and raymarching, as they are well-suited for handling point cloud data, enabling a seamless integration of dynamic physical simulation with advanced visualization techniques. Despite the complexity of molten glass, which behaves as both a fluid and a solid depending on temperature and movement, our approach proposes a promising solution for realistic real-time rendering.

2 The methodology

To address the complex challenge of simulating molten glass in real time, our team adopted a structured and experimental approach rooted in both graphics programming and interactive simulation. Our goal was not to replicate the full physical behavior of molten glass, but rather to create a visually convincing and responsive system suitable for immersive environments. This required a balance between artistic fidelity, technical feasibility, and performance constraints.

2.1 Technical Scope and Approach

The simulation of molten glass presents several unique challenges : it must appear fluid, semi-transparent, and dynamically deformable. Traditional mesh-based rendering approaches are not well-suited for representing such materials, especially when their shape continuously evolves. Instead, we chose a **raymarching-based rendering technique**, which allows us to render **implicit surfaces** directly without relying on predefined geometry.

Raymarching is a rendering method where rays are cast through the scene, and the algorithm advances step by step along each ray until it encounters a surface. This is particularly well-suited for representing mathematically defined surfaces such as metaballs, soft blends between shapes, or deformable volumes — all of which are ideal representations for molten glass.

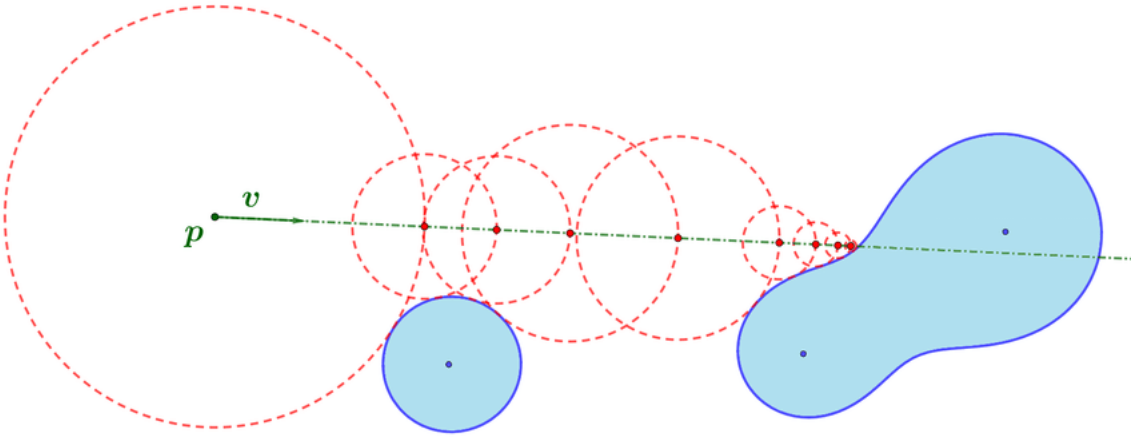

FIGURE 2 – Raymarching system

2.2 Project Breakdown

The technical development of the project was organized into several incremental phases :

1. **Initial Raymarching Setup**
   We began by creating a minimal raymarching loop capable of rendering a basic torus shape using signed distance functions (SDFs). This stage allowed us to validate our rendering pipeline and experiment with implicit surface definitions.

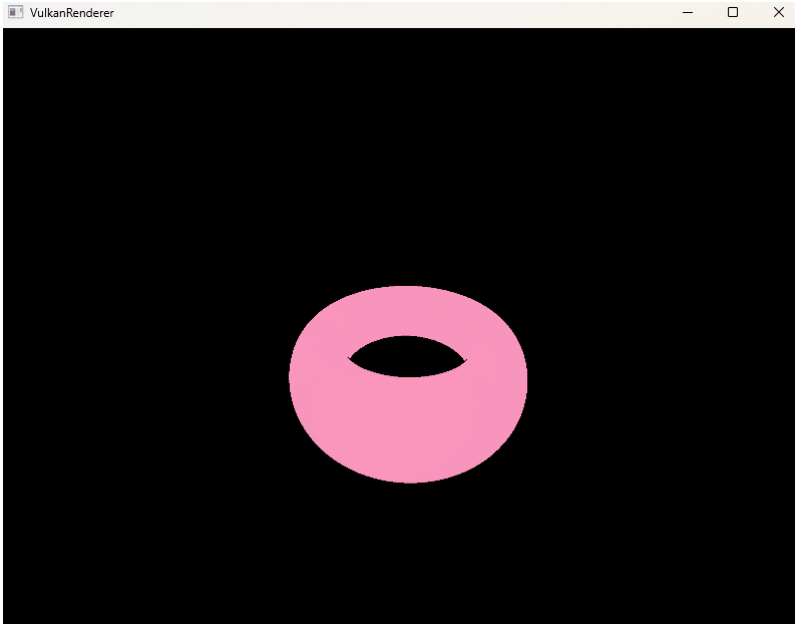

FIGURE 3 – Raymarching torus

2. **Implementation of Shape Blending**
   To simulate the soft, flowing properties of molten glass, we introduced blending operations between multiple SDFs. By combining a torus and a sphere with smooth transitions, we created organic shapes that better represent the deformations encountered in real glassblowing.

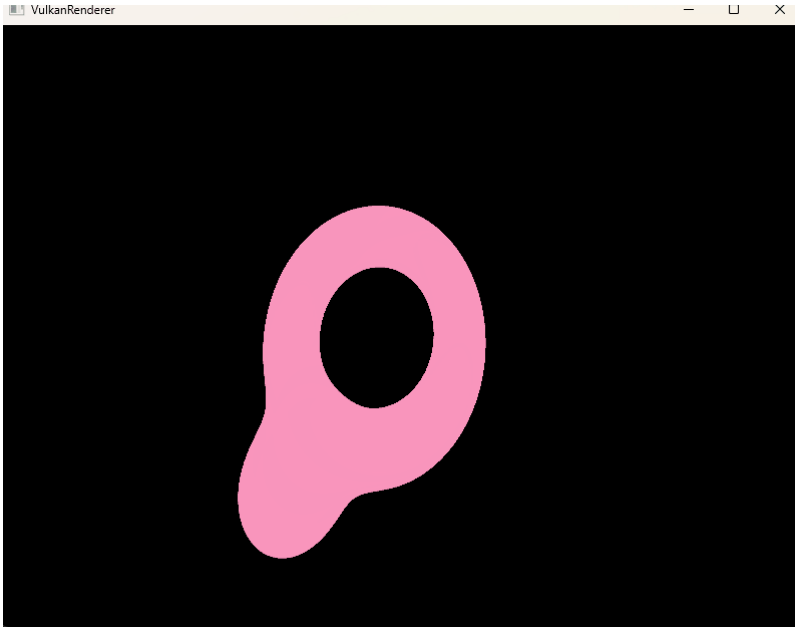

FIGURE 4 – Raymarching blending

3. **Lighting and Reflection Enhancements**
   Realistic shading is essential for transparent and reflective materials like glass. We incorporated Phong-based lighting and reflection approximations to enrich the visual realism of our simulations. Fresnel effects and surface normals derived from SDF gradients were also integrated to reinforce the material appearance.

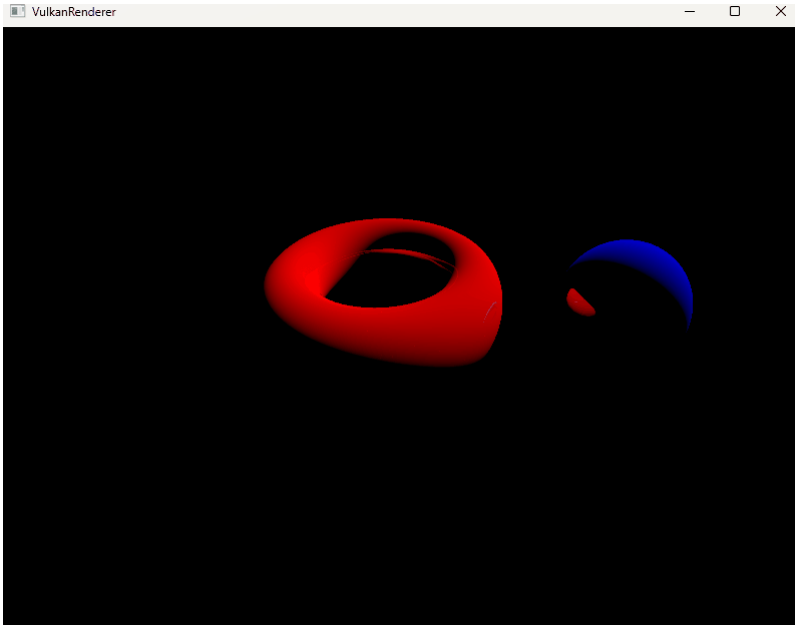

FIGURE 5 – Raymarching Phong light

4. **Transition to Compute Shaders**
   Our initial implementation used fragment shaders for raymarching, but we encountered limitations in performance and flexibility. We decided to switch to compute shaders, which offered greater control over parallel execution and memory usage. This transition enabled us to improve efficiency and prepare for more complex simulations involving 3D point cloud data.

5. **Space Partitioning algorithm**
   The slow part of the ray-marching algorithm is the sphere-distance part. To improve the performance with big point cloud, we need a space partitioning algorithm. Because we want the minimal number of generation in our binary tree, we want to keep the same amount of points in each sub-divided space. We don't use an usual k-d tree (like [1]) because of the possible none-uniform density of the points.

   We created our own homemade space partitioning algorithm to fit our requirements.

6. **Custom Space Partitioning algorithm**
   The first step of our space partitioning algorithm is to create an axis-aligned bounding box (AABB) of our point cloud. We make cuts, recursively, on the X, Y and Z axis to make smaller boxes. The cuts are axis aligned. The cuts are made from the median of the points inside the AABB, to keep the same amount of points in each sub-box. Keeping the same amount of point in smaller boxes is very important to get the less number of generations in our binarytree.
7. **Binary tree to**
   We can represent every AABB and cut as nodes. We create a binary tree where the root is the AABB of the whole point cloud. Every node contain a smaller box created by cuts.

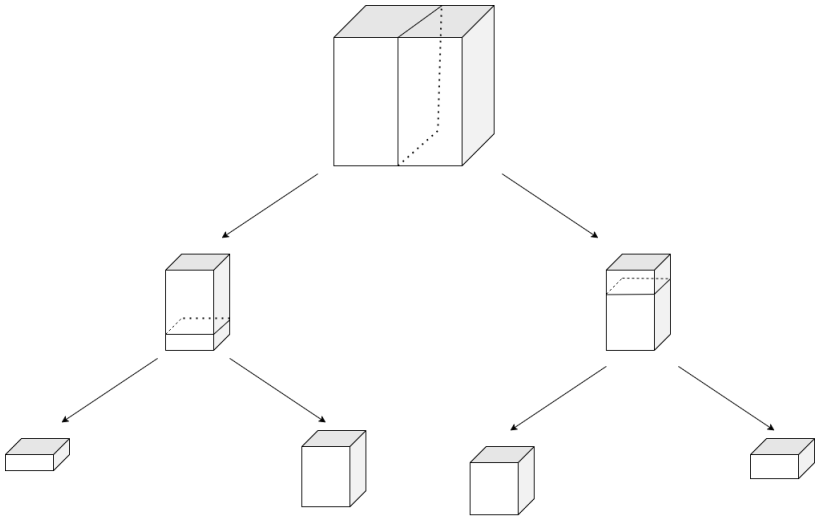

FIGURE 6 – Binary tree

8. **Unique index, morton number**
   To align theses boxes and information in the memory, we need to give unique indexes to each node. We use the morton number to store the nodes. The root index is 1 and every children indexes are the parent index and an additional 0 or a 1 ; 0 when the box is before the cut and a 1 when the box is after the cut.
9. **Space Partitioning implementation in raymarching compute shader**
   TODO

2.3 Development Environment

Our project is implemented using Vulkan, a low-level graphics API that offers fine-grained control over GPU resources and pipeline stages. Vulkan was chosen over

OpenGL for its superior performance, multithreading capabilities, and support for compute shaders.

The rendering system is written in C++, while all GPU-related operations (raymarching logic, lighting calculations, memory access) are written in GLSL, with separate modules for fragment and compute shaders. We used version control (Git) and organized our codebase into modular components to ensure clarity and scalability.

2.4 Key Technical Choices

Several key decisions shaped our methodology :

— Use of raymarching : Enabled the rendering of smooth, deformable shapes without relying on explicit meshes.

— Implicit surfaces and blending : Allowed the simulation of glass behavior in a fluid-like state with natural transitions between forms.

— Compute shader migration : Provided significant gains in performance and flexibility by decoupling the rendering from the traditional graphics pipeline.

— Preparation for volumetric and point-based rendering : Created a foundation for future work involving 3D data structures and more physically-based simulation models.

2.5 Team Workflow (not sure that is necessary)

The project was developed in a team of three, with clearly defined technical roles. We focused on the implementation of the rendering techniques, including the design of shaders and the performance transition to compute shaders. Regular weekly meetings were held with our supervisor at LORIA, which allowed us to validate our progress, discuss architectural decisions, and refine our objectives. We followed an iterative and exploratory workflow, frequently testing and benchmarking our changes to assess visual quality and rendering speed.

3 Results and discussion

3.1 Achieved results

At this stage of the project, we have successfully implemented a fully functional raymarching rendering loop capable of visualizing implicit surfaces in real time. Basic shapes such as spheres and torus are rendered using signed distance functions, and we have developed blending techniques to create smooth transitions between them, mimicking the organic behavior of molten glass(viscosity).

In addition to shape blending, we have integrated dynamic lighting and reflection models. Surface normals are calculated from the gradient of the SDFs, and lighting effects such as specular highlights and environmental reflections are applied to reinforce the appearance of transparent, reflective material. These enhancements provide a visually convincing representation of molten glass.

One of the most impactful milestones was the migration from a traditional fragment shader pipeline to compute shaders. This change allowed us to gain significant control over execution, memory usage, and thread distribution, leading to better performance and future scalability.

We are able to create a whole binary tree of the space partitioning algorithm from point cloud. We managed to give to every node an unique indexe to move easily from a sub-spaces to another.

3.2 Performance and Realism Evaluation

Our transition to compute shaders has improved frame rendering times and reduced GPU workload by optimizing the way threads access memory and perform raymarching steps. Unlike fragment shaders, compute shaders enabled us to decouple rendering from the rasterization pipeline, allowing for more direct and efficient computation of each pixel's color.

While we have not conducted formal benchmarks yet, visual fluidity and responsiveness are noticeably improved, especially when increasing the complexity of the rendered shapes.

In terms of realism, our blending technique and light handling have proven effective in conveying the semi-solid, semi-fluid nature of molten glass. The material responds well to lighting, and the reflections provide strong visual cues about shape and curvature.

3.3 Limitations and Ongoing Work

Despite promising results, several key features remain under development :

— We currently simulate the glass deformation only visually, without incorporating real physical simulations or user interaction.

— The use of point clouds as a base for molten glass geometry is still a work in progress and requires integration with the existing raymarching system.

— We do not yet support live user interaction, such as shaping the glass in real time, which is a long-term goal of the project.

Additionally, handling transparency and internal refraction — two hallmarks of real glass — poses additional challenges that we aim to explore in the next stages.

3.4 General Discussion

The results obtained so far demonstrate that real-time simulation of molten glass using raymarching is a feasible and effective approach. The combination of implicit surfaces and compute shaders allows for a flexible and powerful rendering method capable of capturing complex visual phenomena.

Our system lays a strong foundation for further development, including interactivity, volumetric rendering, and integration into AR/VR applications. More broadly, this project showcases how digital tools can contribute to the preservation of traditional craftsmanship, offering new possibilities for training, education, and cultural dissemination — all while reducing environmental impact.

4 Conclusions, future prospects

5 Bibliographical references

[2] by Floney Yang, which helped us understand collision techniques, though it does not address deformable materials like glass.

GitHub repositories like [3] and [4], which offer practical raymarching examples but lack real-time physical simulation.

Adrian Biagioli's blog [5] and Nabil N. Mansour's article [6], both of which provide excellent foundations but do not cover dynamic systems like ours.

[7] focuses on using Finite Element Method (FEM) models to optimize the control of glass manufacturing processes, such as bottle production. It addresses real-time control challenges, aiming to improve energy efficiency and product quality through advanced supervisory control systems.

However, this study does not provide a solution to our problem. It primarily targets industrial control rather than real-time graphical visualization. The FEM models discussed are computationally intensive and unsuitable for interactive simulations. Additionally, the paper does not explore techniques like raymarching or implicit surfaces, which are crucial to our project's goal of simulating molten glass dynamically and realistically.

Références

[1] Bruce Naylor. Constructing good partitioning trees. *Graphics Interface*, September 2001.

[2] Floney Yang. Raymarching collision, November 2024.

[3] Hecomi. Raymarching collision project (github), November 2024.

[4] Sakri Koskimies. Raymarching engine (github), November 2024.

[5] Adrian Biagioli. Raymarching presentation, November 2024.

[6] Nabil Mansour. SDF raymarching, November 2024.

[7] Wojciech Grega, Adam Pilat, and Andrzej Tutaj. Modelling of the glass melting process for real-time implementation. *International Journal of Modeling and Optimization*, 5(6) :366–373, 2015. Number : 6.