GPU BASED REAL-TIME WELDING SIMULATION WITH SMOOTHED-PARTICLE HYDRODYNAMICS

Qing Gu

A thesis
in
The Department
of
Computer Science

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Computer Science
Concordia University
Montréal, Québec, Canada

January 2016

Concordia University
School of Graduate Studies

This is to certify that the thesis prepared

By: **Qing Gu**

Entitled: **GPU Based Real-time Welding Simulation with Smoothed-Particle Hydrodynamics**

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

Dr. Nikolaos Tsantalis Chair

Dr. Adam Krzyzak Examiner

Dr. Charalambos Poullis Examiner

Dr. Tiberiu Popa Supervisor

Dr. Sudhir Mudur Co-supervisor

Approved _____
Chair of Department or Graduate Program Director

_____ 20 _____

Amir Asif, Ph.D..P.Eng., Dean
Faculty of Engineering and Computer Science

Abstract

GPU Based Real-time Welding Simulation with Smoothed-Particle Hydrodynamics

Qing Gu

Welding training is essential in the development of industrialization. A good welder will build robust workpieces that ensure the safety and stability of the product. However, training a welder requires lots of time and access professional welding equipment. Therefore, it is desirable to have a training system that is economical and easy to use. After decades development of computer graphics, sophisticated methodologies are developed in simulation fields, along the advanced hardware, enables the possibility of simulation welding with software. In this thesis, a novel prototype of welding training system is proposed. We use smoothed-particle hydrodynamics (SPH) method to simulate fluid as well as heat transfer and phase changing. In order to accelerate the processing to reach the level of real-time, we adopt CUDA to implement the SPH solver on GPU. Plus, Leap Motion is utilized as the input device to control the welding gun. As the result, the simulation reaches decent frame rate that allows the user control the simulation system interactively. The input device permits the user to adapt to the system in less than 5 minutes. This prototype shows a new direction in the training system that combines VR, graphics, and physics simulation. The further development of VR output device like Oculus Rift will enable the training system to a more immersive level.

Acknowledgments

Firstly, I would like to thank my supervisors. It's their help and professional support allow me to finish this thesis. I also would like to thank my family for their unconditional love. Finally, I would like to thank all of my friends for their support.

Contents

List of Figures

Chapter 1

Introduction

Welding is a fabrication process that fuses metals together. The fusing is caused by melting the base metal. Usually, a filler material is injected into the joint to form a weld pool. By cooling off, the filled joint will have the same durability as the base metal. There are many welding methods exist, for instance, shielded metal arc welding, gas tungsten arc welding, gas metal arc welding, etc. Welding plays a significant role in the industry. Essentially, Some of the fuselage and the chassis components are fabricated with tungsten inert gas (TIG) welding. The quality of welding has a direct influence on the user's safety. In contrast to the importance of welding, welding training and certification is not accessible for most of the people due to the expense of equipment and resources like electricity and gas. Also, for beginners, it's not safe to practice welding without supervising. Therefore, welding student can only practice their skills at certain facilities. To address this problem, people start to seek methods in computer simulations. For instance, 123 Certicification Inc. in Montreal has developed an Arc+ welding simulator which is a simulation machine including both software and hardware. But their equipment is expensive and simulation is based on experimental data.

There exist certain welding simulation methods. Some of the studies [TMZ+11, Faw13] concentrate on finite element models. In these models, workpieces are considered as solid metals and be separated into small finite elements. Heat transfers are considered occurring between the finite elements. This method is mainly used to investigate the heat transfer, but not the deformation of the workpieces. Recently, there are several studies focus on using fluid simulation [RC09,IIFS12]. In these papers, fluid simulation is used to simulate the molten filler materials. However, both of them are single-phase simulations. Although filler material is modeled as liquid particles whose viscosity changes with temperature, they will not solidify even if their temperature is lower than freezing point. The parent material does not melt when the temperature is higher than the melting point. Another crucial issue is that the simulation is too slow to satisfy the real-time application.

Phase changing is discussed in [SSP07], in their method, they talked about the 3 states of particles: liquid, the elastic body, and rigid body. The last two phases require cold particles to form certain reference bodies in order to calculate the elastic force or sum of torque. Still, this scheme is difficult to combine with parallel scheme. Also, regarding the fluid simulation, several methods are proposed to solve smoothed-particle hydrodynamics (SPH) in real-time. Most of them adopted

GPU computation, e.g. [GSSP10,IABT11]. GPU was originally designed for 3D graphics rendering, which contains much more threads than CPU. Different from CPU which is designed as MIMD processor to handle different tasks in the operating system, GPU has the SIMD architecture that is suitable for rendering vertices and pixels. Since in graphics rendering, we just perform the same operations on the given elements, e.g. multiply each vertex by model, view, and projection matrices, applying a gaussian filter to rendered frame buffer, etc. These implementations mentioned in the papers achieved high frame rate, but not no heat transfer nor multiphase were discussed.

Furthermore, several studies focus on generating better simulation result using iterative pressure solver to ensure incompressibility, for instance, [SP09, SSP07]. Also, an implicit viscosity solver is proposed this year by [PICT15] to generate better result in a non divergence free velocity field. However, the implementations of these solvers are all CPU based and requires more iteration, implementations with GPU or real-time applications are barely introduced.

This thesis focuses on welding simulation based on SPH and CUDA. SPH was originally introduced by [GM77] for astrophysical problems but further modified to simulate fluid. This method smoothed the effect of particles by a kernel function, allowing simulating fluid with a limited number of particles by giving the interpolation equation:

$$A(\mathbf{r}) = \sum_j m_j \frac{A_j}{\rho_j} W(|\mathbf{r} - \mathbf{r_j}|, h)$$

where A is the quantity (a scalar or a vector) of a particle, $\mathbf{r}$ is the position of the particle and $\mathbf{r_j}$ is the position of a neighboring particle. W is a kernel function that smooths the contribution of neighbor particles. By implementing this solver with CUDA, we achieved real-time fluid simulation with high granularity. Furthermore, we incorporate a heat transfer scheme and phase changing methods to this solver to simulate the solidification and melting of metals. Moreover, we adopted Leap Motion as the input method to control the welding gun. Leap Motion is a finger tracker that can capture the position and directions of fingers in very high refreshing rate. Comparing to the marker method in Arc+, this method is more accessible and flexible.

There are following contributions in this thesis:

- We achieved real-time fluid simulation based welding simulation with GPU.
- We incorporated heat transfer and phase changing scheme that compatible with our system.
- We developed a decent rendering method to render metal particles.
- A Python console is embedded into the system to control simulation system at runtime.
- We adopted Leap Motion as an intuitive and low cost input method.
- We implemented a mesh importer that allows user to use the customized meshes as workpieces.
- The system is designed to be easily incorporated with VR goggles.

Chapter 2

Related Work

In this section, related works will be listed in the following fashion. Firstly, we will introduce the background of fluid dynamics as well as the derivation of Naver-Stokes equation of incompressible fluid. Then the method of smoothed hydrodynamic particles is introduced. After that, we will focus on the developing of SPH in the past decades and state-of-the-art methods, mostly on correction of incompressibility and viscosity. Also, techniques of user interface improvement based on the modern hardware are investigated.

2.1 Fluid Dynamics and Navier-Stokes Equation

Fluid simulation is a physics simulation process employed fluid dynamics equations, mostly Navier-Stokes equation. Navier-Stokes equation describes the acceleration of fluid particles regarding their hydrodynamics pressures, viscosity forces and external body forces. Having a good understanding of Navier-Stokes equation will contribute to better implementation of the simulation system. This section will give a brief introduction of the basic knowledge of fluid dynamics as well as the derivation of Naver-Stokes equation.

2.1.1 Lagrangian vs. Eulerian Specification

In the Lagrangian specification, fluids are treated as small particles move with flow, observer focus on the movement of a particular particle, and moves with it. In this case, the translation vector, i.e. the position, can be represented as:

$$\mathbf{r} = \mathbf{r}(a, b, c, t)$$

Where $(a, b, c) = \mathbf{r}(a, b, c, t_0)$ is the Lagrangian variable. It denotes the position of the particle at starting point t_0. Thus, different particles can be labeled by their starting positions. The velocities and accelerations are the first and second derivative of $\mathbf{r}$. Note that the acceleration is only the function of t, thus they can be given as:

$$\mathbf{u} = \frac{d\mathbf{r}}{dt} = \nabla \mathbf{r}$$

$$\mathbf{a} = \frac{d\mathbf{u}}{dt} = \nabla(\nabla(\mathbf{r}))$$

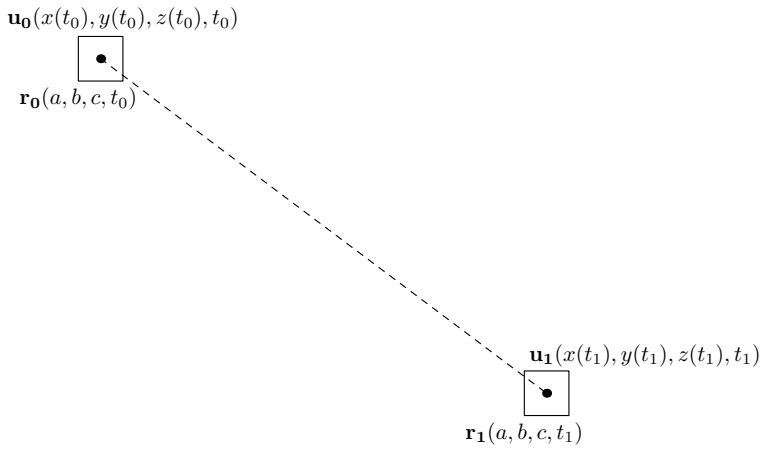

Figure 1: From t_0 to t_1, in the Lagrangian specification, the position of the point is only the function of time. However, in Eulerian specification, the particles observed at different positions, represent as the squres in the figure. Thus, the velocity of fluid in the small space is function of time.

As shown in figure 1, while Lagrangian specification focus on the position of particles, Eulerian specification focuses on the velocity of particles. It is a method to study the velocity field of fluids. Observer focus on a particular space, and observes the fluid particles flow through it. In Eularian specification, the velocity vector of a particle is given as:

$$\mathbf{u} = \mathbf{u}(x, y, z, t)$$

where (x, y, z) is a small volume in space. Different from Lagrangian specification, x, y, z are all functions of time t. Thus, by given the speed as $\frac{d(x,y,z)}{dt} = (u_x, u_y, u_z)$, the acceleration can be represented as the full derivative of velocities:

$$\mathbf{a} = \frac{D\mathbf{u}}{Dt} = (\frac{\partial}{\partial t} + \mathbf{u} \cdot \nabla)\mathbf{u}$$

This derivative is also called material derivative. This equation indicates that the change of velocity has been composed by two part: the local velocity change in time dt and the velocity change casued by the difference of velocity in different positions. The Eulerian specification is widely used in fluid dynamics analysis. The Navier-Stokes equation is also derived by using this specification.

2.1.2 Contniuum Equation

Continuum Equation is broadly used in physics to represent transportation of materials. The continuum equation of fluid can be derived by applying conservation of mass. According to the principle of mass conservation, the pure mass input of the control volume should be equal to the increment of the density of the control volume.

Figure 2 shows a control volume, and u_x is the speed of flow. On the x direction, the input of mass of the control volume in time dt is $\rho u_x dydzdt$. Applying the Taylor approximation, at the

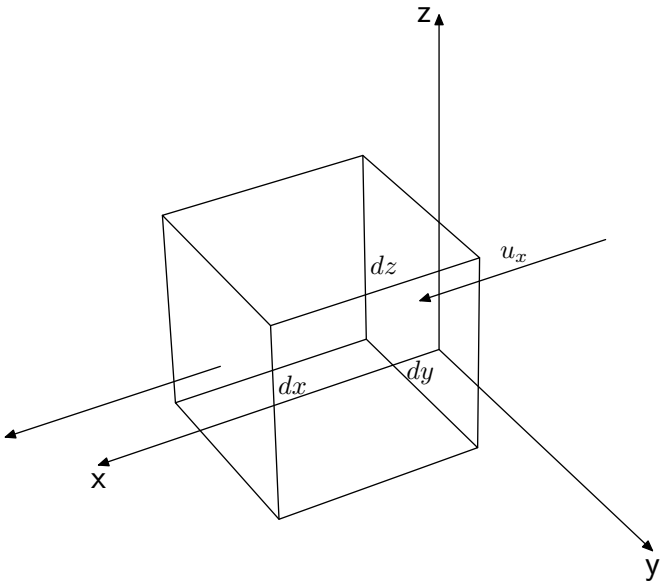

Figure 2: Control Volume

same time, we can get the output of the mass is $(\rho u_x + \frac{\partial(\rho u_x)}{\partial x}dx)dydzdt$. Thus the pure input mass in dt is:

$$\rho u_x dydzdt - (\rho u_x + \frac{\partial(\rho u_x)}{\partial x}dx)dydzdt = -\frac{\partial(\rho u_x)}{\partial x}dxdydz$$

The pure input mass is calculated by the different of input mass and output mass, considering the mass input on x, y, z directions, we have:

$$\frac{\partial \rho}{\partial t} + \frac{\partial(\rho u_x)}{\partial x} + \frac{\partial(\rho u_y)}{\partial y} + \frac{\partial(\rho u_z)}{\partial z} = \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \tag{1}$$

where $\frac{\partial \rho}{\partial t}$ is the increment of density. The density of incompressible fluid is constant. Based on Euler specification, the material derivative of density should be zero, i.e. $\frac{D\rho}{Dt} = 0$. Plug it into (1), the continuum equation for incompressible fluid is given as:

$$\nabla \cdot \mathbf{u} = 0.$$

2.1.3 Stress and Strain

Strain is a value that measures the deformation of object comparing to its original position and stress is the force that causes the deformation. Consider the 2D situation in figure3 on $x - y$ plane, a square control volume a, b, c, d is deformed to new shape a', b', c', d' in time t to $t + \Delta t$. On x axis, the linear strain rate can be represented by $\varepsilon_{xx} = \frac{\partial u_x}{\partial x}$ and angular strain rate can be represented by $\varepsilon_{xy} = \frac{1}{2}(\frac{\partial u_y}{\partial x} + \frac{\partial u_y}{\partial x})$. Simply expand this to the other directions in 3D, we can define strain the strain tensor:

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} & \varepsilon_{xz} \\ \varepsilon_{yx} & \varepsilon_{yy} & \varepsilon_{yz} \\ \varepsilon_{zx} & \varepsilon_{zx} & \varepsilon_{zz} \end{bmatrix} = \begin{bmatrix} \frac{1}{2}(\frac{\partial u_x}{\partial x} + \frac{\partial u_x}{\partial x}) & \frac{1}{2}(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}) & \frac{1}{2}(\frac{\partial u_x}{\partial z} + \frac{\partial u_z}{\partial x}) \\ \frac{1}{2}(\frac{\partial u_y}{\partial x} + \frac{\partial u_x}{\partial y}) & \frac{1}{2}(\frac{\partial u_y}{\partial y} + \frac{\partial u_y}{\partial y}) & \frac{1}{2}(\frac{\partial u_y}{\partial z} + \frac{\partial u_z}{\partial y}) \\ \frac{1}{2}(\frac{\partial u_z}{\partial x} + \frac{\partial u_x}{\partial z}) & \frac{1}{2}(\frac{\partial u_z}{\partial y} + \frac{\partial u_y}{\partial z}) & \frac{1}{2}(\frac{\partial u_z}{\partial z} + \frac{\partial u_z}{\partial z}) \end{bmatrix} \tag{2}$$

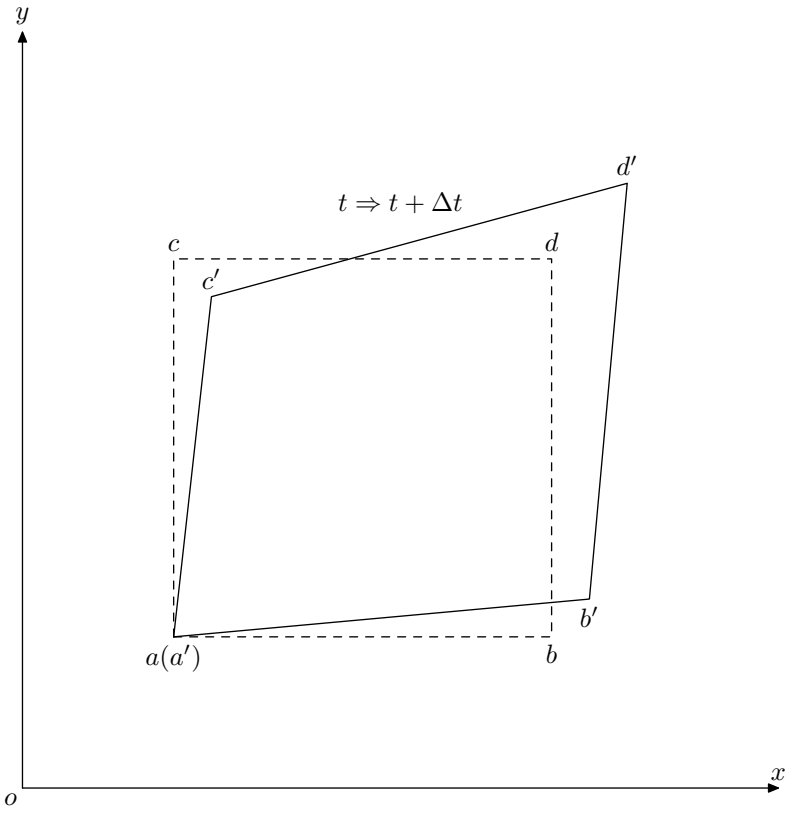

Figure 3: Square a, b, c, d is deformed to a', b', c', d' from t to $t + \Delta t$

In which the three elements on the diagonal are the linear strain rate and the others are angular strain rate.

As shown in figure 4, we use P_{ab} to denote the component of stress that applied on the face whose normal is a and stress direction is b. For instance, P_{xy} denotes the y component of stress applied to the surface whose normal is x. By putting them together, we can have the following stress tensor, given as:

$$\mathbf{P} = \begin{bmatrix} P_{xx} & P_{xy} & P_{xz} \\ P_{yx} & P_{yy} & P_{yz} \\ P_{zx} & P_{zy} & P_{zz} \end{bmatrix}.$$

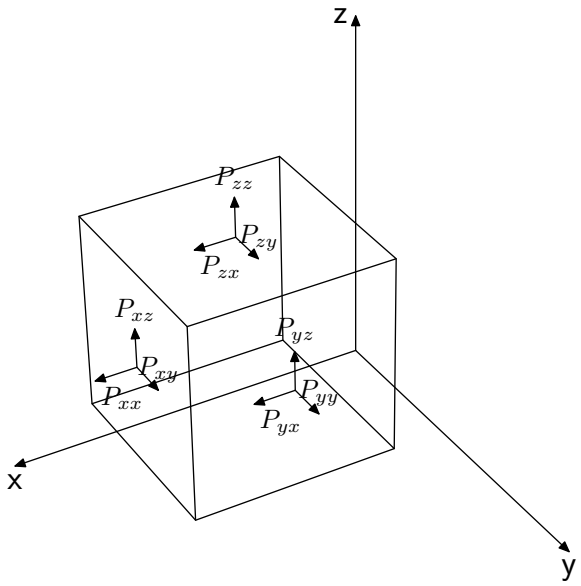

Figure 4: The stresses on the control volume. Stress is decomposed into 6 components indicated in the figure.

The values on the diagonals are orthogonal normal stresses and the others are orthogonal shear stresses. The common part $p = \frac{1}{3}(P_{xx}, P_{yy}, P_{zz})$ is extracted. Thus, the stress tensor can be rewritten as the linear combination of the following two parts:

$$\mathbf{P} = \begin{bmatrix} P_{xx}+p & P_{xy} & P_{xz} \\ P_{yx} & P_{yy}+p & P_{yz} \\ P_{zx} & P_{zy} & P_{zz}+p \end{bmatrix} - p\mathbf{I} = \mathbf{D} + p\mathbf{I} \tag{3}$$

Where $\mathbf{D} = \begin{bmatrix} P_{xx}+p & P_{xy} & P_{xz} \\ P_{yx} & P_{yy}+p & P_{yz} \\ P_{zx} & P_{zy} & P_{zz}+p \end{bmatrix}$ is caused by viscous, let's call it viscous stress tensor. $\mathbf{I}$ is the unit tensor and p is called hydrodynamic pressure and irrelevant to the direction.

2.1.4 Newton's Law of Friction

Viscosity is caused by friction between particles, generally speaking, the friction force is proportional to the differential of speed, i.e. the viscous stress tensor is proportional to strain tensor. In 1 dimensional situation, it can be represented as $\tau = \mu \frac{du}{dy}$. By using the strain tensor (2) and stress tensor (3) above, the Newton's law of friction in 3D can be represented by $\mathbf{D} = 2\mu\boldsymbol{\varepsilon}$, by plugging it into stress tensor (3), we have:

$$\mathbf{P} = 2\mu\boldsymbol{\varepsilon} + p\boldsymbol{\delta} \tag{4}$$

It is safe to say that Newton's law of friction served as the glue between stress tensor and strain tensor. Also note that here we assume that the viscous stress tensor is linearly proportional to strain tensor. This assumption, unfortunately, is not always true. Numbers of fluids satisfy Newton's law of friction, for instance, water, gases and molten metals. We call these fluids Newtonian fluids. On the other hands, there are also a lot of other kinds of fluids. Figure5 lists different kinds of fluids. Though, our discussion will focus on Newtonian fluids.

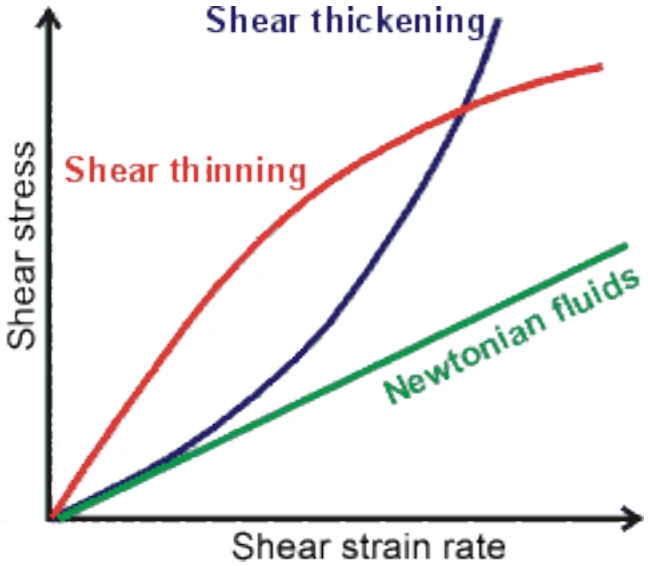

Figure 5: The relationship between shear stress and shear strain rate of different fluids.

2.1.5 Navier-Stokes Equation

Navier-Stocks equation is a continuum equation that describes the movement of the fluid. It describes the relationship among external forces, viscosity force and pressure force. According to the conservation of momentum, the increment of the momentum should be equal to the momentum flows into the control volume plus the momentum contributed by external forces(both body forces and surface forces), given as:

$$\mathbf{p_{increment}} = \mathbf{p_{flow}} + \mathbf{p_{bodyforce}} + \mathbf{p_{surfaceforce}}. \tag{5}$$

We will first discuss this equation on x direction, then expand to the tensor form in 3 dimension. Firstly, according to the equation of momentum $\mathbf{p} = m\mathbf{v}$, the increment of momentum in dt on x

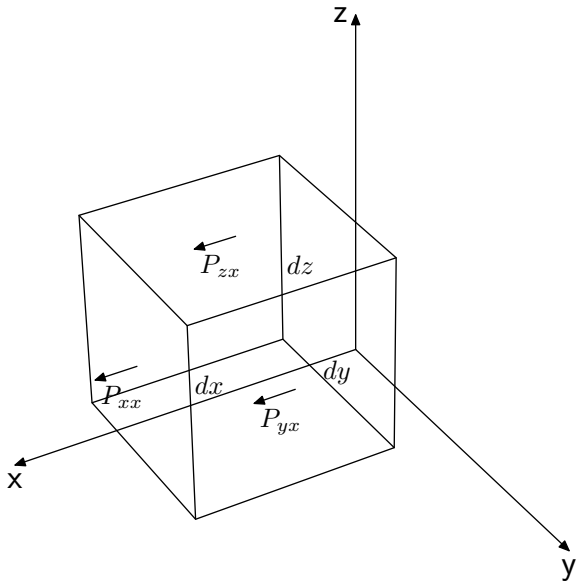

Figure 6: Control volume with stresses on x direction.

direction is:

$$\mathbf{p_{increment}} = \frac{\partial(\rho u_x)}{\partial t} dxdydz \tag{6}$$

On the right side of equation 5, we have the momentum flows into the control volume:

$$\mathbf{p_{flow}} = (-\frac{\partial(\rho u_x u_x)}{\partial x} - \frac{\partial(\rho u_y u_x)}{\partial y} - \frac{\partial(\rho u_z u_x)}{\partial z})dxdydz \tag{7}$$

Note that we need to consider all of the three surfaces in figure 6.

Then, the density contributed by the body force on x direction is easy to derive:

$$\mathbf{p_{bodyforce}} = \rho X dxdydz \tag{8}$$

where X is the body force on x direction.

The surface forces on x direction need also to take account of all three surfaces:

$$\mathbf{p_{surfaceforce}} = (\frac{\partial P_{xx}}{\partial x} + \frac{\partial P_{yx}}{\partial y} + \frac{\partial P_{zx}}{\partial z})dxdydz \tag{9}$$

By plugging all of the four components into (5), we will have the momentum conservation equation on x direction:

$$\begin{aligned} \frac{\partial(\rho u_x)}{\partial t} dxdydz - (-\frac{\partial(\rho u_x u_x)}{\partial x} - \frac{\partial(\rho u_y u_x)}{\partial y} - \frac{\partial(\rho u_z u_x)}{\partial z})dxdydz = \\ \rho X dxdydz + (\frac{\partial P_{xx}}{\partial x} + \frac{\partial P_{yx}}{\partial y} + \frac{\partial P_{zx}}{\partial z})dxdydz \\ \Rightarrow \frac{\partial(\rho u_x)}{\partial t} + (\frac{\partial(\rho u_x u_x)}{\partial x} + \frac{\partial(\rho u_y u_x)}{\partial y} + \frac{\partial(\rho u_z u_x)}{\partial z}) = \\ \rho X + (\frac{\partial P_{xx}}{\partial x} + \frac{\partial P_{yx}}{\partial y} + \frac{\partial P_{zx}}{\partial z}) \end{aligned}$$

By plugging in continuum equation (1):

$$\frac{du_x}{dt} = X + \frac{1}{\rho}(\frac{\partial P_{xx}}{\partial x} + \frac{\partial P_{yx}}{\partial y} + \frac{\partial P_{zx}}{\partial z}) \tag{10}$$

Now, it's time to expand it to other directions:

$$\frac{du_y}{dt} = X + \frac{1}{\rho}(\frac{\partial P_{xy}}{\partial x} + \frac{\partial P_{yy}}{\partial y} + \frac{\partial P_{zy}}{\partial z})\frac{du_z}{dt} = X + \frac{1}{\rho}(\frac{\partial P_{xz}}{\partial x} + \frac{\partial P_{yz}}{\partial y} + \frac{\partial P_{zz}}{\partial z}) \tag{11}$$

Still on x direction, we use Newton's fraction law to substitute stresses with strains:

$$\frac{du_x}{dt} = X - \frac{1}{\rho}\frac{\partial p}{\partial x}v(\frac{\partial^2 u_x}{\partial x^2} + \frac{\partial^2 u_x}{\partial y^2} + \frac{\partial^2 u_x}{\partial z^2}) \tag{12}$$

Where $p = \frac{1}{3}(P_{xx}, P_{yy}, P_{zz})$ is the hydrodynamic pressure mentioned before. Now, writing it in tensor form, we will have the Navier-Stokes equation:

$$\frac{d\mathbf{u}}{dt} = \mathbf{f} - \frac{1}{\rho}\nabla p + v\nabla^2\mathbf{u} \tag{13}$$

Navier-Stokes equation is a partial derivative equation can be solved under few conditions. Nevertheless, our implementation requires solution under all conditions, thus, only numerical solutions exist.

2.1.6 Surface Tension

Apart from internal forces and external forces we have discussed, surface tension also contributes to the movement of particles, and only exists on the free surfaces.

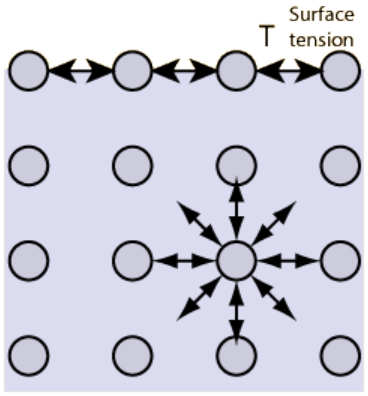

Figure 7: Surface Tension.

Surface tension is generated by the cohesive forces. Cohesive forces are attract force between particles. It's balanced inside the fluid. Because on each direction, the mass of fluid are the same. However, for particles on the free surfaces, because there is only one side has particles, the cohesive force is unbalanced, which causes the phenomenon of surface tension. In physics, the surface tension is described as the force that perpendicular to the length and tangent to the free surface. The force can be represented as $\sigma = \Delta T/\Delta L$.

2.2 SPH

Since we are using Eulerian specification, the conservation of mass is guaranteed, thus, SPH uses the following Navier-Stocks Equation.

$$\rho\frac{\partial \mathbf{u}}{\partial t} = -\nabla p + \mu\nabla^2\mathbf{u} + \rho\mathbf{g} \tag{14}$$

Where $\mathbf{u}$ is the velocities of particles, μ is the stiffness parameter, ρ is the densities of particles. The 3 parts on the right side of the equation represents the pressure force density, viscosity force density and external force density which are needed to be solved one by one. Navier-Stokes equation is established on the continuum hypothesis. With this assumption, the properties of fluid like densities, velocities, accelerations are well-defined at infinitely small space. In fact, fluids are made of molecules, for example, the molar mass of water is $18.01528g/mol$, which means that there are $6.02214129(27) \times 10^{23}$ molecules in $18.01528g$ of water. Each molecule is so small such that we can approximately consider it infinitely small.

In this following sections, the basic SPH implementation is firstly introduced. Then, the variations of SPH methods are presented. The variations mostly focus on developing more stable pressure solver and viscosity solver.

2.2.1 Interpolation

One issue arises at fluid simulation is granularity. Due to the limitation of computational power, fluids are impossible to be simulated at the molecule level. That is to say, we need certain interpolation methods to integrate numbers molecules into a bigger particles without significant impact on the accuracy of the simulation. One method proposed in [GM77] involves the following summation equation. With this equation, the effect of a particle is smoothed within its support radius.

$$A(\mathbf{r}) = \sum_j m_j \frac{A_j}{\rho_j} W(|\mathbf{r} - \mathbf{r_j}|, h) \tag{15}$$

In which W is the kernel function and h is the smooth distance. The kernel function W defines the influence a particle can have to another by giving their distance r.

2.2.2 Densities

The first step of SPH is to calculate the densites ρ using summation equation, plug in ρ into eq15, we can get the density of i^th particle.

$$\rho_i = \sum_j m_j W(\mathbf{x_{ij}}, h) \tag{16}$$

With densities of all particles that calculated by the the equation above, we can start to calculate the pressure force p using the following equation.

2.2.3 Pressures

Hydrodynamics pressure is formed by the average of the orthogonal pressure and irrelevant to the direction. Thus, we can define it as a scalar. In order to calculate the gradient of pressure force of each particle, we need to compute the pressure at each point according to the density computed by (16). [DG96] provides a method as flowing equation based on idea gas law:

$$p_i = k(\rho_i - \rho_0), \tag{17}$$

Where k is the stiffness parameter, ρ_i is the density of i^{th} particle given by (16) and ρ_0 is the rest density of fluid. (e.g. rest density of water is $1g/cm^3$). In order to maintain incompressibility, a large k is required.

Besides the ideal gas equation, Tait equation that was published by Peter Guthrie Tait in 1888 is also used to simulate the weakly compressible liquids [BT07]. In this approach, the pressure of the particle can be written as:

$$P = B\left((\frac{\rho}{\rho_0})^\gamma - 1\right) \tag{18}$$

Where $\gamma = 7, B$ are constants govern the density fluctuation $\frac{|\delta\rho|}{\rho_0}$;

Then apply pressure forces calculated from (17) or (18) to the gradient of summation equation (15), we will have.

$$-\nabla p = -\sum_j \frac{m_j}{\rho_j} p_j \nabla W_{ij} \tag{19}$$

Note that the pressure force density calculated in (19) is not symmetric, resulting in violation of the Newton's 3rd law. Thus after symmetrization, we can get the final equation that used to calculate the pressure force density.

$$\nabla p = -\sum_j \frac{m_j}{\rho_j} \frac{p_i + p_j}{2} \nabla W_{ij} \tag{20}$$

The velocity computed above is not guaranteed to be divergence free. Therefore, other methods e.g. [SP09, HLL$^+$12] are developed to add a correction step. In these methods, an intermediate velocity is calculated without effects of pressure force. Based on the intermediate velocity, an offset to the density is applied, giving the new density as:

$$\rho_i^* = \sum_j m_j W_{ij} + \Delta t \sum_j (\mathbf{v}_i^* - \mathbf{v}_j^*) \cdot \nabla W_{ij}. \tag{21}$$

Where $\mathbf{v}_i^*, \mathbf{v}_j^*$ are velocities of particles, Δt is the length of time step and ρ_i^* is the corrected density. Then based on the corrected density, the pressure force is calculated, giving fluid a more divergence free velocity field.

Though non-iterative EOS methods yield good results, iterative methods are developed to further enforcing incompressibility, e.g. PCISPH [SP09] and LPSPH [HLL$^+$12]. In these methods velocity is firstly calculated without the effect of pressure force. Then, the position is updated by the

velocity repeatedly. In the iteration part, density ρ^* is updated by new position derived from density calculated in the previous iteration until $\rho_{err} = \rho^* - \rho_0$ is smaller than a threshold η. These methods provide a flexibility on controlling the compressibility as well as allowing bigger time step.

Another methods emerged recently is to solve pressure force with a pressure Poisson equation (PPE), e.g. IISPH [ICS+14]. In these methods, Laplacian of pressure is written as [CR99, PTB+03]

$$\nabla^2 p_i = \frac{\rho_0}{\Delta t} \nabla \cdot \mathbf{v}_i^*. \tag{22}$$

or as [SL03, KGS09]:

$$\nabla^2 p_i = \frac{\rho_0 - \rho_i^*}{\Delta t^2} \tag{23}$$

However, PPEs need to be solved with successive over-relaxation, conjugate gradient or multigrid techniques, resulting in more iterations.

2.2.4 Viscosities

Viscosity is introduced to stabilize the velocity field of the particle system. The velocities of particles are diffused to neighbor particles which makes particles move in groups. Using the second part on the right of (14), the viscosity force density is given by using the Laplacian of velocity field in the following form. [MCG03]

$$\mu \nabla^2 \mathbf{u} = \mu \sum_j \frac{m_j}{\rho_j} \mathbf{u_j} \nabla^2 W_{ij} \tag{24}$$

However, the equation above also suffers from the same issue of non-symmetric. After symmetrization, we have:

$$\mu \nabla^2 \mathbf{u} = \mu \sum_j (\mathbf{u_j} - \mathbf{u_i}) \nabla^2 W_{ij} \tag{25}$$

The method above computes the viscosity explicitly by plugging in Newton's law of friction to convert stress into strain. Another method to solve viscosity implicitly is proposed by [PICT15]. The main idea is to decompsite the velocity gradient into three parts: spin tensor R, expansion-rate tensor V and shear-rate tensor S in the following fashion:

$$\nabla \mathbf{v} = \tag{26}$$

$$\frac{1}{2}(\nabla \mathbf{v} - (\nabla \mathbf{v})^T) \qquad \mathbf{D} \tag{27}$$

$$+\frac{1}{3}(\nabla \cdot \mathbf{v})\mathbf{I} \qquad \mathbf{V} \tag{28}$$

$$+(\frac{1}{2}(\nabla \mathbf{v} + (\nabla \mathbf{v})^T) - \frac{1}{3}(\nabla \cdot \mathbf{v})\mathbf{I}) \qquad \mathbf{S} \tag{29}$$

Note that the speed gradient $\nabla \mathbf{v}$ is a second rank tensor given as:

$$\nabla \mathbf{v} = \begin{bmatrix} \frac{\partial v_x}{\partial x} & \frac{\partial v_x}{\partial y} & \frac{\partial v_x}{\partial z} \\ \frac{\partial v_y}{\partial x} & \frac{\partial v_y}{\partial y} & \frac{\partial v_y}{\partial z} \\ \frac{\partial v_z}{\partial x} & \frac{\partial v_z}{\partial y} & \frac{\partial v_z}{\partial z} \end{bmatrix} \tag{30}$$

The viscosity is controlled by a parameter $\xi \in [0, 1]$ applied on shear-rate tensor S. Rewrite (30) to:

$$\nabla \mathbf{v} = \mathbf{R} + \mathbf{V} + \sigma \mathbf{S} \tag{31}$$

When $\xi = 0$, no shear force is applied to the particles, particle will remain the shape. When $\xi = 1$, the system has no resist to the shear force, resulting in fluid with no viscosity, Figure 8 shows the different result by setting $\xi = 0.8$ and $\xi = 0.2$.. Additionally, when the density is smaller than the rest density, the pressure solver will generate attraction force which causes instability. This implicit viscosity model does not address this problem, but stops worsen it by removing the expansion rate tensor $\mathbf{V}$, yields:

$$\nabla \mathbf{v} = \mathbf{R} + \xi \mathbf{S} \tag{32}$$

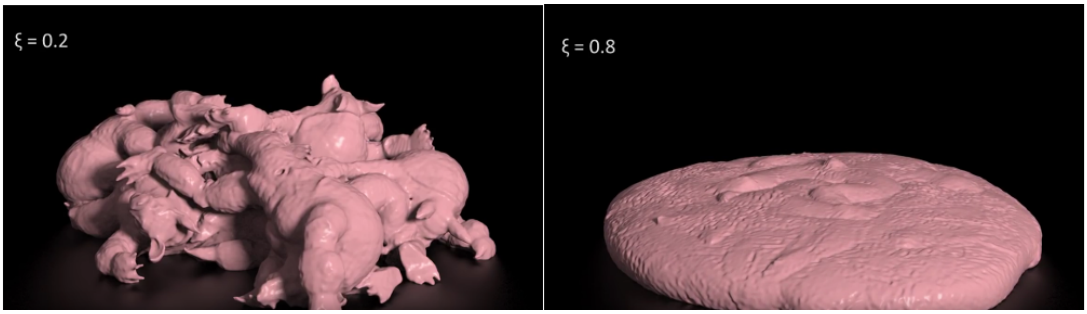

Figure 8: On the left, when ξ is small, fluid is more viscous than the one with higher ξ on the right.

2.2.5 Surface Tension

As mentioned above, surface tension is caused by cohesion forces between particles. The particles on the free surfaces are affected by cohesion forces from only one side, resulting in unbalanced composition force that stops them from moving away.

The early researches focus on minimizing the surface curvatures [Mor00]. In these methods, a color function is given to describe the shape of the fluid.

$$c = \sum_j \frac{m_j}{\rho_j} \mathbf{W}(r - r_j) \tag{33}$$

The gradient of the color function is used to define the direction of force direction.

$$\mathbf{n} = \sum_j \frac{m_j}{\rho_j} \nabla \mathbf{W}(r - r_j) \tag{34}$$

Since the effect of the surface tension is to minimize the curvature of surfaces, we can use the Laplacian of the color function to describe the strength of the force. Thus, the surface tension is given by the following equation.

$$f_t = -k \nabla^2 c \frac{\mathbf{n}}{|\mathbf{n}|} \tag{35}$$

Note that in order to apply forces on surface only, a threshold t is given and the surface tensions are applied when and only when $|\mathbf{n}| > t$.

This method, though widely used, have certain issues. Firstly, inner particles have arbitrary directions. Though it can be solved by applying a threshold, the choice of threshold is very important to the stability of simulation. Secondly, this model is very sensitive to SPH disorder since it highly relies on the contribution of neighbor particles while calculating the Laplacian. Lastly, the external forces that applied on the surface violate the conservation of momentum.

An important method is proposed by [IIFS12] which provides a surface tension combined cohesion force and tangent force. The cohesion force is calculated by:

$$\mathbf{F}_a^{attract} = d \sum_b \gamma' f_{ab}^{attract} \tag{36}$$

where d is the particle diameter, γ' is a constant computed from surface tension coefficient γ, and $f_{ab}^{attract}$ is a kernel function given as:

$$f_{attract}(|r_a - r_b|, h) = \begin{cases} q & (0 \leq q < 1) \\ 2 - q & (1 \leq q < 2) \\ 0 & (2 \leq q) \end{cases} \tag{37}$$

Tangent force is calculated by the following equation:

$$F^{tangent} = \frac{d\gamma}{dT} d \cdot n \times \nabla T \times n, \tag{38}$$

where n is the surface normal vector and $\frac{d\gamma}{dT} = 0.21 mN/mK$ is a constant.

While calculating the tangent force, it takes account of the gradient of temperature, makes it more suitable for SPH system with heat transferring. However, it also has issues on the attraction force. The kernel function is a two segments of linear function, causes artifacts of particles on the surfaces.

[AAT13] proposed a versatile solution of surface tension that combined cohesion term and surface area minimization term, which generates stable and momentum conserve surface tension. According to this method, cohesion term is defined as:

$$\mathbf{F}_{i \leftarrow j}^{cohesion} = -\gamma m_i m_j C(|\mathbf{x}_i - \mathbf{x}_j|) \frac{\mathbf{x}_i - \mathbf{x}_j}{|\mathbf{x}_i - \mathbf{x}_j|}$$

where i, j are neighboring particles, m is mass and $\mathbf{x}$ denotes the positions. C is a spline function given as:

$$C(r) = \frac{32}{\pi h^9} \begin{cases} (h-r)^3 r^3 & 2r > h \wedge r \leq h \\ 2(h-r)^3 r^3 - \frac{h^6}{64} & r > 0 \wedge 2r \leq h \\ 0 & otherwise \end{cases}$$

Also, the surface area minimization term is calculated by using neighbor's smoothed color field calculated by :

$$\mathbf{n} = h \sum_j \frac{m_j}{\rho_j} \nabla \mathbf{W}(r - r_j). \tag{39}$$

Here, an parameter h is added to make the computed normal scale independent. The symmetric curvature force is given as:

$$\mathbf{F}_{i\leftarrow j}^{curvature} = -\gamma m_i(\mathbf{n}_i - \mathbf{n}_j).$$

A correction factor is calculated by the following equation:

$$K_{ij} = \frac{2\rho_0}{\rho_i + \rho_j}$$

Finally, those two terms can be combined together as:

$$\mathbf{F}_{i\leftarrow j} = K_{ij}(\mathbf{F}_{i\leftarrow j}^{cohesion} + \mathbf{F}_{i\leftarrow j}^{curvature})$$

2.2.6 Heat Transfer

Heat transfer is essential in welding simulation. Hot particles injected into system transfer their thermal energy to the surrounding particles, resulting in temperature and phase changing. The temperature also contribute to the change of the viscosity. The general form of thermal equation is given as:

$$c_p \frac{dT}{dt} = \frac{1}{\rho}\nabla(k\nabla T), \tag{40}$$

where T is the absolute temperature, c_p is the heat capacity per unit mass at constant pressure, ρ is the density and k is the conductivity. Monaghan and Cleary [CM99] proposed the SPH heat conduction by using integral approximants as:

$$\frac{dU_a}{dt} = \sum_b \frac{4m_b}{\rho_a\rho_b}\frac{k_a k_b}{k_a + k_b}T_{ab}F_{ab}, \tag{41}$$

where $F = \mathbf{q}\cdot\nabla W(\mathbf{q})$ and $\nabla W(\mathbf{q})$ is the gradient of kernel function. The heat conduction equation in (41) ensures that heat flux is automatically continuous across material interfaces.

2.2.7 Neighborhood Search

SPH computation requires a huge amount of computation between neighbor particles. Without neighboring search, the time complexity of searching over all particles is $O(n^2)$. However, because the kernel functions only take account of the neighbor particles within the support radii, narrowing down the searching space will boost the processing speed significantly. A commonly used technique is uniform grid which separates space into small cubic cells with length equal to the size of support radii. As depicted in figure 9 Particles are inserted into cells, and the one particle has effects to the particles that reside in cells nearby.

Anther method used is hashing. It is broadly used in SPH system for infinite space. [THM+03] provides a hashing function which maps particle position $\mathbf{p}(x, y, z)$ to a hash table of size m:

$$c = [(\lfloor\frac{x}{d}\rfloor \cdot p_1)xor(\lfloor\frac{y}{d}\rfloor \cdot p_2)xor(\lfloor\frac{z}{d}\rfloor \cdot p_3)] \tag{42}$$

where p_1, p_2, p_3 are large prime numbers chosen as 73856093, 19349663 and 83492791 [THM+03]. As shown in Figure 10, particles in space is hashed into an array of limited size.

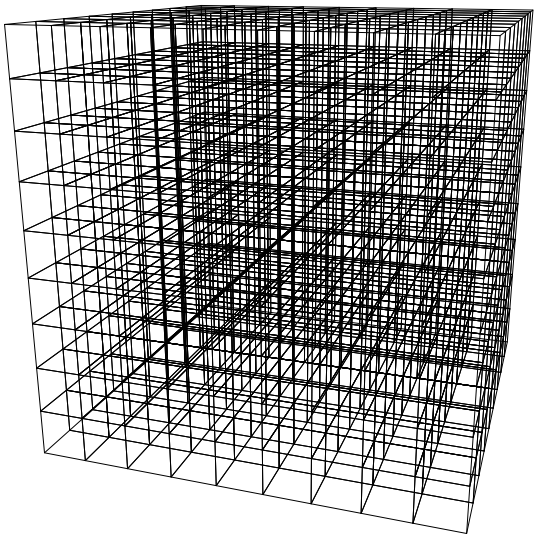

Figure 9: Space is separated by the grid. Each particle is inserted in to one of the cells. The searching area of neighborhood is narrowed down to the neighboring cells

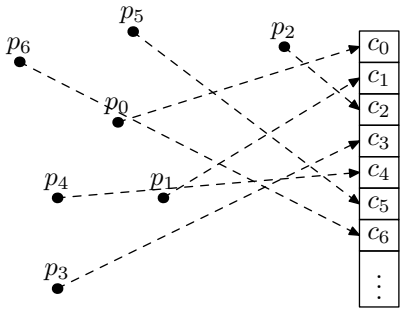

Figure 10: Points in space are hashed to a hash table.

Another method called Z-index sort is proposed by [IABT11, GSSP10]. Instead of sorting them by cell ids and put them linearly in the memory, particles are arranged in a z-curve style shown in figure 11. This z-curve style increases the memory hit. In this arrangement, particles that are close in space more likely to be close in memory, comparing to the linear arrangement.

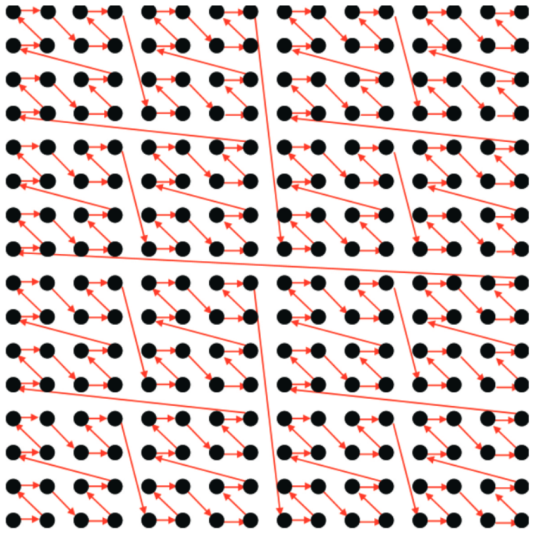

Figure 11: This image shows the memory arrangement of a 2D exmaple. Particles are put in 2 by 2 blocks.

2.2.8 GPU SIMD

GPU programming technology is soaring these years. Since graphics computation requires processing of large amount of data at the same time, GPU is designed to be good at SIMD(Single Instruction Multi Data) processing. CUDA is an API designed by nVidia to give the power of GPU general purpose computing. With GPU SIMD architecture, calculations that involve the same operations on a large amount of data can be significantly accelerated. For instance, vector addition is an operation that takes two vectors and return a vector in which each element is the sum of the corresponding elements in the two vectors. In SISD system, we need to iterate over the full length of the vector to calculate the output. However, in GPU SIMD implementation, we just need to populate data into the device memory and assign all the threads with the same addition operation. Thus, as shown in figure 12, each thread needs to take care of one element. There exist several GPGPU(General Purpose GPU) APIs. For instance, CUDA by nVidia, OpenCL and the compute shader of OpenGL Shading Language. They all have fairly good performance [FVS11]. Considering the compatibility, OpenCL is supported by all of the operation systems as well as almost any graphics card, GLSL is supported on any graphics card with OpenGL 4.3 compatibility(However, by now, Mac OS's driver supports only up to OpenGL 4.1), and CUDA can only be used on nVidia GPUs. On the other hand. GLSL has better integration with OpenGL graphics applications although all of the three has interfaces to interop with OpenGL buffers.

Nevertheless, amongst the three, CUDA provides relatively easy-to-use interfaces as well as

complete debugging tools. In GPU programming, debugging is a tough task because of the high concurrency and the errors are sometimes random [ABD+15], resulting in huge workload on debugging.

CUDA nVidia drivers provide very handy debugging tools as well as memory checking tools. The Nsight IDE can debug device code, cuda-memcheck can check GPU memory leaks and nvidia-smi provides by the driver can track GPU memory use and other parameters. All of the tools make it easier to develop device code than other platforms.

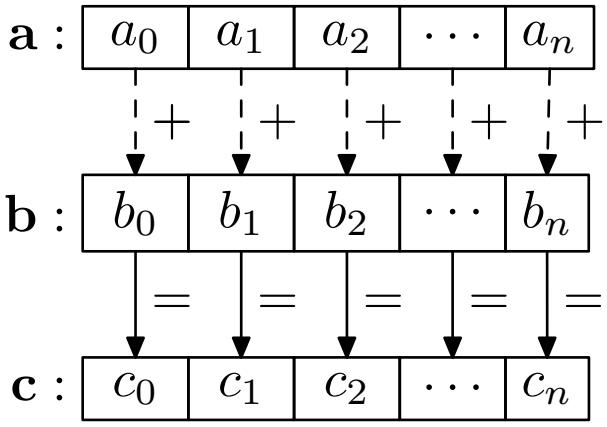

Figure 12: Vector addition is applied to an array of vectors with the same addition operation.

2.3 Rendering

One method broadly used to render particle system is metaball. Each metaball has a density function. By putting a set of metaballs together, we can represent a smooth surface as the isosurface of the density field. Polygonization or ray casting ca be further used to visualize this isosurface [LC87, LC87]. In [GSSP10], a GPU based real-time metaball ray casting rendering is offered. In the first step, the surface particles are extracted, then a distance field volume is created by rasterizing the surface point cloud into a volume with scalar values. The normals are further calculated by the volume, and lighting is calculated based on the normal informations. The surface extracting by the distance to the center of mass $\mathbf{r}_{CM_i}$ given by (43): if $\mathbf{r}_{CM_i}$ is larger than a threshold, the particle is considered on surface. This value can be extracted from the simulation result.

$$\mathbf{r}_{CM_i} = \frac{\sum_j m_j \mathbf{r}_j}{\sum_j m_j} \tag{43}$$

Another widely used method is screen based rendering. In this method, particles are firstly rendered to deptho map as squares or spheres, and then a smooth method is applied on the depth map to smooth the edges of particles. Lastly, normals are calculated based on the smoothed depth map. In [vdLGS09], bilateral filter is used to smooth the depth map of spheres. Comparing to Gaussian filter, bilateral filter preserves edges. However, since it's non-separable, therefore requires more iterations.

2.4 User Interfaces

Traditional input method like mice and keyboards have certain issues at simulation of intuitive interactions. For instance, the movements of the objects controlled by keystrokes are usually artifact. Using input devices that capture human movement in 3D has a better result in applications that depends on the natural input. Microsoft Kinect is a body movement capturing device that have been widely used not only in recreational fields but also research facilities. However, the design of the Kinect is aimed to capture whole body movement at a distance, and become limited when capturing detailed movements like fingers. On the other hand, the Leap Motion is a device that captures the movement of hands and fingers in close range and map them into 3D space. The high refresh rates guarantee the real-time mapping results in intuitive and flawless interactions to 3D objects. Also, with the development of virtual reality, VR goggles like Oculus Rift can provide users immersive experience. This kind of VR device feeds images directly into users eyes, and can track users' heads relative movement with an inertial tracker and the absolute position with an external tracker [BHP+13]. Additionally, the combination of 3D input and output device provides solutions for further enhancement virtual reality experience. In Figure 13, Leap Motion has provided a mount that can attach their device on Oculus Rift DK2. This setup allows the users to visualize their hands in VR goggles, as well as interact with 3D objects with their hands.

Figure 13: Leap Motion with Oculus Rift DK2.

Chapter 3

Proposed Approach

Implementing a real-time welding system has certain challenges. First of all, SPH simulation is an intensive algorithm, making it real time requires deliberated designed algorithms that not only time efficient but also memory efficient. Secondly, the welding process involves heat transfer between particles, and it is desirable to have a heat transfer model that compatible with SPH. Third, in the welding process, molten metals are solidified by losing heat, and can be molten again by absorbing energy from neighboring particles. This process involves multiphase interactions and needs to be well explained. Lastly, to have better user interface, make welding gun trace more plausible, a proper input method is required. In this chapter, methods are stated to solve the described challenges. Real-time is achieved by adopting GPU computation by populating workloads on a large scale of GPU threads. Then, a heat transfer model is given to calculate the temperatures transformation and emission. Multi-phase interaction is also solved in the frame of SPH. Lastly, Leap Motion is used as user input device to capture user input.

3.1 Realtime implementation with GPU

CUDA is used in this implementation. As we know, GPU is designed in the architecture that prefers SIMD operations. With SIMD operations, data are firstly populated into memory and perform the same operation at the same time. For instance, vector summation is a typical SIMD operation since each element has the same summation operation. When adding two vectors. SPH at some level can be described as this kind of problem – Each particle can be treated as a single element. Each thread performs SPH summation equation and composes Navier-Stokes equation to calculate the acceleration, and then performs leap frog interaction to update its speed and position. It is simple enough at first glance, but while diving in, it becomes more difficult. Figure 14 shows the process of a particle in one frame.

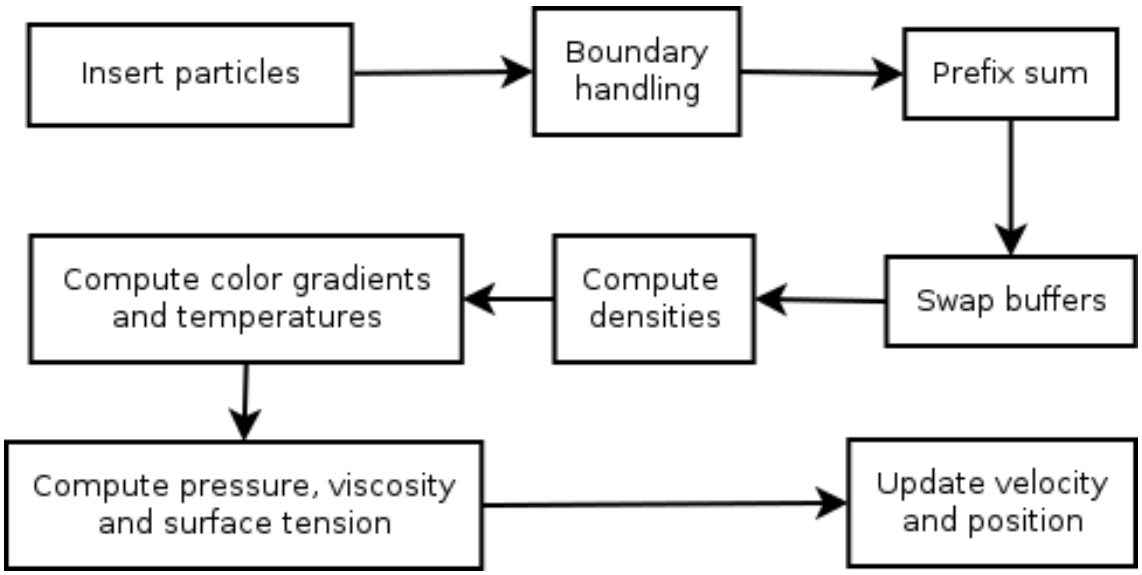

Figure 14: Flow Chart of particle handling in a frame.

3.1.1 Neighboring Search

Considering the summation equation, we don't need to search brutally for all particles, but only the particles within the support radius of kernel functions. Also, since we only care about the fluid simulation in a certain area, it is not necessary to use hashing. A grid is adopted to narrow down the searching area by separating the space into cells whose size are as same as the support radii. Each particle will be inserted into a grid. Thus, the neighboring space is narrowed down to the neighboring 27 cells. Because of the impressibility of the fluid, number of particles in each cell is limited.Figure 15 illustrates neighborhood searching in 3D. The length of the cells is equal to the searching radii guaranteeing that all of its neighbors are withing this 3×3 grid.

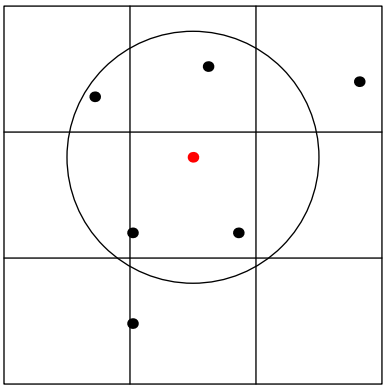

Figure 15: In this figure, the neighbor of red point is searched within the 3×3 grid, with the length of the cell equal to searching radius.

In order to keep track of particles in cells, After the insertion, each particle will have a cell ID, and each cell will have a count indicates the number of particles in the cell. Each cell will also

maintain the index of the first particle in the cell and will be used later for query particles after the sort.

3.1.2 Sorting

An important issue arises at GPU parallel implementation of uniform grad is memory coherence. After the insertion process, particles whose distance are close are not necessarily close in memory, resulting in incoherence for following processes. This issue can be solved by sorting particles by their cell IDs. Counting sort [CLRS01] is an efficient parallel sorting algorithm. We use this algorithm to sort indices because it is good at sorting integers with limit range. Counting sort involves three steps. Firstly, in the counting phase, a histogram is generated by counting the number of particles in each cell. Then, a prefix sum is performed to determine the initial position of each cell. Finally, particles are relocated in memory according to given indices. A toy example is shown in figure 16

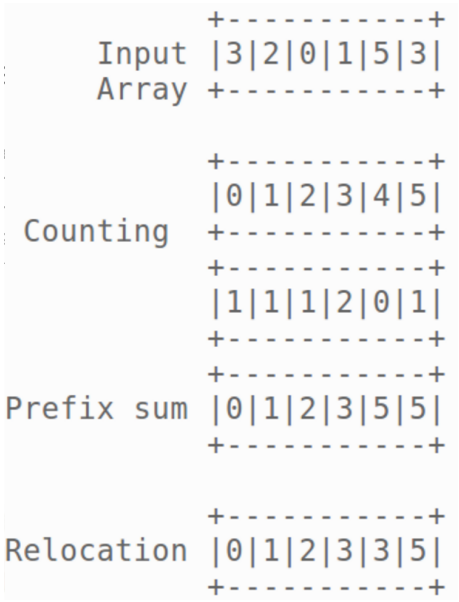

Figure 16: The histogram is firstly generated, than a prefix sum operation is performed to calculate the start indices of sorted array. Finally, input array is relocated according to the indices.

Counting

Counting phase is done during the insertion process; each cell will keep an integer indicates how many particles are there.

Prefix Sum

Prefix sum is a typical reduction process. In linear implementation, prefix sum is performed by iteration over all elements in an array, the time complexity is $O(n)$. However, in parallel implementation, this method doesn't apply. To avoid race condition, the array need to be synchronized each

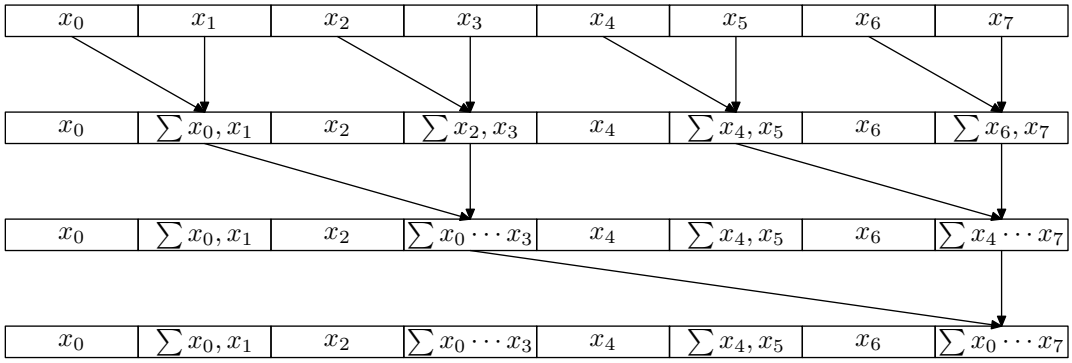

Figure 17: In reduction process, the strides are 1,2 and 4.

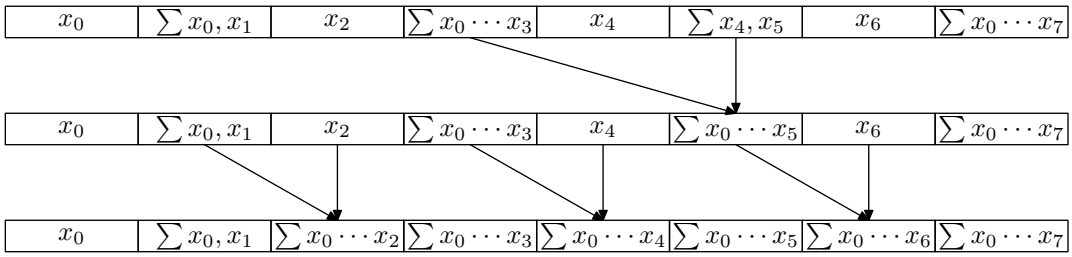

Figure 18: In post reduction phase, elements omitted in reduction phase are added together to form the final prefix sum output.

time when summation is performed, that is to say, each element in the array needs to perform a full summation process, without using the summation result of previous element. It is not work-efficient comparing to linear solution. [Ble90] describes a work-efficient multi-thread prefix sum algorithm. This process consists of two phases. In the first phase, the array is forwardly reduced as shown in figure 17 with stride from 1 to number of threads in a block, multiplying by 2 in each iteration. Then in the second phase described in figure 18, the array is reversely added back to get the final result.

This process is further accelerated by adopting shared memory. Shared memory is a part of memory in the graphics card hierarchy which is designed to share between the threads in one thread block. It has relatively small capacity comparing to global memory but has lower latency. If no bank conflicts between thread, the latency of shared memory is 100 times smaller than uncached global memory. However, since it is allocated for each thread block, the size of shared memory is limited. Moreover, because of the limitation of the number of threads per block, e.g. maximum 1024 threads per block in GPUs higher than 2.0 compute capability, All of the particles can not share a single shared memory. In Mark Harris' [HSO07] CUDA implementation, he discussed the work efficient implementation in one block. In his implementation, each thread will take care of the summation of two elements, thus, with the maximum of 1024 threads in a thread block, it can handle an array of 2048 elements.

In our implementation, we have a grid with 16×16 $times16 = 4096$ cells. In this case, the array is separated into two arrays with 2048 elements each. Then the prefix sum is applied to the two

arrays simultaneously. In order to merge the outputs, an extra array is created to store the last element of each output. By applying prefix sum on the extra array, we can get the offset of each segment. The final output is got by adding all of the elements with their corresponding offset in the extra array. Figures 19 shows the prefix sum scheme. This scheme can be applied to the array has less than 2048^2 elements. More elements can be achieved by recursively calling this method.

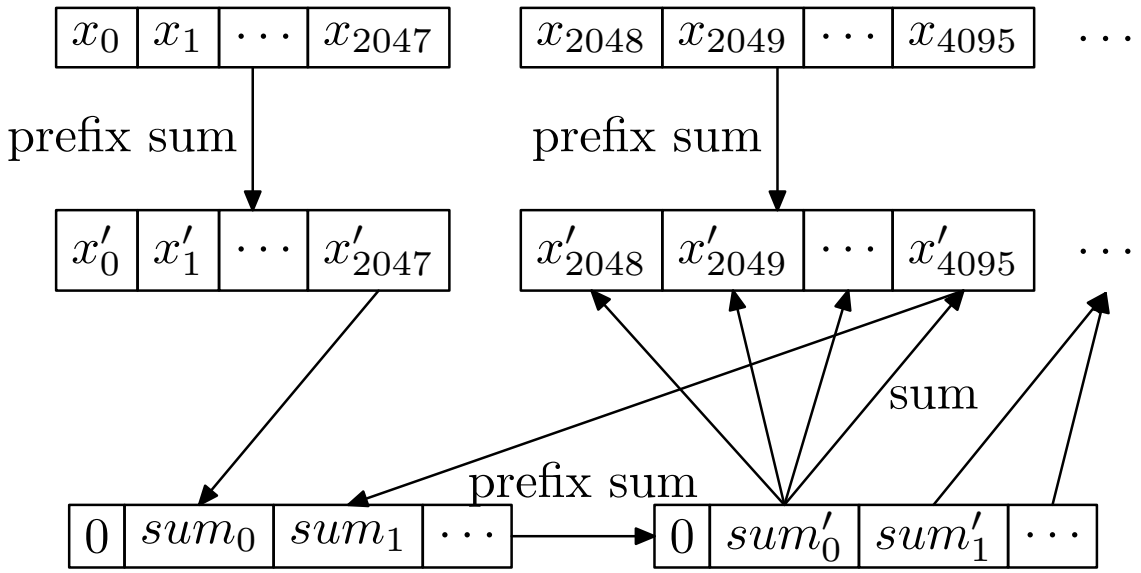

Figure 19: The first prefix sum is performed in each thread bock. Then the last elements of outputs are copied to an extra array $sum_0 \cdots sum_n$. After that, a second prefix sum is performed on the extra array. Finally, the final output is achieved by summing elements in extra array back to outputs of all thread bocks.

Relocation

The last step is to relocate the particles in a correct order. In traditional method, data need to copy back after being sorted. However, in our implementation, a ping-pong buffer is used to eliminate the copy process. In the memoroy allocating process, two sets of memory are allocated, let's call them ping buffer and pong buffer. In the odd frames, particles are inserted into ping buffer, and than sorted and copied to pong buffer, then preform the further operation on pong buffer, vice versa.

3.1.3 Summation Equation

Interpolation is the crucial step that influnce the accuracy of the simulation. Here, the SPH summation is used to calculate the contribution of neighboring particles to the centering particle:

$$A(\mathbf{r}) = \sum_j m_j \frac{A_j}{\rho_j} W(|\mathbf{r} - \mathbf{r_j}|, h) \tag{44}$$

where W is the kernel function. The first and second derivatives are simply derived:

$$\nabla A(\mathbf{r}) = \sum_j m_j \frac{A_j}{\rho_j} \nabla W(|\mathbf{r} - \mathbf{r_j}|, h) \tag{45}$$

$$\nabla^2 A(\mathbf{r}) = \sum_j m_j \frac{A_j}{\rho_j} \nabla^2 W(|\mathbf{r} - \mathbf{r_j}|, h) \tag{46}$$

This interpolation method will achieve good result if kernel function is carefully chosen.

3.1.4 Kernel Functions

There is a wide choice of kernel functions as long as it satisfy certain criteria. For example, compact support, normalized and differentiable. According to [MCG03] The kernel functions been used are:

$$W_{poly} = (\mathbf{x_{ij}}, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - x_{ij}^2)^3 & 0 \leq x_{ij} \leq h \\ 0 & otherwise \end{cases} \tag{47}$$

$$W_{spiky} = (\mathbf{x_{ij}}, h) = \frac{15}{\pi h^6} \begin{cases} (h - x_{ij})^3 & 0 \leq x_{ij} \leq h \\ 0 & otherwise \end{cases} \tag{48}$$

$$W_{viscous} = (\mathbf{x_{ij}}, h) = \frac{15}{2\pi h^3} \begin{cases} -\frac{x_{ij}^3}{2h^3} + \frac{x_{ij}^2}{h^2} + \frac{h}{2x_{ij}} - 1 & 0 \leq x_{ij} \leq h \\ 0 & otherwise \end{cases} \tag{49}$$

The shapes of kernel functions are depicted in figure 20. W_{poly6} is used for all cases except viscosity and pressure. W_{spiky} is used for pressure since in W_{poly6}, the second derivative vanies when two particles are getting closer, which vilolates the fact that the closer the particles are, the more repulsion force they have. However, the gradient of W_{spiky} satisfy the phenomenon by reaching the maximum when two particles overlap. Viscosity adopts the Laplacian of kernel functions. In W_{poly6}, the second gradient becomes negative, which means that the effect of viscosity becomes smaller when two particles getting closer, this also doesn't satisfy the truth. While with $W_{viscosity}$, the second derivative is positive and increases when two particles are becoming closer.

3.1.5 Densities

Densities of particles are calculated by substituting A with ρ in summation equation (15).

$$\rho_i = \sum_j m_j W(\mathbf{x_{ij}}, h)$$

The kernel function used here is poly6 (47).

3.1.6 Pressure Forces

Pressure force is the surface force applied on control volume. This force is caused by the difference of density in the fluid. Firstly, let's take a look at the common form of gas state law:

$$pV = nRT$$

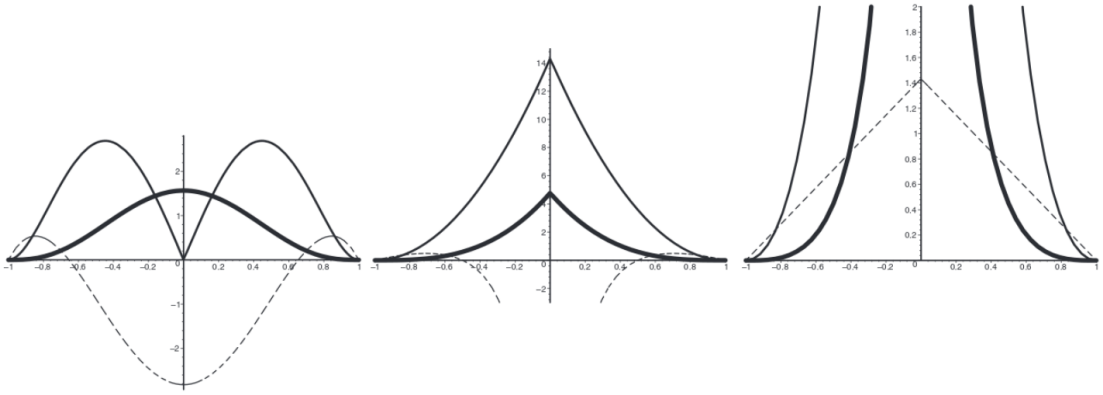

Figure 20: From left to right are W_{poly6}, W_{spiky} and $W_{viscosity}$. and their first and second derivatives.

where p is the pressure, V is the volume, n is the amount of substance, R is the ideal gas constant equal to the product Boltzmann constant and Avogadro Constant and T is the temperature. Since $n = \frac{m}{M}$, this equation can be rewritten as:

$$p = \rho \frac{R}{M} T.$$

Note that R and M are constants for the same kind of fluid, we use a variable k as a function of temperature $k(T) = \frac{R}{M}T$. Thus, the gas state law can be written as:

$$p = k\rho$$

To calculate the pressure force of i^{th} particle p_i, the following equation is applied [DG96,MCG03]:

$$p_i = k(\rho_i - \rho_0),$$

where ρ_i is the density at location i yielded by (16), ρ_0 is the rest density of fluid and k is the gas constant which is a function of temperature. The equation added $-\rho_0$ as an offset to pressure. Since we are calculate the gradient of pressure ∇p, this offset will mathematically have no effect on the result. On the other hand, it will also stabilize the simulation by limiting value in a relatively smaller range.

Applying SPH summation equation, the pressure of i^th particle is:

$$f_i^{pressure} = -\sum_j \frac{m_j}{\rho_j} p_j \nabla \mathbf{W_{ij}}$$

Unfortunately, the pressure given by the equation above is not symmetric. For example, given a pair neighboring particles a, b, assuming each particle represents the same volume V, the pressure force applied on each others are:

$$F_a^{pressure} = -\frac{m_b}{\rho_b} p_b \nabla \mathbf{W_{ab}} \cdot V, \tag{50}$$

$$F_b^{pressure} = -\frac{m_a}{\rho_a} p_a \nabla \mathbf{W_{ba}} \cdot V \tag{51}$$

in which $f_a^{pressure} = f_b^{pressure}$ is not always satisfied since ρ_a and ρ_b, p_a and p_b are not necessarily equal, results in violation of Newton's third law. Here, we symmetrization them by taking the arithmetic mean of two neighboring particles, yields:

$$f_i^{pressure} = -\sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla \mathbf{W_{ij}}$$

The kernel function we used here is kernel spiky (48).

3.1.7 Viscosity

Viscosity term in Navier-Stokes equation is $\mu \nabla^2 \mathbf{u}$. Velocity of previous iteration is known, so we can calculate the Laplacain of velocity and plug it in to the SPH summation equation, yields:

$$f_i^{viscosity} = \mu \sum_j m_j \frac{\mathbf{u_j}}{\rho_j} \nabla^2 \mathbf{W_{ij}}$$

However, like the pressure force, viscosity also suffers from unsymmetrical forces. Applying the same trick, we have:

$$f_i^{viscosity} = \mu \sum_j m_j \frac{\mathbf{u_j} - \mathbf{u_i}}{\rho_j} \nabla^2 \mathbf{W_{ij}}$$

3.1.8 Surface Tension

Surface tension is essential to simulate the particles near the free surfaces. In the traditional methods that are implemented by minimizing the surface curvature, an important issue is that while applying external forces on surfaces, conservation of momentum is violated. As we know, surface tension is caused by cohesion forces among particles. The particles on free surfaces is affected by unbalanced cohesion forces from their neighbors, results in the phenomenon of surface tension. It is desirable to add cohesion forces on surfaces instead of introduce new forces to the system. In this implementation, we use the method proposed in [AAT13], combined with method suggests in [IIFS12]. Firstly, the cohesion force is calculated by:

$$\mathbf{F}_{i \leftarrow j}^{cohesion} == \gamma m_i m_j C(|\mathbf{x_i} - \mathbf{x_j}|) \frac{\mathbf{x_i} - \mathbf{x_j}}{|\mathbf{x_i} - \mathbf{x_j}|}$$

3.1.9 Heat Transfer

Within the particles, thermos energy will transport from hot particles to cold particles. The temperature of cold particles will heat up by surrounding cold particles. The heat conduction can be easily integrated into SPH system by smoothing out with a kernel function. In [IIFS12], a heat transfer formula is given as:

$$\frac{\partial T_a}{\partial t} = \frac{1}{\rho_a C_a} \left(\frac{6}{\lambda_a} \sum_b k_b (T_b - T_a) W_{ab} + Q_a \right)$$

$$\lambda_a = \sum_b (r_b - r_a)^2 W_{ab}$$

Where C_a is the specific heat, T_a and T_b are the temperature of current particle and neighbor particles. W_{ab}, again is the kernel function. This formula can be easily integrated into SPH system.

3.1.10 Multi-Phases

Fluid simulations involve solid and fluid phases. Temperature plays an important role in the phase changing. When the temperature is higher than the melting point, particles will be fluid. However, when the temperature is lower than the melting point, it will become solid particles.

Fluid particles satisfied the process of the SPH. The stress of fluid is proportional to the temperature, given by [GA05]:

$$\sigma_y = \sigma_{yr} + C_y(T - T_r) \tag{52}$$

where σ_y is the stress of the material at temperature T, T_r is the reference temperature, σ_{yr} is the stress at T_r and C_y is a material coefficient. Thus, the viscosity coefficient is a linear function of temperature.

Solid particles have a different behaviour. Particles merge into a rigid body when solidified. The transformation of rigid bodies is determined by its angular velocity, acceleration and linear velocity and acceleration. To calculate them, we need first to get the torque of the rigid body.

$$\boldsymbol{\tau}_i = (\mathbf{r}_i - \mathbf{r}^{cm}) \times \mathbf{F}_i \tag{53}$$

Where $\boldsymbol{\tau}_i$ is the torque of a particle in the body. $\mathbf{r}_i$ is the position of a particle, $\mathbf{r}^{cm}$ denotes the center of mass and $\mathbf{F}_i$ is the total force applied on the particle. By summing them together by $\boldsymbol{\tau}_{body} = \sum_{i \in body} \boldsymbol{\tau}_i$, we can get the torque of the rigid body. The external force is calculated in the same fashion by $\mathbf{F}_{body} = \sum_{i \in body} F_i$. Note that the forces and torques within the same rigid body is omitted.

Thus the linear acceleration of rigid body is given as $a = F/m$ and the angular momentum is updated in every frame by $\mathbf{L} = \mathbf{L} + \Delta t$

In the first attempt, we modeled solid particles as the rigid body. In this method, we cluster solidified particles, then calculate their mass center, mass, and moment of inertia. According to the external forces applied on each particle, we can derive the linear acceleration and angular acceleration. This process has pretty good simulation with both solid and fluid particles.

However, it requires a lot more computation power since clustering is not easy to parallelize.

Considering molten metals are very viscous fluid, when solidified, instead of split away to form different rigid bodies, it tend to stick together as a whole rigid body. Furthermore, since workpiece is fixed on a table, it is safe to freeze simply particles when they are solidified.

3.2 Boundary Handling

Collision with rigid bodies is integrated with SPH system since all rigid bodies are modeled by fixed SPH particles. However, Collision with grid boundaries are solved by two methods. In the first methods, we reflect the particle's speed by the normal of the boundary. An energy penal coefficient is added if the collision is not perfectly a elastic collision. In another methods, in order to simulate ultimate flow, particles move outside of the grid is removed. This is achieved at the insertion part.

If the position of particle is inserted to a space outside of the gird, this particle will be assigned with a cell ID that's more than the number of the cells in grid, and atomically subtract the number of particles by one. During the sorting part, the particles outside of the grid will be sorted at the end of the array. During the following operations like calculating forces and densities, the GPU only calculate the number of particles that's given, thus the particles outside of the grid will be omitted. Also, a particle can be removed in the same fashion. This methods does not requires relocation of the memory, and keeps the speed fast.

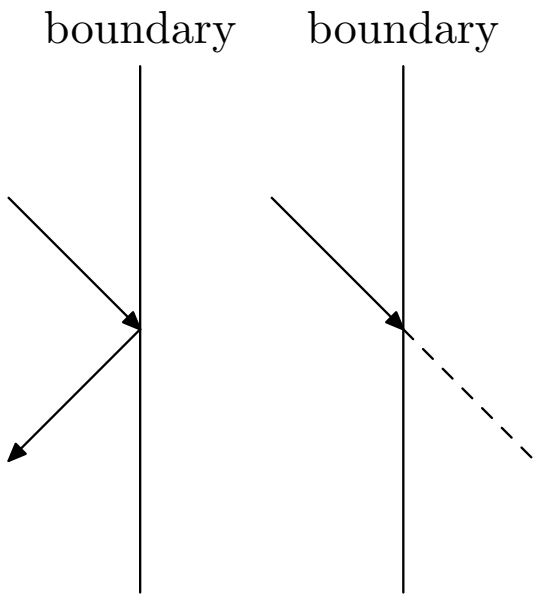

Figure 21: On the left is the reflective boundary, particles bounce back when reached outside box. On the right is the free boundary, particles are disappeared when going beyond the boundary.

3.3 Rendering

The scene is rendered with OpenGL 4.3. Particles and environment, including welding gun and skybox, are rendered separately: one uses deferred rendering because the lack of connectivity information and another uses forward rendering. The texture of skybox is reused to map onto the particle system to calculate the reflection.

3.3.1 Rendering of Particles

There are a lot of rendering techniques exists in SPH. One popular method is using marching cube. This process is in a way similar to the uniform grid in neighbor search. However, it is generally slow and have bumps on the surfaces. Another method that is widely used is screen space rendering. Screen space rendering is a derivative of deferred rendering. It passes all necessary informations like depth, normals and textures to the final stage, than calculate them. In our implementation, firstly particles are rendered to spheres by substituting depth. Then, we smoothed the depth image using

bilateral filter given by:

$$I^{filtered}(x) = \frac{1}{W_p} \sum_{x_i \in \Omega} I(x_i) f_r(||I(x_i) - I(x)||) g_s(||x_i - x||). \tag{54}$$

where $I^{filtered}$ is the filtered image,I is the original image, x is the coordinate of current pixel, Ω is the window size centered at x. f_r is the gaussian function, g_s is the spatial kernel and W_p is the normalization term:

$$W_p = \sum_{x_i \in \Omega} f_r(||I(x_i) - I(x)||) g_s(||x_i - x||). \tag{55}$$

Comparing to the Gaussain smooth function, the bilateral filter can preserve the edges, as shown in figure 22. This feature is quite useful to smooth the particles on the free surfaces.

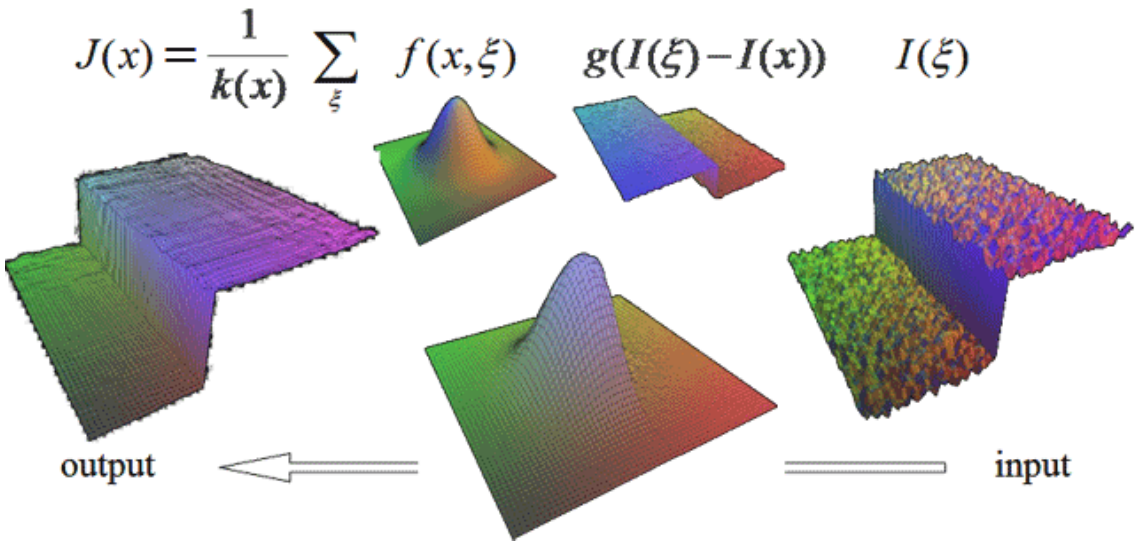

Figure 22: Bilateral function smoothed noises on the surfaces while preserving the edges.

Finally, we use this smoothed depth map to generate normals, and applying lights and reflections.

Rendering of Environment

To generate a relatively realistic background, we used a skybox. Skybox is a widely used techniques in games and simulation programs. A Skybox is a cube in which scene is rendered. There is a texture attached on the inside of the box to simulate the background environment. This skybox is rendered in camera space, and moves with camera. Further more, the texture of skybox can be mapped to the object in the 3D space to simulate the reflection effect.

The welding guns is loaded as a model. This model can be chosen by users.

3.4 User Interface

Leap Motion is adopted to have a good simulation of the welding gun. Leap Motion is a stereo camera with ability to track fingers' positions. In the Leap Motion SDK, the origin of coordinate

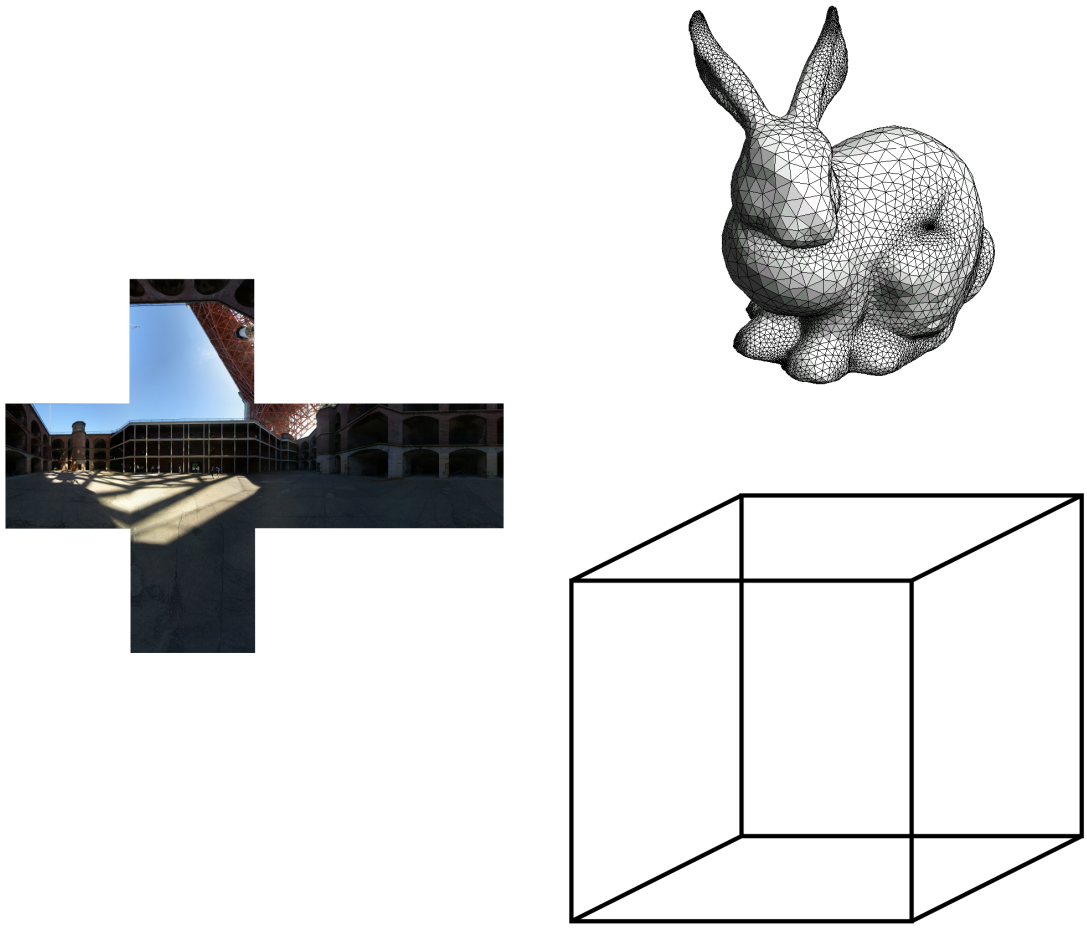

Figure 23: The texture on the left is attached to the cube to render the skybox, and attached to the bunny to generate reflection effect.

is the Leap Motion, the Y axis is perpendicular to the cameras, and X axis is along the long side of the device. The following images shows the coordinate. In every frame, the relative position of fingers are returned. An important issue here is that the Leap Motion has better performance if the fingers' directions are nearly perpendicular to the Y axis. However, since in welding simulation, we want to use the fingers to simulate the welding guns, which means we will mostly point fingers down. Considering the situation in figure 24, we rotated the Leap Motion along X axis for 90 degree and fixed it on an box. As shown if figure 25

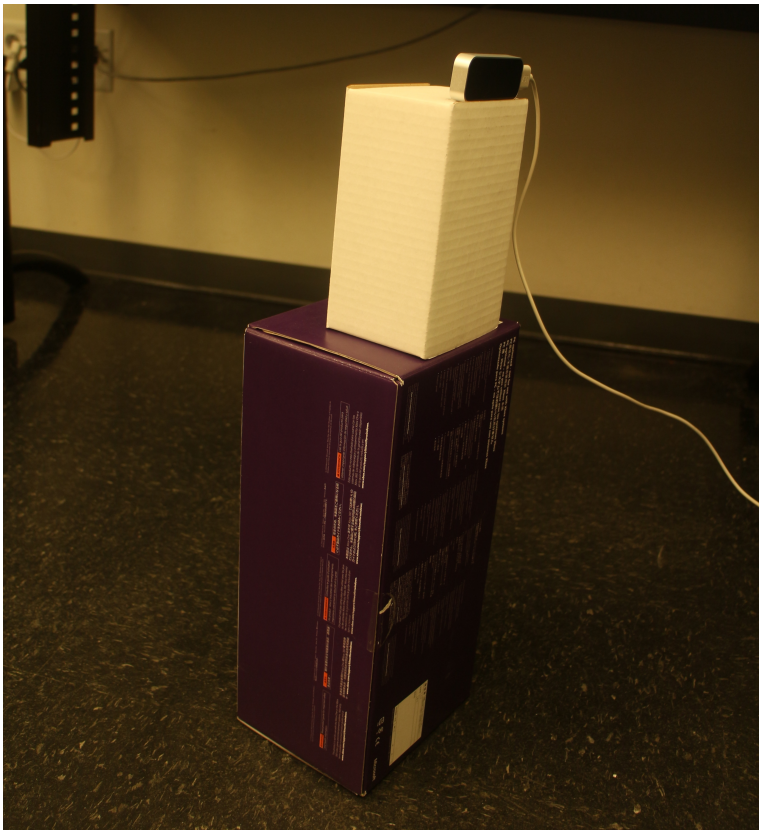

Figure 24: Leap Motion Setup

Generally, Leap Motion is considered a stationary equipment and camera for rendering is also fixed. In this case, the positions of hands are mapped to object space or word space. However, in our application, the camera can be rotated by mouse. Once the camera is moved from the original position, tracking hands positions to object space is not intuitive since people generally detect relative positions of the objects referring their eyes. This issue is addressed by mapping hands positions to view space. Figure 25 shows the rotated coordinate system. When the camera is rotated by mouse without moving hands, the relative position of hands is fixed on the screen. Another advantage of adopting this scheme is that it is easier to be integrated with VR devices. Leap Motion has a VR developer mount that can attach the Leap Motion to Oculus Rift DK2. While the position of hands in the 3D space is relative to the eye position, user feels more consistency.

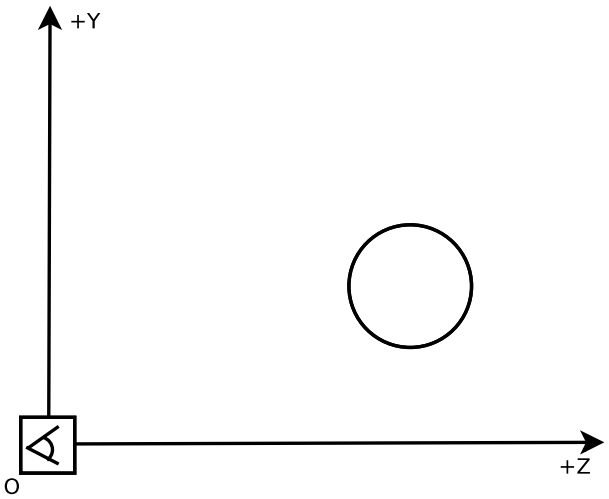

Figure 25: Leap Motion and camera share the same coordinate system. The object captured by Leap Motion is the at the same coordinate that seen by camera.

Chapter 4

Results and Discussions

This section describes the implementation details and some simulation result.

4.1 Implementation

The system is implemented with maintainability and extensibility in mind, as well as the principle of robustibility. Firstly, an object-oriented OpenGL 4.3 framework is firstly built with compatibility of OpenGL compute shader for rendering. It also contains basic forward and deferred renderers and an extensible Python APIs. Within this framework, fluid simulation system is implemented as an plug-in of this system. A Python console is used to control the fluid system. Further more, the welding gun can be controlled by both keyboard and Leap Motion, giving users more choices.

As shown in figure 26, Scene class is the Single Tone that manages all of the resources, including shaders, cameras, lights renderers and objects. Shaders are shared by renderers in order to assemble necessary rendering pipeline. This class is also in charge of shader creation, including vertex shader, geometry shader, fragment shader ad compute shader (if OpenGL 4.3 or above is used), as well as getting uniform locations. Renderers are rendering pipelines, e.g. forward rendering, deferred rendering or raytracing; each renderer has an array of objects attached to it. When to perform rendering, the renderer will render all of the objects assigned to it. With this design, one object can be rendered multiple times if necessary. The object is the superclass of all objects, e.g. meshes and SPH system. It contains buffer objects in GPU and a tree structure. In another word, all of the objects are arranged in the Scene in the tree hierarchy such that translations can be stacked. In this implementation, a basic MeshNode is inherited from the superclass of Object as a basic static mesh, and the welding gun LeapEmitter is inherited from MeshNode class in order to map LeapMoition input to mesh transformations. One more different of the LeapEmitter with other meshes is that the transformations of LeapMoition are mapped to screen space instead of object space. This design ensures that the transformation of welding gun is always consistent. SPH system is also inherited from the Object class. In the initialization part, memories are allocated in GPU as vertex buffers using OpenGL APIs, each vertex represents a particle. Then a CUDA API is used to get the pointers in GPU. With theses pointers, CUDA kernel functions are able to know where to

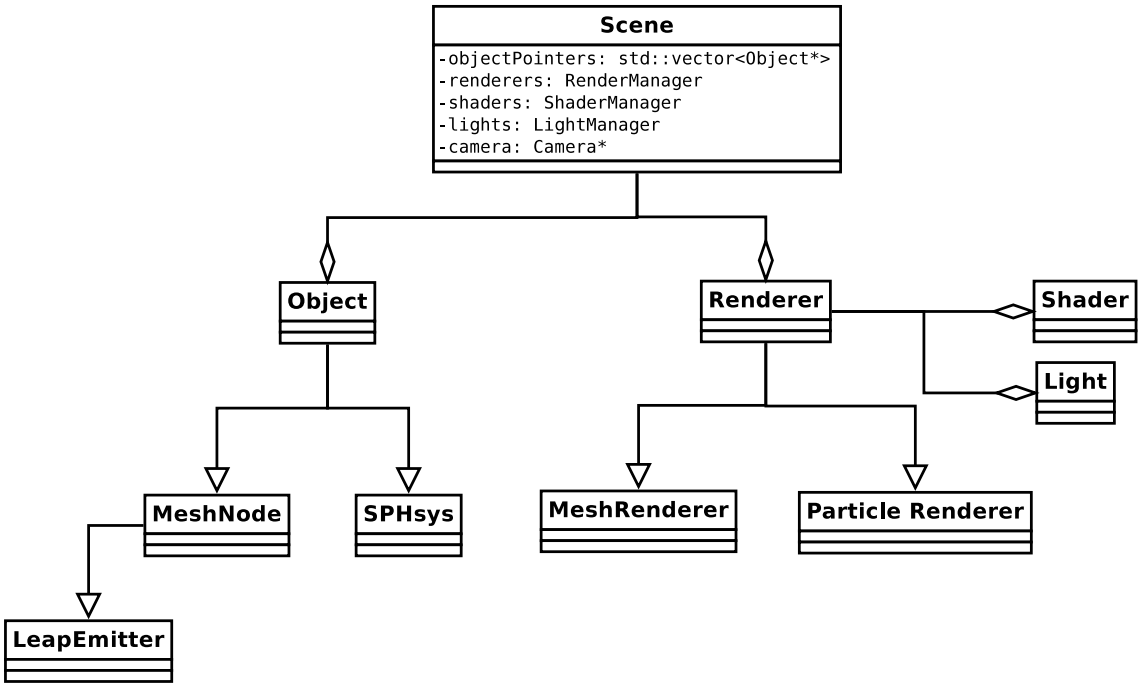

Figure 26: Simplified Class Diagram

find the necessary resources in GPU. In the SPH phase, SPH algorithms are applied to the vertex buffers. Then in the rendering phase, a renderer is assigned to render those vertex buffers. This architecture eliminates the overhead of transferring data between host memory and device (GPU) memory. Note that resources are allocated and deleted by Scene class. All the other classes need to do is free the memory they created in their own destructors. This design ensured the extensibility of the system: users can write their own renderers, objects by inheriting from the base classes, manage their own resources within the class and register to the system, scene class will automatically render them and free the memory when the object is destroyed.

Another feature that designed to enhance extensibility is the exposure of API to Python. Python is a script programming language with an interactive shell as well as plenty of GUI libraries. Instead of tweaking parameters within the C++ code, changing values through Python API is more elegant and convenient. For example, the famous free software Blender uses Python as its GUI and script language. In PyConsole class, Python APIs are implemented. These APIs runs on a separate thread without interfering or blocking the system. Whenever these APIs are called, the functions are encapsulated as lambda functions and pushed into a function queue. On the main thread, the queue is checked in each iteration if there's command waiting.

The discussion will focus on two aspects. First part is the accuracy of simulation. In this part, multiple examples is stated to test the result of the simulation. This will include the extreme situations like melting through. The other part is to examine the performance improvement comparing to traditional implementations. In this section, a table of CPU implementation, naive GPU implementation and advanced GPU implementation is given. Real situations can be shown to the users

for more details.

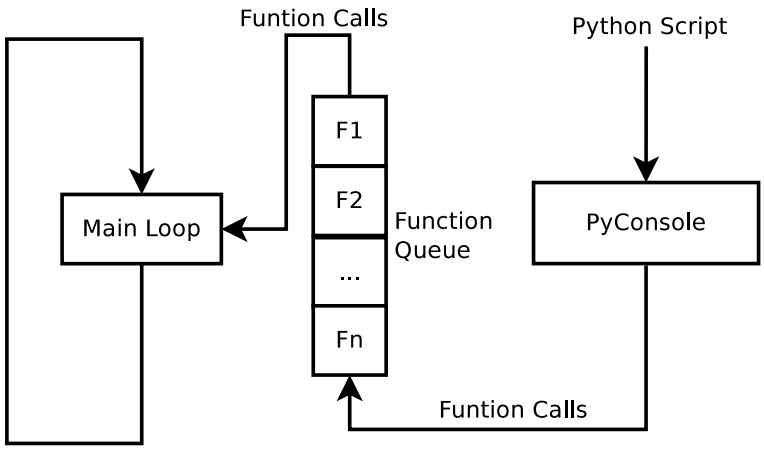

Figure 27: Flow diagram of python console.

4.2 Simulation Results

In this section, a few running examples are listed. First, The plates is used to test certain situations. The, V-Joint and T-Joint are commonly used in welding practices. A bunny is used to indicate that this system can accommodate arbitrary shapes. At last, an extreme situation of melting through is shown. Note that all of the resluts are generated interactively with high frame rate.

4.2.1 Bars

The basic implementation are two bars that put together side by side. The operation is to fill the gap between the bars.

4.2.2 V-Joint

V-Joint is formed by two trapezoid metal put side by side. Welders are required to inject filler materials between the gap to merge them together. In this example, we modeled a V-Joint mesh with grids in Blender, and imported into the particle system with our importer. The result is shown in figure 29. Note that particles on the left is rendered as metal, and particles on the right are rendered as spheres with the temperature color coded.

4.2.3 T-Joint

T-Joint is also a common workpiece for welding and widely used in real-life circumstance. It is formed by putting two plates perpendicularly. The filler material is injected into the right angle between the two plates on both sides of the vertical plate. The result of T-joint is shown in 30. The filler material is injected by using Leap Motion.

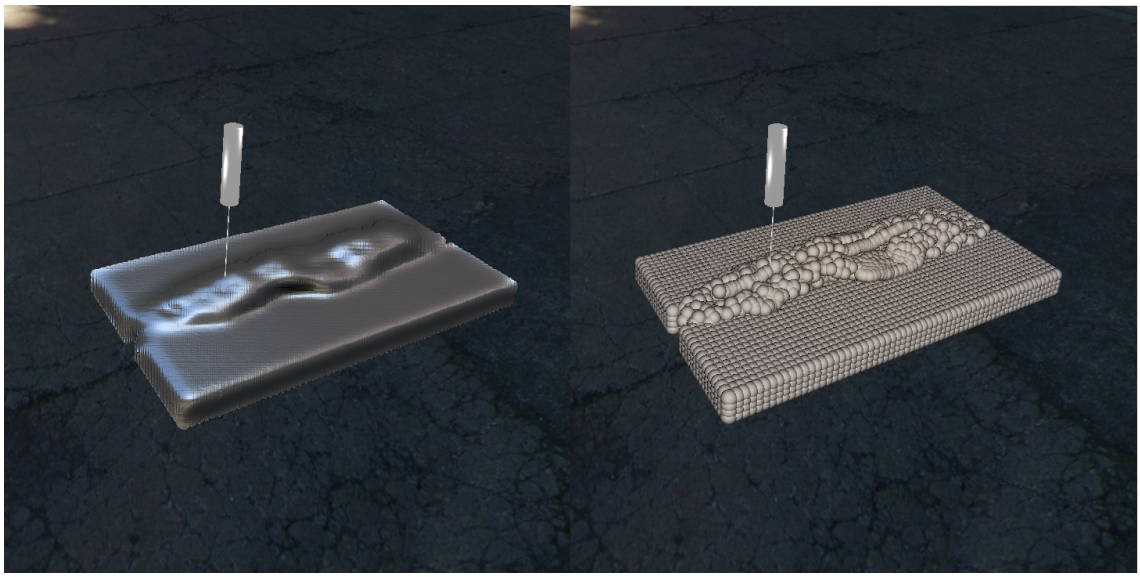

Figure 28: Bars, the color in the picture on the right is color coded.

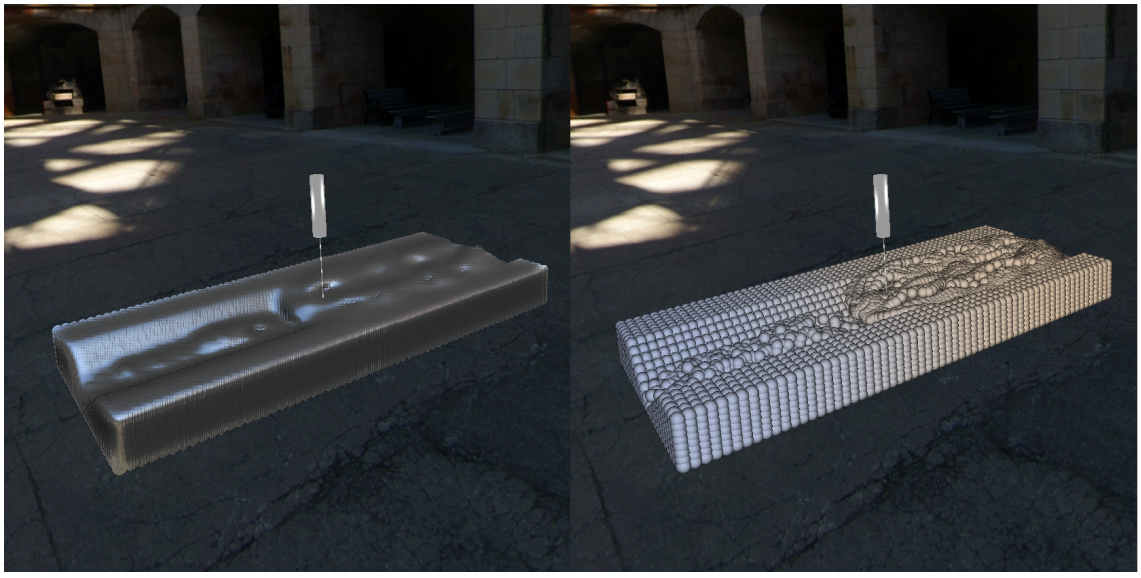

Figure 29: V joint

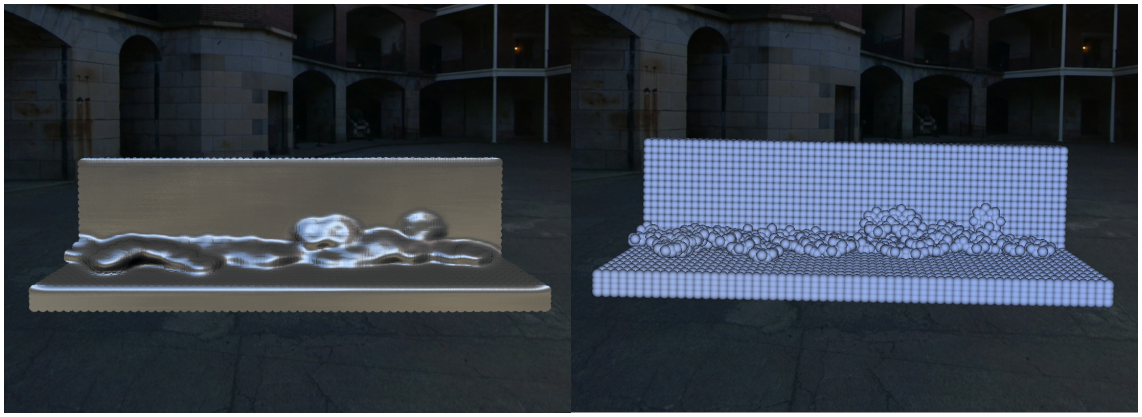

Figure 30: T joint

4.2.4 Bunny

With the simulation, we also investigate the possibility of performing welding on more complicated meshes. Figure 31 shows a result of welding simulation on a bunny. The back of bunny was missing, and the user need to fill the gap with the welding gun. The Leap Motion tracker is easy to use such that this process can be done in easily by anyone without experience this system. Also, this example proves that the simulation can be applied to any surfaces.

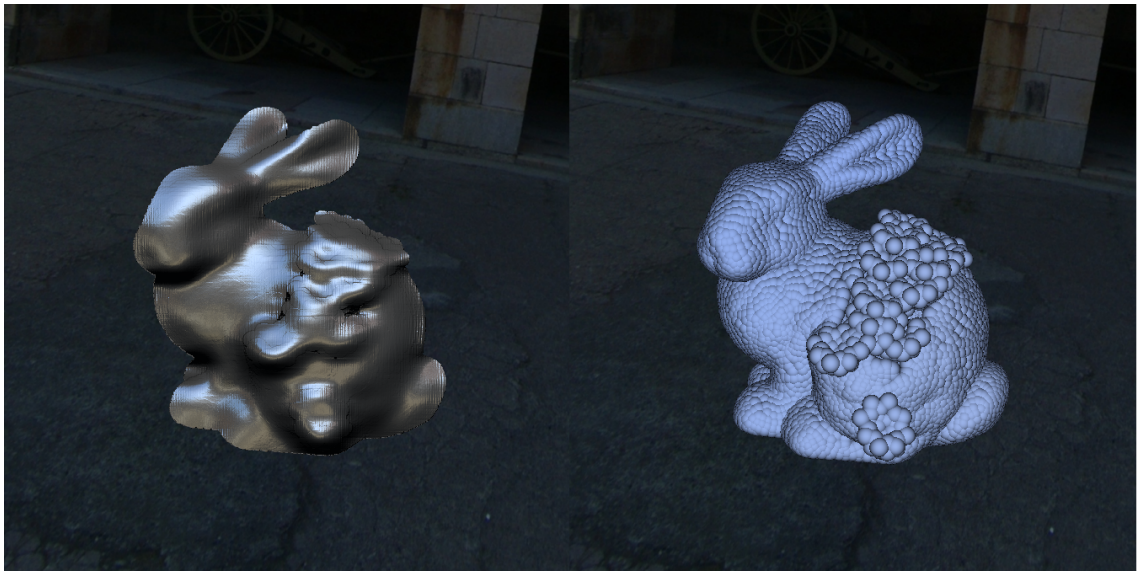

Figure 31: Bunny

4.2.5 Melting-Troughs

Melting-Through happens when hot particles are emitted to one place for too long. The hot particles keep melting the base material. When the particles in the thickness direction are all molten, the

particles will go through and form a whole on the work piece. The example in figure 32 shows this situation.

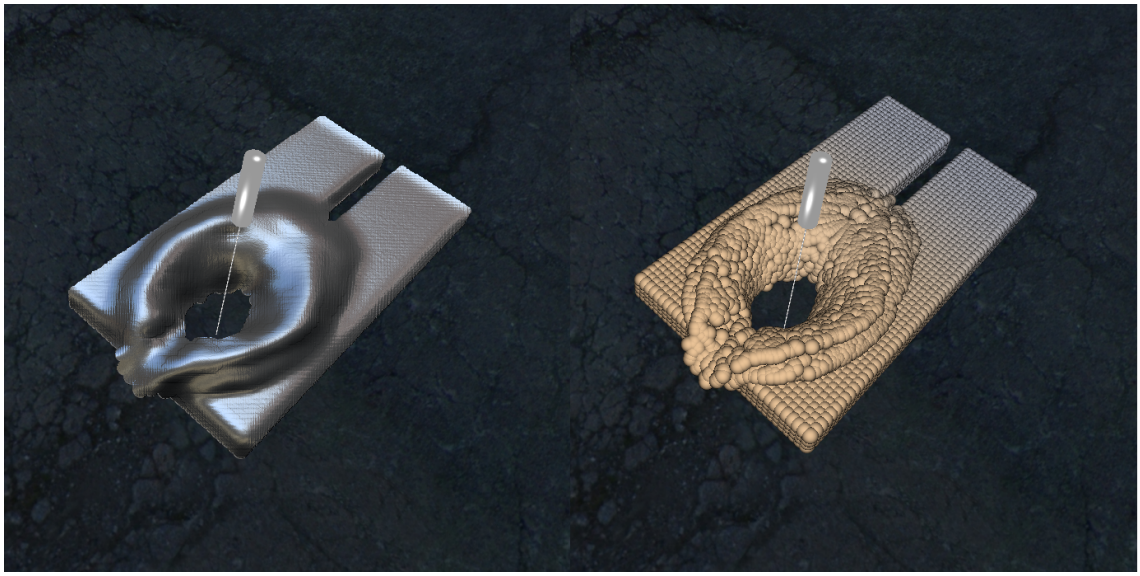

Figure 32: Melting through

4.3 Benchmarks

The speed of the simulation is essential for real-time implementation. Our implementation achieved real-time welding simulation that allows the user to perform interactive welding training precess. Also, the granularity is high in order to have good simulation result. Table 1 shows the FPS of different examples mentioned above.

4.4 Evaluation

In this section, we compared our simulation result with existing simulation methods as well as real welding images.

The first evaluation focuses on the comparison of our simulation and real life cases. In this case, we choose a typical workpiece of two plates. The filler material is injected into the gap to form the welding pool. By melting the base, filler material fuses with the workpiece. Figure 33 shows this case. The simulation result yielded similarity to the result on the right both on the shape and the rendering. Note that the simulation area correspondent to a small part of welding pool on the right image.

The second evaluation focuses on the real welding output as well as a mesh-based simulation system in the industry. Arc+ is a welding simulator that currently developed by a company in Montreal, QC. In their system, they have adopted a mesh-based method that generates the result of welding based on the movement of welding gun and other parameters. Their method is not based on

physical simulation, and the equation is highly tuned to certain workpieces. However, our method is physical based and can handle any surfaces. In this case, we choose a common mesh, T-joint, to compare the welding results. Figure 34 shows the three outputs on the T-joint. In the picture, we observe that our simulation result preserves volume and is more close to the actual case depicted in the bottom image.

Also, comparing to the latest existing methods [IIFS12,Ind09] in welding simulation. Our method incorporates phase changing, heat transfer, injection of new particles in a real-time simulation system.

Figure 33: The left image is the simulation result of our system. The right image is a photo of welding.

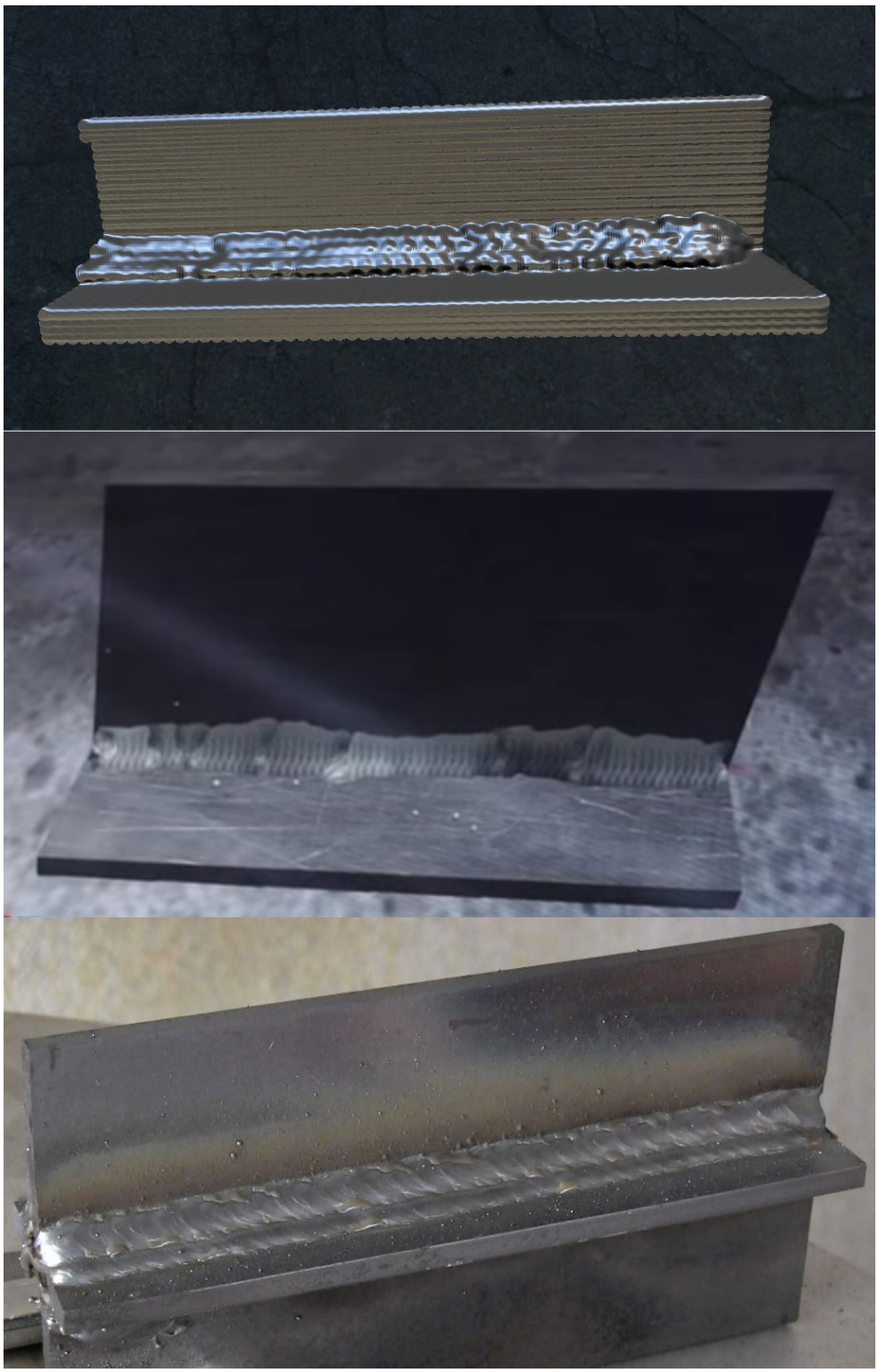

Figure 34: The top image is the simulation result. The middle image is the simulation result of Arc+. The bottom image is a photo of real welding.

Table 1: Benchmarks

Type	Number of Particles	FPS with Rendering	FPS without Rendering
Bars	10076	44.7	82.7
V-Joint	17760	35.8	58.4
T-joint	11714	36.7	110
Bunny	6990	35.8	58.4
Melting-Through	10597	43.2	76.5

Chapter 5

Conclusion

In this work, we implemented a real-time welding training system based on smoothed hydrodynamics particles. The essential part of this system is a real-time multi-phase SPH fluid simulation methods. This method employed CUDA as GPU computation system and parallel counting sort to reduce memory inconsistency. Also, Leap Motion is used to track users input. The result of simulation yields close situations to the real welding process, the finger tracker provides acceptable accuracy. Moreover, the simulation is implemented efficiently. The frame rate reaches real-time that makes user operate interactively. Besides, an object-oriented fluid simulation framework is created that allows users to develop their fluid or rigid objects by simply inherit from base classes and implement the interfaces. The whole project is built from scratch in order to satisfy the requirement of flexibility and enabled the possibility of advanced optimization.

In this section, the challenges and solutions will be summarized, after that, limitations will be stated objectively. Finally, a section of future work will explain the direction of the further development, as well as the potential for further improvement.

5.1 Limitations

Though the simulation system achieved real-time implementations, it still has certain limitations. Here, the limitations and their causes will be stated, as well as possible solutions.

- In order to accelerate simulation, rigid body is simplified to support only one reference body. The artifact will happen when there are more than one rigid bodies in the system. The rigid body that away from the main rigid body will freeze in the air instead falling down. This issue can be addressed by using the previous implementation. However, it will increase the cohesion within particles and makes it difficult to incorporate with GPU parallel scheme, resulting in drastic performance impact.

- Another issue is that, since we are using WCSPH, the negative force is eliminated in order to avoid disorder, resulting non-divergence free velocity field, as shown in 35. However, in incompressible fluid, the negative force doesn't exist and the density should remain constant.

The reason why it happens here is because the velocity field that calculated by SPH is not divergence free. This issue can be addressed by adopting iterative correction pressure solver like PCISPH, LPSPH or pressure projection solver like IISPH. Divergence of velocity field is calculated by:

$$\nabla \cdot v = \sum_b m_b v_b \cdot \nabla W(\mathbf{r} - \mathbf{r}_b, h) \tag{56}$$

Where the kernel function used is $W_{viscosity}$ given by (49) and its gradient is given as:

$$(-\frac{45r^2}{2\pi h^6} + \frac{30r}{2\pi h^5} - \frac{15}{4\pi h^2 r^2})\frac{\mathbf{r} - \mathbf{r_b}}{||\mathbf{r} - \mathbf{r_b}||} \tag{57}$$

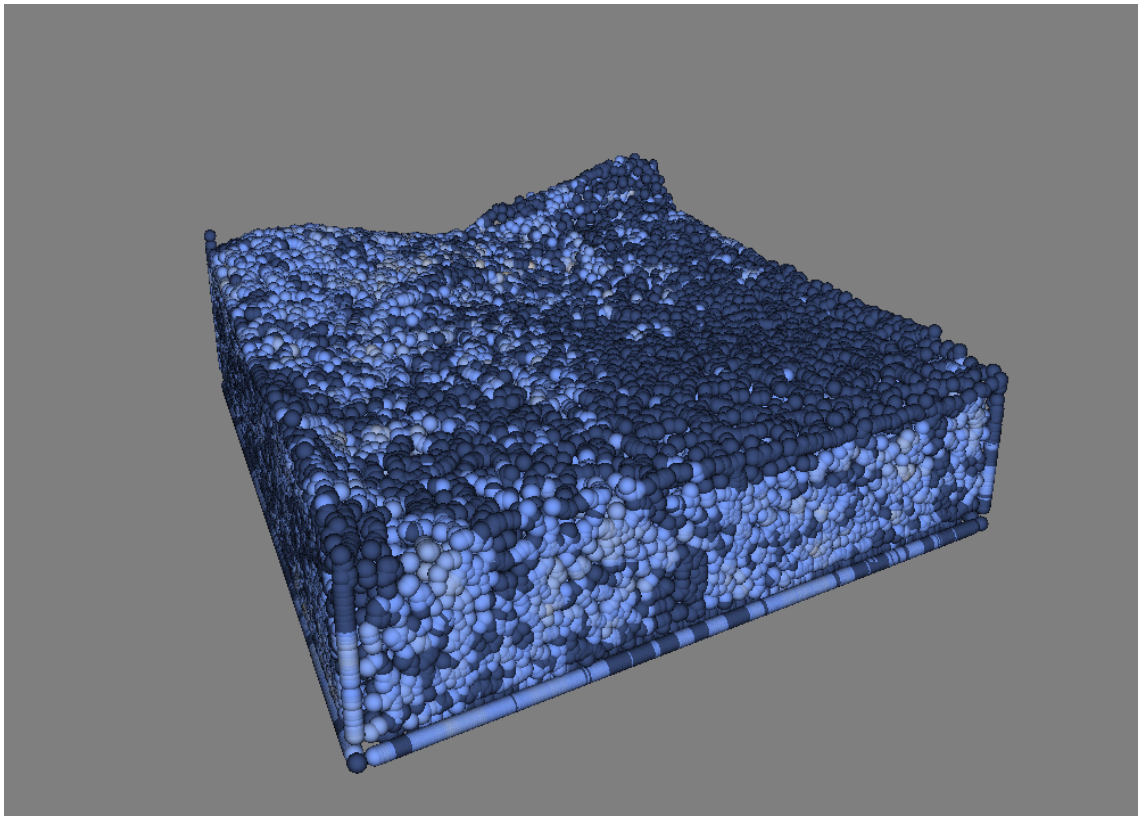

Figure 35: Divergence is color coded. The dark blue ones are particles with zero divergence.

- Also, the time step is relatively small right now, the time step we used here $6.6 \times 10^{-4}s$. This is because the large time step will cause unstablility when the viscosity is high. This issue can be addressed by using implicit viscosity solver, incorporate with the iterative correction pressure solver.

5.2 Future Works

Obviously, this simulation system has a huge room for improvement. To me, the future works mainly focus on improving simulation quality. The first thing that I can do is to allowing larger time step

by adopting iterative correction pressure solver, incorporate with implicit viscosity solver proposed in [PICT15]. Those solvers may slow down the simulation but will have better simulation results. I believe with the developing of hardware, real-time and good quality can be both achieved. If so, the 2nd and 3rd limitation can be addressed. Another idea is to incorporate it with VR kit. As mentioned in introduction part in figure 13, the VR goggle goes well with Leap Motion device. Also, my implementation of OpenGL 4 renderer has taken VR system into account. While multiple cameras are set to render the scene, VR effect can be achieved by passing camera matrix of two eyes and render them into the different lens in the goggles. This matrix is also tweakable for different VR devices.

Bibliography

[AAT13] Nadir Akinci, Gizem Akinci, and Matthias Teschner. Versatile Surface Tension and Adhesion for SPH Fluids. *ACM Trans. Graph.*, 32(6):182:1—-182:8, 2013.

[ABD+15] Jade Alglave, Mark Batty, Alastair F Donaldson, Ganesh Gopalakrishnan, Jeroen Ketema, Daniel Poetzl, Tyler Sorensen, and John Wickerson. Gpu concurrency: weak behaviours and programming assumptions. In *Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS15). ACM, New York*, pages 577–591, 2015.

[BHP+13] Mark Bolas, Perry Hoberman, Thai Phan, Palmer Luckey, James Iliff, Nate Burba, Ian McDowall, and David M Krum. Open virtual reality. In *Virtual Reality (VR), 2013 IEEE*, pages 183–184. IEEE, 2013.

[Ble90] Guy E Blelloch. Prefix Sums and Their Applications. *Computer*, (7597):35–60, 1990.

[BT07] Markus Becker and Matthias Teschner. Weakly compressible sph for free surface flows. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 209–217. Eurographics Association, 2007.

[CLRS01] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 8.2: Counting sort. *Introduction to Algorithms,*, pages 168–170, 2001.

[CM99] Paul W Cleary and Joseph J Monaghan. Conduction modelling using smoothed particle hydrodynamics. *Journal of Computational Physics*, 148(1):227–264, 1999.

[CR99] Sharen J Cummins and Murray Rudman. An sph projection method. *Journal of computational physics*, 152(2):584–607, 1999.

[DG96] Mathieu Desbrun and Marie-Paule Gascuel. *Smoothed particles: A new paradigm for animating highly deformable bodies*. Springer, 1996.

[Faw13] Hasan Fawad. Heat transfer modeling of metal deposition employing welding heat source. In *Applied Mechanics and Materials*, volume 315, pages 463–467. Trans Tech Publ, 2013.

[FVS11] Jianbin Fang, Ana Lucia Varbanescu, and Henk Sips. A comprehensive performance comparison of cuda and opencl. In *Parallel Processing (ICPP), 2011 International Conference on*, pages 216–225. IEEE, 2011.

[GA05] John A Goldak and Mehdi Akhlaghi. Thermal analysis of welds. *Computational Welding Mechanics*, pages 71–117, 2005.

[GM77] Robert A Gingold and Joseph J Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly notices of the royal astronomical society*, 181(3):375–389, 1977.

[GSSP10] 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*, pages 55–64. Eurographics Association, 2010. `https://graphics.ethz.ch/~sobarbar/papers/Sol10b/Sol10b.pdf`.

[HLL+12] Xiaowei He, Ning Liu, Sheng Li, Hongan Wang, and Guoping Wang. Local poisson sph for viscous incompressible fluids. In *Computer Graphics Forum*, volume 31, pages 1948–1958. Wiley Online Library, 2012.

[HSO07] Mark Harris, Shubhabrata Sengupta, and John D Owens. Gpu gems 3. *Parallel Prefix Sum (Scan) with CUDA*, pages 851–876, 2007.

[IABT11] Markus Ihmsen, Nadir Akinci, Markus Becker, and Matthias Teschner. A parallel sph implementation on multi-core cpus. In *Computer Graphics Forum*, volume 30, pages 99–112. Wiley Online Library, 2011.

[ICS+14] Markus Ihmsen, Jens Cornelis, Barbara Solenthaler, Christopher Horvath, and Matthias Teschner. Implicit incompressible sph. *Visualization and Computer Graphics, IEEE Transactions on*, 20(3):426–435, 2014.

[IIFS12] M Ito, S Izawa, Y Fukunishi, and M Shigeta. SPH Simulation of Gas Arc Welding Process. *Iccfd7*, pages 1–9, 2012.

[Ind09] Process Industries. Investigation of Flow Dynamics and Plastic Deformation in Arc. (December):1–6, 2009.

[KGS09] Abbas Khayyer, Hitoshi Gotoh, and Songdong Shao. Enhanced predictions of wave impact pressure by improved incompressible sph methods. *Applied Ocean Research*, 31(2):111–131, 2009.

[LC87] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *ACM siggraph computer graphics*, volume 21, pages 163–169. ACM, 1987.

[MCG03] 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*, pages 154–159. Eurographics Association, 2003.

[Mor00] Joseph P Morris. Simulating surface tension with smoothed particle hydrodynamics. *International journal for numerical methods in fluids*, 33(3):333–353, 2000.

[PICT15] Andreas Peer, Markus Ihmsen, Jens Cornelis, and Matthias Teschner. An implicit viscosity formulation for sph fluids. *ACM Transactions on Graphics (TOG)*, 34(4):114, 2015.

[PTB+03] Simon Premžoe, Tolga Tasdizen, James Bigler, Aaron Lefohn, and Ross T Whitaker. Particle-based simulation of fluids. In *Computer Graphics Forum*, volume 22, pages 401–410. Wiley Online Library, 2003.

[RC09] DAS Raj and Paul W CLEARY. Investigation of flow dynamics and plastic deformation in arc welding using sph. 2009.

[SL03] Songdong Shao and Edmond YM Lo. Incompressible sph method for simulating newtonian and non-newtonian flows with a free surface. *Advances in Water Resources*, 26(7):787–800, 2003.

[SP09] Barbara Solenthaler and Renato Pajarola. Predictive-corrective incompressible sph. In *ACM transactions on graphics (TOG)*, volume 28, page 40. ACM, 2009.

[SSP07] Barbara Solenthaler, Jürg Schläfli, and Renato Pajarola. A unified particle model for fluid–solid interactions. *Computer Animation and Virtual Worlds*, 18(1):69–82, 2007.

[THM+03] Matthias Teschner, Bruno Heidelberger, Matthias Müller, Danat Pomerantes, and Markus H Gross. Optimized spatial hashing for collision detection of deformable objects. In *VMV*, volume 3, pages 47–54, 2003.

[TMZ+11] Abdelaziz Timesli, Abdelhadi Moufki, Hamid Zahrouni, Bouaaza Braikat, and Hassan Lahmam. Numerical model based on sph method to simulate friction stir welding. In *10e colloque national en calcul des structures*, pages Clé–USB, 2011.

[vdLGS09] Wladimir J van der Laan, Simon Green, and Miguel Sainz. Screen space fluid rendering with curvature flow. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 91–98. ACM, 2009. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.443.6926&rep=rep1&type=pdf`.