If thou tastest a crust of bread, thou tastest all the stars and all the heavens.

– Robert Browning (1812-1889).

University of Alberta

Physically-Based Baking Animation Using Smoothed Particle Hydrodynamics for Non-Newtonian Fluids

by

Omar Isidro Rodriguez-Arenas

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

Examining Committee

Herb Yang, Computing Science

Pierre Boulanger, Computing Science

Faye Hicks, Civil and Environmental Engineering

Abstract

This thesis explores the mesh-free fluid simulation method of Smoothed Particle Hydrodynamics (SPH), and its application in the modelling of the baking process. A framework is proposed to generate dramatic-looking animations while maintaining the simulation physically plausible. In this framework, SPH can successfully model non-Newtonian fluids with the different physical changes that take place during the baking process including 1) volume expansion due to the increase the fluid's inner pressure, 2) the fluid-solid phase change due to the change in the mechanical properties of the fluid, and 3) the surface browning that gives the end result a more realistic look. An adaptive field function is proposed for the purpose of reconstructing a surface from the results of the SPH simulation, allowing for the expansion of the fluid to take place while maintaining its volume whole. A qualitative comparison is presented between the resulting animations and images of different types of baked goods. Moreover, the future research to the SPH method is explored along with a discussion on how the SPH method can be further improved to create better animations of the baking process.

Acknowledgements

First and foremost, I wish to thank Dr. Herb Yang for his continuous support during my studies; the pool of knowledge I tapped in his shared experiences and vast knowledge helped to guide the final results of my research.

I would also like to thank the members of my examining committee Dr. Pierre Boulanger and Dr. Faye Hicks for taking the time to examine my work in depth. Their invaluable feedback improved greatly the quality of this report.

My gratitude also goes towards current and former members of the Computer Graphics Lab at the University of Alberta: Xida Chen, Jason Gedge, Dr. Cheng Lei, Dr. Daniel Neilson, Gopinath Sankar, Yufeng Shen and Yilei Zhang. The knowledgeable conversations provided valuable insight to numerous aspects of my thesis.

Furthermore, I would also like to thank my family: Roberto Rodriguez, Clarissa Arenas and Sofia Arenas for their constant and unyielding support all through my life.

And last but not least I would like to thank my life partner Mariana Paredes-Olea who was by my side through this process to share ideas, help organize my thoughts, and give meaning to my life. Without her, the work presented in this thesis would not have been possible.

Table of Contents

List of Figures

List of Symbols

ψ	3D indicator function used in the Poisson surface reconstruction method
A	3D position vector function
$\mathbf{x}$	3D position
$\mathbf{a}$	Acceleration
W_v	Activation energy for viscous flow
$\mathbf{L}$	Angular momentum
ω	Angular velocity
λ_a	Apparent thermal conductivity
η	Apparent viscosity
N_a	Average number of neighbours of a 3D point
α_ϕ	Base radius of the adaptive field function
b	Behaviour of powerlaw index
ρ_b	Bubble density
P_b	Bubble pressure
V_b	Bubble Volume
O	Bubble-dough particle differentiation function
$\mathbf{x}_{cm}$	Centre of mass
η_c	Cinematic viscosity
C_b^*	CO_2 equilibrium concentration for a bubble
G	Colour contribution function
C	Colour property
X_c	Concentration of CO_2
γ	Constant in Tait's Equation
α_Π	Constant in the computation of artificial viscosity related with bulk viscosity
β_Π	Constant in the computation of artificial viscosity related with von Neumann-Richtmyer viscosity
ϵ_Π	Constant in the computation of artificial viscosity used to prevent numerical divergences
C_a	Cross section surface area

D_c	Current working tree depth
t_ϕ	Density field function threshold
a_ϕ	Density field function user defined value
b_ϕ	Density field function user defined value
ϕ	Density field function
ρ	Density
d	Depth of the root of the subtree
δ	Dirac delta function
$\mathbf{u}$	Displacement from the material coordinates
r_{ij}	Distance between particles i and j
r_ϕ	Distance to point in evaluation
$\mathscr{P}$	Distinct patch of ∂M
μ_e	Elasticity coefficient
$\mathbf{D}_e$	Elasticity matrix
χ	Equation whose answer defines an isosurface
$\mathbf{f}$	External forces
ρ_f	Final density
$\mathscr{A}$	Finite point set
ζ_c	Fluid consistency index
ζ_f	Fluid flow index
B	Gas constant
R_g	Gas constant
$\dot{m}_g$	Gas mass flow leaving the cake
G_p	Gas phase partition coefficient
$\bar{g}$	Geometric mean of the initial density estimates
ε_G	Green strain
$\dot{Q}$	Heat flow rate
k_h	Heat transfer coefficient
H_e	Height
I	Identity matrix
$\mathbf{I}$	Inertia stress tensor
D	Intensity of deformation
J	Jump number
W	Kernel function
$\mathbf{S}_K$	Kirchhoff stress
ΔH_v	Latent heat of water
M_f	Mass fraction
K	Mass permeability
m	Mass

α_e	Matrix element
β_e	Matrix element
M_w	Moisture
N	Neighbouring particles that fall within the support of x
ε_n	Non-elastic strain
$\mathbf{N}$	Normal vector
α_c	Normal velocity coefficient
$\mathbf{N}_s$	Normalized sum of the normal vectors belonging to the facets with which particle collided
b	Nth time step
N_c	Number of CO_2 gas cells in dough
$n_{\mathrm{CO_2}}$	Number of CO_2 in moles
k	Number of dimensions being subdivided in the k-d tree data structure
d	number of dimensions
N_s	Number of particles in each search
N_t	Number of particles that define a fluid
D_L	Overall CO_2 transfer coefficient from dough to the bubble
ϵ_p	Porosity
P	Pressure
β_ϕ	Radius coefficient of the adaptive field function
r	Radius
R_c	Rate of CO_2 generation
C_λ	Relaxation time scaling factor
λ	Relaxation time
$\mathbf{R}$	Rotational tensor
y	Sample in Q
k_g	Scaling factor of the order of unity
ϵ_g	Sensitivity parameter that exists in the interval $0 \leq \epsilon_g \leq 1$
S	Set of points that will make up the kd-tree
Y	Set of samples
$\tilde{F}$	Smoothing filter used in the Poisson surface reconstruction method
h	Smoothing length
C_{heat}	Specific heat capacity
c	Speed of sound
α	Squared radius of the sphere used to construct the alpha-shape
ε_S	Strain
$\mathbf{S}$	Stress tensor

β_c	Surface friction parameter
	Surface in the Poisson surface reconstruction method
σ	Surface tension
T	Temperature
e_t	Thickness
t	Time
τ	Torque vector
$\mathbf{D}'$	Traceless stress tensor
$\tilde{F}_x$	Translation of the smoothing filter used in the Poisson surface reconstruction method
M	Unknown model in the Poisson surface reconstruction method
U	Value to e used as the pair UV coordinates
$\mathbf{V}$	Vector field used in the Poisson surface reconstruction method
$\mathbf{D}$	Velocity gradient
$\mathbf{v}$	Velocity
μ	Viscosity coefficient
Π	Viscosity tensor
Ω	Volume of the integral that contains x
V	Volume
ΔH_{vap}	Water vaporization enthalpy
ϵ	XSPH velocity tuner
E	Youngs modulus

Chapter 1

Introduction

Fluid flow simulations are a problem of great interest for different fields ranging from medicine to engineering to the entertainment industry. Goals and requirements for simulating fluids are varied as well: its use by industrial and engineering settings focus on accuracy as the main objective, and results obtained can be the difference between a successful procedure or a disaster of great consequences. In applications for the entertainment industry; however, accuracy may take the back seat in favour of either speed for real-time software such as video games, or for a more dramatic or visually appealing animation.

Also, for computer graphic researchers, many fluid phenomena have been of interest in the past: water splashing, the movement of fire, smoke swirling in the air, etc. This thesis presents another example. The simulation of the baking process is an ever-present, and important part of our daily lives; furthermore, it presents many interesting fluid phenomena such as volume expansion, and fluid-solid phase change. Even though this process is generally perceived as a simple process, it is very complex to simulate due to the many physical and chemical changes that occur but that are not completely understood. Our engineering understanding of the baking process is very limited, and recently there has been an increase in research interest for simulating and visualizing baking phenomena, resulting in varied attempts to create an accurate mathematical model. One example is the work done by Zhang et al. [91] [92] [90], Fan et al. [25], and Lostie et al. [49] (described in detail in Chapter 2). However, the goal of the research presented in this thesis is to create appealing animations with the purpose of entertainment. For this reason, different assumptions will be made in contrast to more rigorous work where the result of a dramatic animation will be favoured.

Due to the interesting research challenges posed by the problem of animating fluids' movements, there have been many models that are designed for simulating specific phenomena, like Enright et al.'s [23] method for the animation and rendering of photo-realistic water effects, Lenaerts et al.'s [42] work on porous flow simulations, and Stora et al.'s [82] simulation of lava flows, to cite a few. However, the baking process has been largely untouched by researchers in the area of computer graphics animation.

Dough can be difficult to model because it is generally a non-Newtonian fluid. The difference between a Newtonian fluid (i.e. water, air, wine) and a non-Newtonian fluid (i.e. mayonnaise, peanut butter, bread dough) is that for a Newtonian fluid, viscosity is only dependent on temperature, whereas for a non-Newtonian fluid, viscosity can change as the shear rate changes. Non-Newtonian fluids present very interesting behaviours such as the Weissenberg effect where the fluid will climb up a rod used to mix it inside a container. Such a phenomenon is often observed when mixing batter with an electric beater. Another example is the "memory" effect, characteristic of some non-Newtonian fluids, that causes them to maintain its shape until a strong enough force is applied. Examples of non-Newtonian fluids can be seen in Figure 1.1.

Due to the difficulty of accurately modelling non-Newtonian fluids in computer graphics an often taken recourse is to fake the non-Newtonian fluid. On the contrary, because the proposed framework will simulate the baking process, it has to be able to model non-Newtonian fluids to represent the dough. To do so, the present work builds upon the research done by Mao and Yang [51] [52] [53] using the Smoothed Particle Hydrodynamics (SPH) framework.

The SPH framework is a mesh-free computational method for simulating fluid

(a) Honey holding on to a honey dipper from [39].

(b) Peanut butter holding its shape from [41].

Figure 1.1: Examples of non-Newtonian fluids.

flow that was initially developed for astrophysical simulations by Lucy [50], Gingold et al. [28]. Roy [74] along with Desburn and Gascuel [17] were the first to use SPH for generating animations for entertainment purposes.

The work presented in this thesis expands the SPH framework with the following contributions:

- An improved method to achieve the fluid-solid phase change based on the work by Mao and Yang [51] [52].
- An original method to simulate the volume expansion of a fluid by modelling the increase in pressure of gas trapped inside the fluid.
- An adaptive density function used to create a mesh that is capable of illustrating the results from the volume expansion.

These new phenomena are then used in tandem to create animations of the baking process, which takes into account volume expansion, solidification, and surface browning phenomena that take place during baking.

The body of this thesis is organized as follows. Chapter 2 summarizes background information and related works on fluid simulation, smoothed particle hydrodynamics; also, mathematical models of the baking process are discussed. Chapter 3 describes the implementation details of the SPH framework along with a description of the new phenomena; furthermore, the method for reconstructing surfaces from the results of the SPH framework are discussed. Chapter 4 includes information on how the new phenomenon is put together to simulate the baking process. Finally, Chapter 5 contains a discussion on the results obtained from applying the SPH framework to model the baking process, as well as a brief discussion on possible future work.

Chapter 2

Previous work

The purpose of this Chapter is to present the background knowledge upon which the work presented in this thesis is based. Section 2.1 provides a brief overview of the equations that govern most fluid flow of interest to computer graphics animations known as the Navier-Stokes equations. In Section 2.2 an introduction to the fundamentals of the framework used to create the animations presented in this thesis is given as well as a review of different attempts at creating different hydrodynamic phenomena using the same framework. Section 3.5 gives a brief review on the works of surface reconstruction, an important aspect used to visualize the results of the fluids simulations. Finally Section 2.4 reviews some of the attempts at modelling the baking process.

2.1 Fluid simulation

The goal of computational fluid dynamics (CFD) is to predict the behaviour of fluids using the Navier-Stokes equations (NS) [67]. These equations can be solved using either an Eulerian or Lagrangian representation of the fluid flow.

In the Eulerian representation the fluid's properties are measured as it flows through static control volumes. The NS equation for the flow of incompressible Newtonian fluids is described on Equation 2.1:

$$\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} = -\frac{1}{\rho} \nabla P + \mu \nabla^2 \mathbf{v} + \mathbf{f} \tag{2.1}$$

where **v** is the flow velocity, t time, ρ the fluid density, P the pressure, $\mu\nabla^2\mathbf{v}$ the viscosity, and **f** represents all the other external forces being exerted on the fluid such as gravity or centrifugal force.

In contrast to the Eulerian representation, the Lagrangian representation works as a particle system, where the fluid is discretized into particles each with its own position and velocity. A Lagrangian model can have a mesh that connects these particles or it can be mesh-free.

The NS equation for particle motion in its Lagrangian representation takes the form described in Equation 2.2:

$$\frac{d\mathbf{v}}{dt} = -\frac{1}{\rho} \nabla P + \mu \nabla^2 \mathbf{v} + \mathbf{f} \tag{2.2}$$

The Eulerian and Lagrangian forms of the NS equation are very similar, the difference being in that the left hand side of the Eulerian form in Equation 2.1 $\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v}$ is replaced by the substantive derivative $\frac{d\mathbf{v}}{dt}$ in Equation 2.2 which can be intuitively thought of a property's time rate change of measured by an observer moving with the specific particles under study. Because the particles move with the fluid, the substantive derivative of the velocity field becomes the time derivative of the velocities of the particles omitting the advection term $\mathbf{v} \cdot \nabla \mathbf{v}$. A detailed comparison of the Eulerian and Lagrangian forms of the NS equation can be found in References [72] and [64] .

The work presented in this thesis is based on the model known as *smoothed particle hydrodynamics* (SPH) which is a mesh-free Lagrangian approach for simulating fluid flows.

2.2 Smoothed particle hydrodynamics

2.2.1 SPH fundamentals

The SPH method was initially developed by Lucy [50] and Gingold et al. [28] for astrophysical simulations but since then it has been extended to different types of fluid simulations.

The key aspect in the SPH formalism is known as the integral representation of a function $A(\mathbf{x})$. This concept begins with the following identity:

$$A(\mathbf{x}) = \int_{\Omega} A(\mathbf{x}')\delta(\mathbf{x} - \mathbf{x}')d\mathbf{x}' \tag{2.3}$$

where A is a function of the 3D position vector **x** and Ω the volume of the integral that contains **x** and $\delta(\mathbf{x} - \mathbf{x}')$ the Dirac delta function

$$\delta(\mathbf{x} - \mathbf{x}') \begin{cases} 1, & \mathbf{x} = \mathbf{x}' \\ 0, & \mathbf{x} \neq \mathbf{x}' \end{cases} \tag{2.4}$$

An approximation to the integral representation of Equation 2.3 can be obtained by replacing the Dirac delta function with a smoothing function $W(\mathbf{x}-\mathbf{x}', h)$, where h is called the *smoothing length* and it defines the radius of the volume of influence of the smoothing function W.

Because in the SPH method the system is represented by a finite number of particles, the integral representation must be further approximated using what is known as particle approximation. This is achieved by converting Equation 2.3 into a discretized form of summation over all the particles in the support domain as shown in Figure 2.1.

The infinitesimal volume $d\mathbf{x}'$ in Equation 2.3 at the location of particle $subj$ can be replaced by a finite volume of the particle ΔV_j:

$$A(\mathbf{x}_i) = \sum_{j=1}^{N} A(\mathbf{x}_j)W(\mathbf{x}_i - \mathbf{x}_j, h)\Delta V_j \tag{2.5}$$

where N is the number of particles that fall within the support of position **x**, j the particle index, and $\mathbf{x}_j$ the position for particle j. The finite volume ΔV can be expressed as

$$\Delta V_j = \frac{m_j}{\rho_j} \tag{2.6}$$

where ρ_j and m_j are the particle's density and mass respectively. With this the continuous function $A(\mathbf{x})$ can be written as

$$A(\mathbf{x}_i) = \sum_{j=1}^{N} \frac{m_j}{\rho_j} A(\mathbf{x}_j) W(\mathbf{x}_i - \mathbf{x}_j, h) \tag{2.7}$$

Equation 2.7 states that the value of function $A(\mathbf{x})$ at particle **x** is given by using the average of the values of the function at all the particles that fall under the support domain of particle **x**. Furthermore, the smoothing function W acts as a weight function determining how much the value $A(\mathbf{x}_j)$ at the neighbouring particle contributes to $A(\mathbf{x})$.

Similar to $A(\mathbf{x}_i)$ the gradient $\nabla A(\mathbf{x}_i)$ is given as

$$\nabla A(\mathbf{x}_i) = \sum_{j=1}^{N} \frac{m_j}{\rho_j} A(\mathbf{x}_j) \nabla W(\mathbf{x}_i - \mathbf{x}_j, h) \tag{2.8}$$

and the Laplacian $\nabla^2 A(\mathbf{x})$ is given by

$$\nabla^2 A(\mathbf{x}_i) = \sum_{j=1}^{N} \frac{m_j}{\rho_j} A(\mathbf{x}_j) \nabla^2 W(\mathbf{x}_i - \mathbf{x}_j, h) \tag{2.9}$$

The weight function W can take different forms, modifying the accuracy and stability of the simulation depending of the case when it is used. As stated by Liu et al. [45] to get valid results the smoothing function must have at least the following properties:

Unity The smoothing function must be normalized over its support domain.

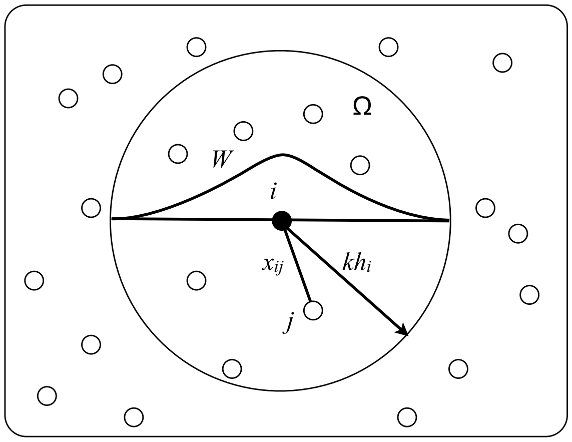


Figure 2.1: SPH support domain.

$$\int_{\Omega} W(\mathbf{x} - \mathbf{x}', h) d\mathbf{x}' = 1 \tag{2.10}$$

Compact support The smoothing function should have a compact support.

$$W(\mathbf{x} - \mathbf{x}') = 0, \text{ for } |\mathbf{x} - \mathbf{x}'| > kh \tag{2.11}$$

This property is mainly for algorithmic purposes making the SPH computation a local operation. Consider the use of a Gaussian smoothing kernel that is not compactly supported, this would make the algorithm to have a complexity of $O(n^2)$ rendering it impractical.

Positivity $W(\mathbf{x} - \mathbf{x}') \geq 0$ for any point at $\mathbf{x}'$ within the support domain of the particle at point x.

If negative values are allowed in the computation, simulations of physical phenomena may give erroneous results with negative density and energy.

Decay The smoothing function value for a particle should be monotonically decreasing with the increase in distance away from the particle.

It is logical to think that the closer the particles are, the more they will affect each other.

Delta function property The smoothing function should satisfy the Dirac delta function condition as the smoothing length approaches zero.

$$\lim_{h\to 0} W(\mathbf{x}-\mathbf{x}', h) = \delta(\mathbf{x}-\mathbf{x}') \tag{2.12}$$

This property ensures that as the distance between the particles approaches zero, the approximation of the value approaches the function value.

Symmetric property The smoothing function should be an even function.

Two particles at different positions but separated by the same distance should be affected equally.

Smoothness The smoothing function should be sufficiently smooth, or in other words, the smoothing function needs to be sufficiently continuous to obtain good results.

An in-depth discussion on how each of the terms of the NS equation are approximated using the SPH formalism on the work presented in this thesis can be found in Chapter 3.

2.2.2 Hydrodynamic phenomena with SPH

Roy [74] showed how the SPH framework can be used in the context of computer graphics by animating weakly compressible fluids. Later, Desburn and Gascuel [17] introduced SPH to the computer graphics community in general with their work on highly deformable bodies. Since then, the computer graphics community has embraced this method for simulating fluid flows extending it to simulate many hydrodynamic phenomena (for examples see below).

Due to the nature of SPH, a particularly interesting phenomena to recreate using this framework is bubbly liquids. The bubbles can be obtained by altering the behaviour of a subset of the fluids particles.

Bubbles

Thürey et al. [84] propose a framework to create real-time simulations of bubbles and foam by combining several fluid animation techniques. In their approach, the volume of the fluid is represented with a shallow water simulation. The bubbles inside the volume are represented with a particle based simulation. And finally the foam on the surface of the water is represented with SPH. In the foam simulation, each particle represents a bubble, but for the surface reconstruction of the foam the entities are not separate; that is, there is no thin fluid sheet between them. Instead they form the overall foam volume. Results from their hybrid approach can be seen in Figure 2.2.

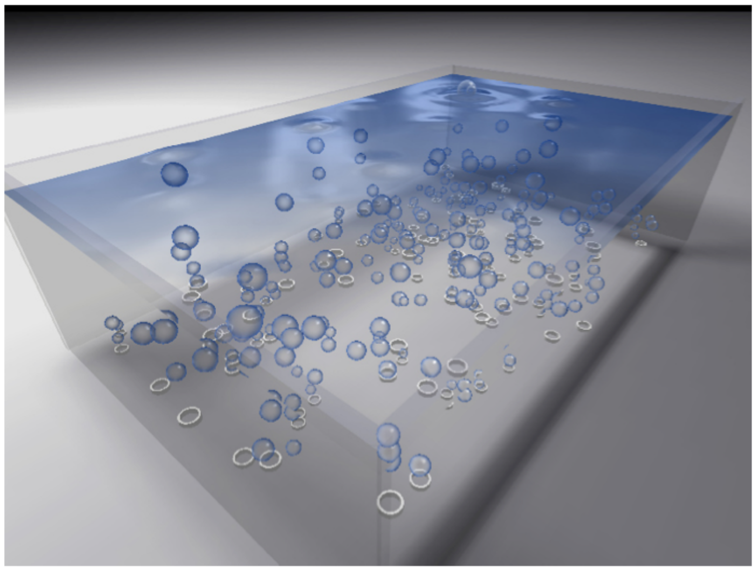


Figure 2.2: Thürey et al. [84] results on bubble and foam simulation.

Cleary et al. [14] also present their work on bubbling and frothing liquids with a different approach than Thürey et al. [84]. In their work both the fluid and the bubbles are simulated using SPH to create an animation offline. Firstly the fluid volume is generated and in a second stage the motions of the bubbles are obtained using a similar SPH framework adding buoyancy to the particles as well as spring forces between the particles to avoid the overlap of bubbles. Because the animation is generated offline and in two stages, this framework can handle a higher number of particles when compared to the one proposed by Thürey et al. [84] as illustrated by Figure 2.3.

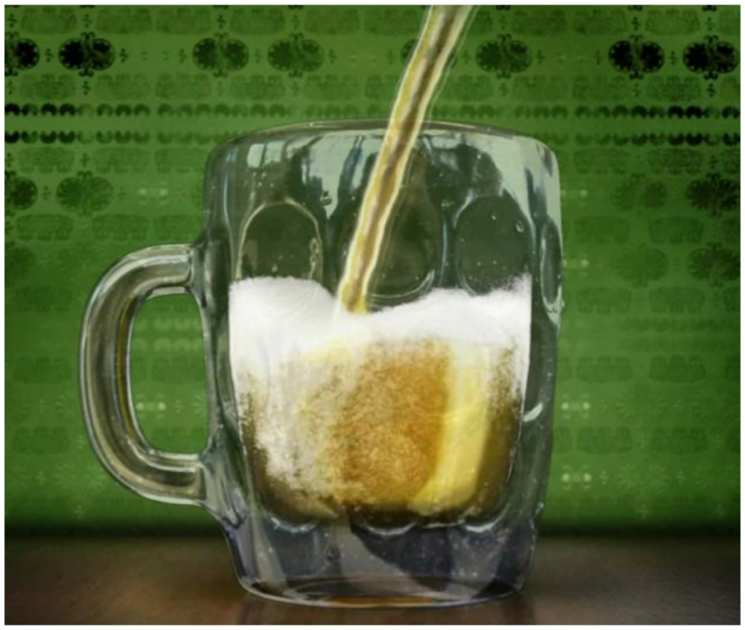


Figure 2.3: Cleary et al. [14] results animating frothy and foaming liquids.

Fluid/Solid phase transitions

Stora et al. [82] use the SPH framework to model the flow of lava. To achieve a realistic animation, the viscosity of the lava flow is not considered constant throughout the fluid. Instead, the viscosity at each particle is tied to its temperature. Two types of heat transfer are considered in the simulation: the lava's internal heat transfer and the heat transferred that occurs between the lava's surface and its surrounding environment; this allows the lava to cool off as time progressed and the fluid's viscosity increases.

The heat transfer inside the material is modelled by integrating the general heat equation:

$$\frac{dT}{dt} = k_{hint} \nabla^2 T, \tag{2.13}$$

where k_{hint} is the fluid's internal heat coefficient and T is the particle's temperature.

In their approach, they compute $\nabla^2 T$ under the SPH formalism as

$$\nabla T_i = \sum_{j \neq i} m_j \frac{T_j - T_i}{\rho_j} \nabla W(\mathbf{x}_i - \mathbf{x}_j, h) \tag{2.14}$$

$$\nabla^2 T_i = \sum_{j \neq i} m_j \frac{\nabla T_i}{\rho_j} \nabla W(\mathbf{x}_i - \mathbf{x}_j, h) \tag{2.15}$$

The heat transfer at the boundary of the lava is modelled in the following manner

$$\left(\frac{dT}{dt}\right)_{ext_i} = k_{hext}(T_i - T_{ext}) \frac{r_i^2}{\rho_i} \tag{2.16}$$

where k_{hext} is the fluid's external heat coefficient and the small radius r_i of the lava being represented by particle i is computed as

$$\frac{4}{3}\pi r_i^3 = \frac{m_i}{\rho_0} \tag{2.17}$$

Figure 2.4 illustrates lava flowing down a volcano simulated using the method proposed by Stora et al. [82].

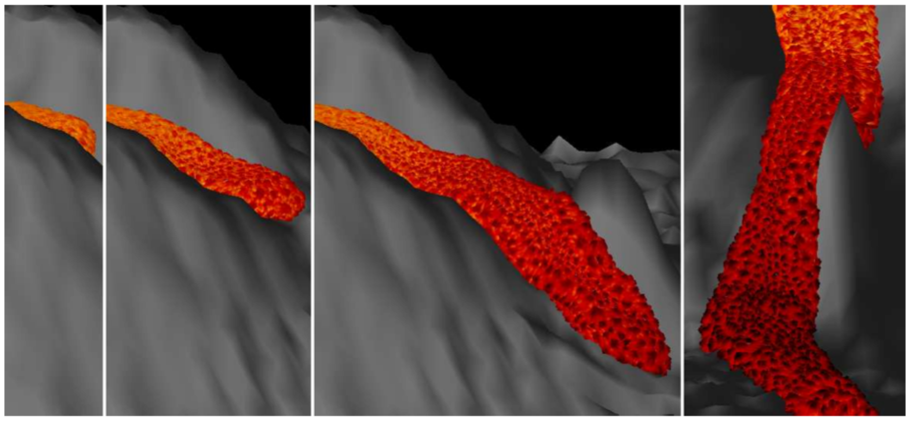


Figure 2.4: Stora et al. [82] results on simulating lava flow.

Mao and Yang [51] [52] [53] proposed a SPH-based method to create animations with melting and solidification effects due to heat transfer in non-Newtonian fluids along with immiscible fluid-fluid collisions. To create the animations for non-Newtonian fluids Mao and Yang [51] [52] [53] add a stress tensor term describing the fluid's elasticity to the NS equation of momentum:

$$\frac{d\mathbf{v}}{dt} = -\frac{1}{\rho}\nabla P + \nabla \cdot \mathbf{S} + \mu\nabla^2\mathbf{v} + \mathbf{f} \tag{2.18}$$

As stated by Mao and Yang [68] [29], the stress tensor for a non-Newtonian is a non-linear function of the velocity gradient and is too complex to compute directly. One way of avoiding this complexity is to compute the stress tensor by integrating the stress tensor rate, which is computed with the constitutive equation. The constitutive equation presented in their work is based on a non-linear Maxwell model [22]:

$$\frac{d\mathbf{S}}{dt} = \mathbf{R} + \mu_e\mathbf{D}' - \frac{1}{\lambda}S \tag{2.19}$$

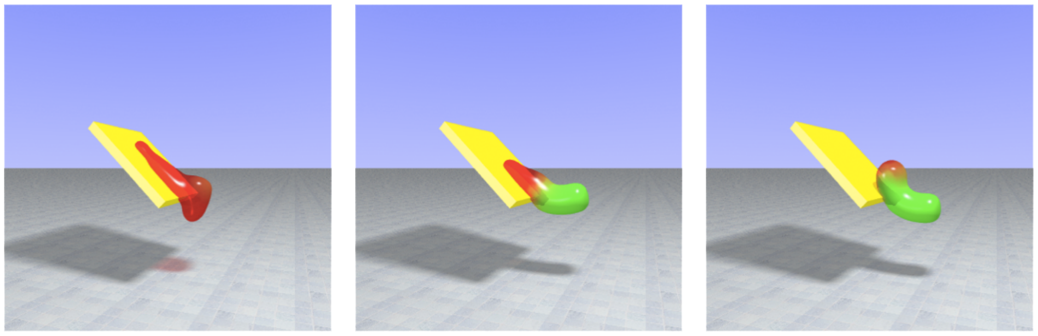
where S is the stress tensor, μ_e the elasticity constant, λ the relaxation time and **R** a rotational tensor.

To achieve the melting and solidification effects the elasticity constant μ_e and relaxation time λ are manipulated based on the particles temperature.

The proposed method manipulates the stress tensor to achieve the melting effects based on the particle's temperature. In this work the heat transfer can occur at the fluid-fluid and fluid-solid boundaries. Screenshots showing the results of melting effects are shown in Figure 2.5.

Keiser et al. [36] propose a method to create fluid animations using the SPH framework that combines the equations of solid mechanics with the NS equations using a Lagrangian approach. In their method they combine the equation for an elastic model

$$\rho\frac{d^2\mathbf{u}}{dt^2} = \nabla\mathbf{S}^s(\mathbf{u}) + \mathbf{f}, \tag{2.20}$$



(a) Fast heat transfer. Whole fluid bar melts.

(b) Slow heat transfer. Half of the fluid bar melts and the other half looks like a solid.

(c) Very slow heat transfer. Fluid bar melts only at the tip and the rest looks like a solid.

Figure 2.5: Mao and Yang's [52] results highlighting fluid bars with different heat coefficients dropped onto a hot plate.

with that for an incompressible Newtonian fluid

$$\rho \frac{d^2\mathbf{u}}{dt^2} = \nabla \mathbf{S}^f(\mathbf{v}) + \mathbf{f}, \tag{2.21}$$

taking the form of

$$\rho \frac{d^2\mathbf{u}}{dt^2} = \nabla \mathbf{S}(\mathbf{u}, \mathbf{v}) + \mathbf{f}, \tag{2.22}$$

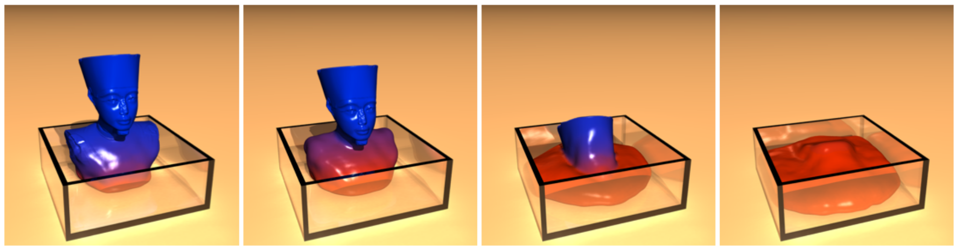
where $\mathbf{u}$ is the displacement from the material coordinates, and $\mathbf{S}(\mathbf{u}, \mathbf{v}) = \mathbf{S}_{solid}(\mathbf{u}) + \mathbf{S}_{fluid}(\mathbf{v})$ the sum of the elastic, viscous and pressure stress.

Their proposed method can output a wide array of phenomena since it can handle the fluid's stiffness, compressibility, plasticity, viscosity and cohesion between particles as parameters. Figure 2.6 shows two of the published results from using their method.

Changes in temperature are also the driving force behind the possible phase changes animations. Heat transfer is performed in a similar way as it is done by Stora et al. [82]. In order to achieve the phase change animations any of the afore mentioned parameters can be linearly interpolated.

Paiva et al. [70] also developed a technique for simulating the fluid/solid phase changes using the SPH framework. Their approach is to model solids as non-

(a) Freezing a quicksilver fluid which is poured into a glass. After removing the glass, the elastic solid bounces onto the ground and fractures.



(b) An elastic solid is dropped onto a heated box and slowly melts into a viscous fluid.

Figure 2.6: Keiser et al. [36] results showing solidification and melting phenomena.

Newtonian fluids with very high viscosity. In order to do so, they employ the generalized Newtonian fluid model proposed by Mendes et al. [18]. The NS equation for momentum is described as:

$$\frac{d\mathbf{v}}{dt} = -\frac{1}{\rho}\nabla P + \frac{1}{\rho}\nabla \cdot \mathbf{S} + \mathbf{f} \tag{2.23}$$

where the viscoplastic stress tensor field **S** for a non-Newtonian fluid is computed as the product of the apparent viscosity η—which is dependent on the intensity of deformation D—and the velocity gradient **D**:

$$\mathbf{S} = \eta(D)\mathbf{D}. \tag{2.24}$$

The intensity of deformation D is defined as:

$$D = \sqrt{\frac{1}{2} \cdot tr(\mathbf{D})^2} \tag{2.25}$$

where tr is the trace operation defined in linear algebra as the sum of the elements on the main diagonal of the matrix, and the velocity gradient **D** which describes the difference in velocity between adjacent fluid layers is defined as:

$$\mathbf{D} = \nabla \mathbf{v} + (\nabla \mathbf{v})^T. \tag{2.26}$$

The apparent viscosity η in Equation 2.24 is modelled as an exponential as defined by Equation 2.27:

$$\eta(D) = (1 - \exp[1 - (J+1)D]) \left(D^{b-1} + \frac{1}{D} \right). \tag{2.27}$$

where b is the behaviour of powerlaw index and J is the jump number. The exponential depends on the parameter J which incorporates many rheological properties such as yield stress, low shear rate viscosity and the consistency index.

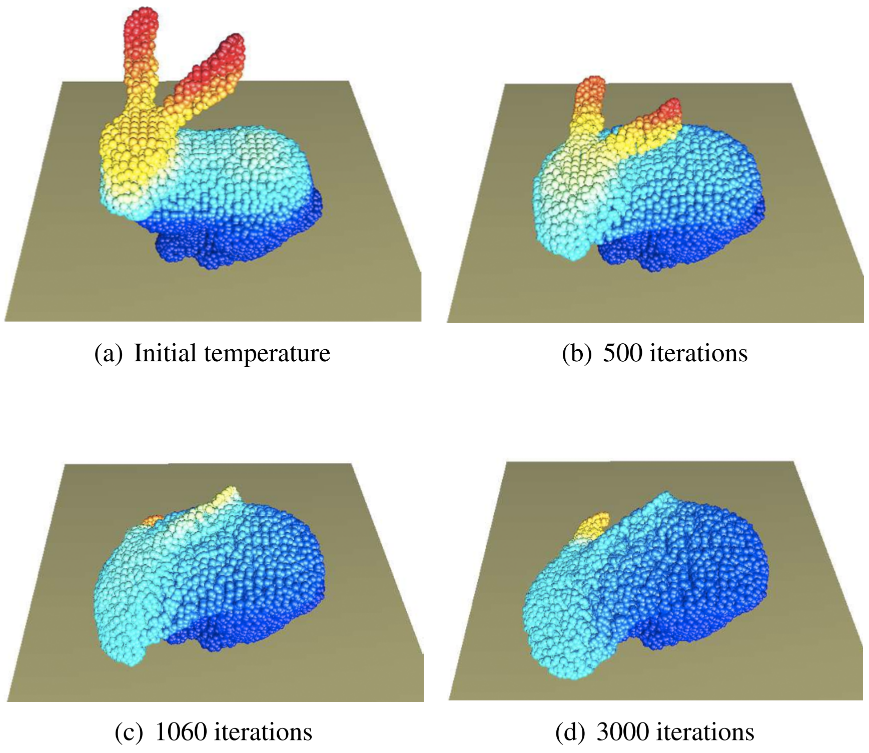
In order to simulate the melting phenomena, J is linearly interpolated based on the particle's temperature. Figure 2.7 illustrates the results of this method in melting the Stanford bunny.

Solenthaler et al. [79] create a unified particle model for fluid-solid interactions that facilitates creation of animations containing rigids, elastics, rigid-elastics, fluids and the melting and solidification phenomena as well as the distinction between multiple close deformable objects and parts of the same deformable object. Their proposed model is similar to that proposed by Keiser et al. [36] when considering the fluid and elastic aspects of the animation; however, Solenthaler et al. [79] take a different approach when dealing with solids: the forces acting upon a rigid body are accumulated and the movement of the body formed by the particles is restricted to translations and rotations. Based on the work presented by Baraff [4] the rotation is handled explicitly by computing a torque vector τ_i as

$$\tau_i = (\mathbf{x}_i - \mathbf{x}_{cm}) \times \mathbf{f}_i \tag{2.28}$$

where $\mathbf{x}_{cm}$ is the centre of mass of the body formed by the particles and $\mathbf{f}_i$ is the sum of all forces calculated with SPH exerted on particle i.

The total force applied to a body is given by


(a) Initial temperature (b) 500 iterations

(c) 1060 iterations (d) 3000 iterations

Figure 2.7: Melting results from Paiva et al. [70].

$$\mathbf{f}_{body} = \sum_{i_{body}} \mathbf{f}_i \tag{2.29}$$

and similarly the total torque is defined as

$$\tau_{body} = \sum_{i_{body}} \tau_i \tag{2.30}$$

After $\mathbf{f}_{body}$ and τ_{body} are computed, time integration is performed by first iterating over a rigid object to calculate the linear and angular velocity of the body. The angular velocity ω is defined as

$$\omega = \mathbf{I}^{-1}\mathbf{L} \tag{2.31}$$

where $\mathbf{I}$ is the inertia tensor and $\mathbf{L}$ the angular momentum, which is updated at every time step as

$$\mathbf{L} \leftarrow \mathbf{L} + \tau \Delta t \tag{2.32}$$

The last step in the time integration is then to update the particles belonging to the body to reflect the changes.

In Solenthaler et al.'s [79] work, the heat transfer between neighbouring particles and outside influences is modelled following the work proposed by Stora et al. [82]. Aside from storing the current temperature, each particle stores the melting and solidification points T_{melt} and T_{solid} respectively. Figure 2.8 shows how the particle is handled based on the temperature giving way to the different phase changes, it the particles' temperature fall in the range of $0 \leq T \leq T_{solid}$ the particles behave as a rigid/elastic. As the particles' temperature increases and reaches the range $T_{solid} < T \leq T_{melt}$ they behave as an elastic material. Finally, when the particles' temperature reaches $T_{melt} < t$ the material takes the properties of a fluid. Their results are illustrated on Figure 2.9.

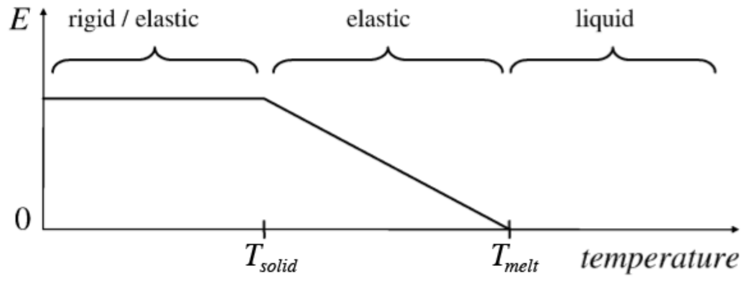


Figure 2.8: Solenthaler et al. [79] stages of the phase change process.

2.3 Surface reconstruction

Once the positions of particles have been calculated, the next step towards creating the animation is to render the scene. Rendering each of the individual particles can give an idea of the shape of the fluid for each frame; however, this method lacks the

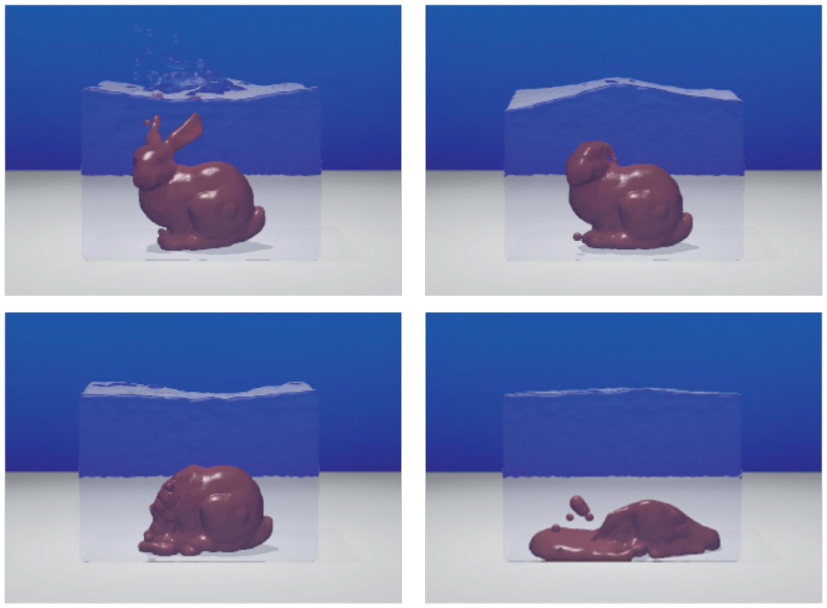


Figure 2.9: Solenthaler et al. [79] results showing melting phenomena.

shape description of the fluid. Another approach is to reconstruct a mesh based on the positions of particles that will give a more accurate description.

Isosurface

Isosurfaces have been widely used for this purpose since the method was introduced by Blinn [10]. An isosurface is a class of surface which can be defined as the solution to some equation

$$\chi(x, y, z) = 0 \tag{2.33}$$

To generate the surface a density field is needed. This density field is generated using a function ϕ which increases in value as the distance to the point where it is evaluated grows smaller. An implicit surface would then be defined as the set of points where the density function is equal to some threshold t_ϕ:

$$\chi(x, y, z) = \phi(x, y, z) - t_\phi \tag{2.34}$$

Once the density field has been computed the resulting mesh can be extracted using the Marching Cubes algorithm [48].

Blinn [10] coined the term “blobby molecules” for his results by defining the density function as

$$\phi(r_\phi) = b_\phi e^{-a_\phi r^2} \tag{2.35}$$

where a_ϕ and b_ϕ are user defined parameters that allow to alter the “blobbiness” of the object.

A problem with using Equation 2.35 as the density function is that it has to be evaluated everywhere, since it extends to infinity.

Wyvill et al. [88] propose another density function that avoids this problem as described in Equation 2.36:

$$\phi(r_\phi) = \begin{cases} a_\phi(1 - \frac{r_\phi^6}{9b_\phi^6} + \frac{17r_\phi^4}{9b_\phi^4} - \frac{22r_\phi^2}{9b_\phi^2}) & r_\phi \leq b_\phi \\ 0 & \text{otherwise} \end{cases} \tag{2.36}$$

Here a_ϕ also works as a scaling factor and b_ϕ delimits ϕ function’s influence. This function has a slight advantage over the metaballs in that it uses the squares of the distance so it does not need to compute the square roots.

There are many approaches for constructing an isosurface from a set of points. One of these methods is proposed by Kazhdan [35] as illustrated on Figure 2.10. This method redefines the problem of reconstructing a surface starting from an oriented set of points as a spatial Poisson problem (Figure 2.10(a)). As an implicit function framework, the Poisson reconstruction method uses a 3D indicator function ψ which is defined as 1 when evaluated at a point inside the model and 0 otherwise (Figure 2.10(b)). The nature of this method is the integral relationship between oriented points sampled from the surface of a model and the indicator function of that model. That is, the oriented point samples can be viewed as samples of the gradient of the model’s indicator function because the gradient of the indicator function is a vector field that is zero almost everywhere, except at points near the surface, where it is equal to the inward surface normal(Figure 2.10(c)).

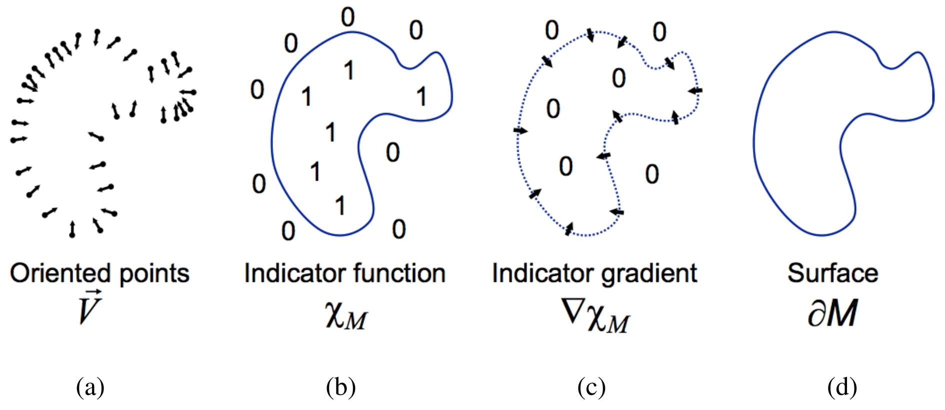


Figure 2.10: Poisson reconstruction in 2D from [35].

Figure 2.10 illustrates this relationship.

The input data Y is a set of samples $y \in Y$, each consisting of a point $y.\mathbf{x}$ and an inward-facing normal $y.\mathbf{N}$ assumed to lie on or near the surface ∂M of an unknown model M. The goal of their method is to reconstruct a watertight, triangulated approximation of the surface by approximating the indicator function ψ of the model and finally extracting the isosurface.

In their work, Kazhdan et al. [35] describe the relation between the gradient of the indicator function and the integral of the surface normal field with the following lemma: Given a solid M with boundary ∂M, let ψ_M denote the indicator function of M, $\mathbf{N}_{\partial M}(\mathbf{x})$ be the inward surface normal at $\mathbf{x} \in \partial M$, $\tilde{F}(\mathbf{x}')$ be a smoothing filter, and $\tilde{F}_x(\mathbf{x}') = \tilde{F}(\mathbf{x}' - \mathbf{x})$ its translation to the point $\mathbf{x}$. The gradient of the smoothed indicator function is equal to the vector field obtained by smoothing the surface normal field:

$$\nabla(\psi_M * \tilde{F})(\mathbf{x}_0') = \int_{\partial M} \tilde{F}_x(\mathbf{x}_0')\mathbf{N}_{\partial M}(\mathbf{x})d\mathbf{x} \tag{2.37}$$

Because the surface geometry is not known, the surface integral cannot be evaluated. However, an approximation can be achieved making use of the information provided by the input set of oriented points. Using the point set Y to partition ∂M into distinct patches $\mathscr{P}_y \subset \partial M$, the integral over a path $\mathscr{P}_y$ can be integrated by the value at point sample $y.\mathbf{x}$, scaled by the area of the patch:

$$\nabla(\psi_M * \tilde{F})(\mathbf{x}'_0) \approx |\mathscr{P}_y| \tilde{F}_{y.\mathbf{x}}(\mathbf{x},)y.\mathbf{N} \equiv \mathbf{V}(\mathbf{x}') \tag{2.38}$$

Once the vector field **V** is formed, the next step is to solve for the function $\tilde{\psi}$ such that, $\nabla\tilde{\psi} = \mathbf{V}$. Since **V** is generally not integrable, an exact solution will often not exist. The best least-squares approximate solution is then found by applying the divergence operator to form the Poisson equation

$$\Delta\tilde{\psi} = \nabla \cdot \mathbf{V}. \tag{2.39}$$

Levin [43] extended use of the method for the approximation of irregular data known as the moving least squares (MLS) [77] for the use in surface reconstruction. Later Alexa et al. [3] introduced it to the computer graphics community. The surface was defined as the set of stationary points of an iterative projection operator: at each step a polynomial approximation of the local neighbourhood is performed from a local planar parametrization.

As Shen et al. [76] point out, this approach for reconstructing surfaces produces good results when dealing with smooth surfaces. However, it cannot reconstruct correctly surfaces with sharp features. In order to solve this problem Shen et al. [76] propose that instead of fitting trivariate polynomials to the data, the standard MLS be used o reconstruct tangential implicit planes prescribed at each input sample position.

Kolluri [38] propose an implicit MLS (IMLS) method that makes use of constant polynomials as the MLS basis. While this method yields a simple weighted average, it also suffers from expanding and contracting effects when a global optimization step is not applied [30].

To overcome this problem Öztireli et al. [69] borrowed techniques from robust statistics [32] and build upon the work by Kolluri [38] to derive a new MLS surface definition by formulating it as a local kernel regression minimization [83] using a robust objective function. This approach is known as the robust implicit moving least squares method (RIMLS).

Triangulation

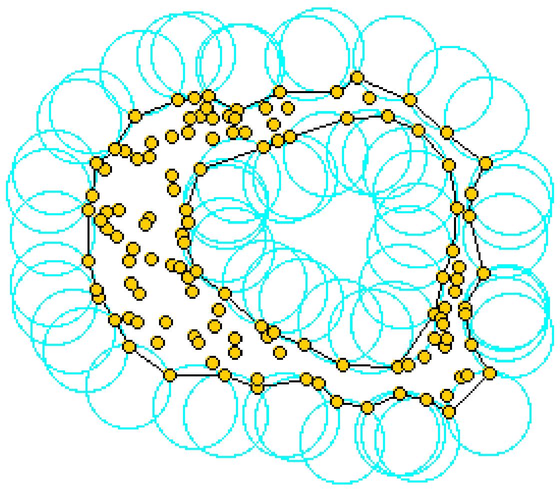


Figure 2.11: Alpha-shape representation in 2D from [15].

Alpha-shapes in the plane were introduced by Edelsbrunner et al. [20] and were later extended to higher dimensions [19] [21] as a geometric tool for reasoning about the "shape" of an unorganized set of points.

Fischer [26] describes alpha-shapes intuitively as a huge mass of ice-cream making up the space $\mathbb{R}$ and containing the points $\mathscr{A}$ as "hard" chocolate pieces. Using one of these sphere-formed ice-cream spoons we carve out all parts of the ice-cream block we can reach without bumping into chocolate pieces, thereby even carving out holes in the inside (e.g. parts not reachable by simply moving the spoon from the outside). We will eventually end up with an (not necessarily convex) object bounded by caps, arcs and points. If we now straighten all round faces to triangles and line segments, we have an intuitive description of what is called the alpha-shape of $\mathscr{A}$. Figure 2.11 illustrates an example of this process in 2D (where the ice-cream spoon in the analogy is simply a circle).

In Fischer's description α is the squared radius of the ice-scream spoon. Thus, as α gets closer to 0 in value the alpha-shape degenerates to the point set $\mathscr{A}$. On the other hand, as α gets closer to ∞ the ice-scream spoon becomes too large making it impossible to get to the ice-cream located on the inside of $\mathscr{A}$.

Formally an alpha-shape is a considered a subcomplex of the triangulation of $\mathscr{A}$. For any value of α, the alpha-shape includes all the simplices in the triangulation which have a with squared radius equal or smaller than α that does not include any points of $\mathscr{A}$. As described by Edelsbrunner et al. [21], the alpha-shape of $\mathscr{A}$ is a polytope that is neither necessarily convex nor connected and cavities may appear and join to form tunnels and holes.

2.4 Mathematical modelling of the baking process

Fan et al. [25] propose a model for the expansion of dough during the baking process. Their model is based on the growth of a single gas bubble composed of CO_2 and water vapour inside the dough. The dough is considered a viscous fluid and elasticity, an important aspect of dough, is not taken into account. CO_2 generation is not considered in their model, but rather the dough at the start of baking contains an N_c amount of gas cells. The volume expansion is driven by the difference between internal and atmospheric pressure given by the diffusion of CO_2 and moisture from the surrounding dough into the bubble. The following assumptions are also made:

- Gas cells contain only carbon dioxide and water vapour.
- Gas cells are considered closed and their number does not change through the baking process.
- The dough behaves rheologically as a power law fluid.
- The dough temperature is uniform throughout the dough's volume and it is dependant on time.

The relative volume of the dough is calculated in this model from the bubble radius as it increases with the rise of temperature and the specific number of gas cells. Figure 2.12 shows a spherical gas cell with cell radius $r_{cell}(T)$ and a bubble radius $r_b(T)$ inside the liquid dough. The equation of motion for the growth of the bubble for a power law fluid is taken from the work by Ramesh et al. [73]:

$$P_b - P_a = \rho_d \left(\ddot{r_b} \dot{r_b} + \frac{3}{2} \dot{r_b}^2 \right) + \frac{4\zeta_c (2\sqrt{3})^{(\zeta_f - 1)}}{\zeta_f} \left(\frac{\dot{r_b}}{r_b} \right)_f^{\zeta} + \frac{2\sigma}{r_b} \qquad (2.40)$$

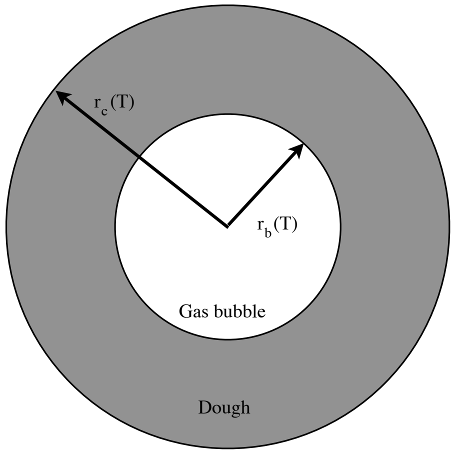


Figure 2.12: Fan et al. [25] representation of a gas cell in dough.

where $P_b - P_a$ is the difference between the bubble pressure and atmospheric pressure, ρ_d is the dough's density, r_b, $\dot{r_b}$ and $\ddot{r_b}$ are the bubble radius' dr_b/dT and $d^2 r_b/dT^2$ respectively. ζ_c is the fluid consistency index, ζ_f is the fluid flow index, σ is the surface tension. $P_b - P_\infty$ is the driving force behind the bubble growth in the model. The dough viscosity is high and therefore the inertia term is ignored turning Equation 2.40 into:

$$P_b - P_a = \frac{4\zeta_c (2\sqrt{3})^{(\zeta_f - 1)}}{\zeta_f} \left(\frac{\dot{r_b}}{r_b} \right)_f^{\zeta} + \frac{2\sigma}{r_b} \qquad (2.41)$$

The bubble pressure P_b is the sum of the CO_2 pressure P_c and water vapour pressure P_v

$$P_b = P_c + P_v \tag{2.42}$$

Water pressure P_v is determined by the water activity and the saturated vapour pressure P_v^0 at the temperature concerned. The changes in saturated vapour due to the temperature are obtained using the Clausius-Clapeyron equation [63]:

$$P_{v2}^0 = P_{v1}^0 \text{ exp } \left[-\frac{\Delta H_v}{R_g} \left(\frac{1}{T_2} - \frac{1}{T_1} \right) \right] \tag{2.43}$$

where P_{v1}^0 and P_{v2}^0 the saturated vapour pressures in equilibrium at temperatures T_1 and T_2 respectively, and ΔH_v the latent heat of water.

In their model CO$_2$ is assumed to behave as a perfect gas and to have a uniform concentration throughout the dough, although it varies with time. So the CO$_2$ pressure P_c can be obtained in the following way:

$$P_c = \frac{X_c^0 m_s + \frac{4}{3}\pi r_{b_0}^3 \rho_c^0}{\frac{m_s}{K_h} + \frac{4\pi r_b^3 M_c}{3R_g T}} \tag{2.44}$$

where X_c is the concentration of CO$_2$, m_s the mass of a cell, ρ_c^0 the initial CO$_2$ density, $symRg$ the gas constant, and M_c the molecular weight of CO$_2$.

The temperature is considered uniform throughout of the dough's volume and the dough is heated at a constant heating rate k_h.

$$\frac{dT}{dt} = k_h \tag{2.45}$$

The relative volume of the dough is tied to the radius of a gas bubble by the following equation:

$$V_r = \frac{4}{3}\pi r_b^3 N_c \rho_d + 1 \tag{2.46}$$

where N_c is the number of CO$_2$ gas cells in the dough.

The temperature dependent viscosity is taken from the work of Bloksma and Niewman [11] and takes the form of:

$$\mu_a = \zeta_{c0}\frac{\dot{r_b}^{(\zeta_f-1)}}{r_b}\exp\left[\frac{W_v}{R_g}\left(\frac{1}{T_2}-\frac{1}{T_1}\right)\right] \tag{2.47}$$

where W_v is the activation energy for viscous flow.

Marcotte et al. [54] propose a framework to simulate the baking of cake with the purpose of improving the quality of the product while saving energy in an industrial setting.

The assumptions made for the development of this program are the following:

- The temperature and moisture remain consistent and constant during all of the cake baking process.
- The cake takes the form of a cylinder and the dough is treated as a viscoelastic material.

The model consists of a surface area and a centre area due to the big differences in temperature and moisture content present within each area during baking.

The framework uses the Kelvin model to describe the volume expansion which is driven by the difference in pressure between the oven air and the gas mixture inside of the bubbles in the dough. The bubble pressure is assumed to depend on the CO_2 pressure and water vapour pressure similarly to Fan et al.'s [25] proposal; that is, as the sum of the vapour pressure P_v and CO_2 pressure P_c as shown in Equation 2.42.

In their work P_v also takes the form described by the Clausius-Clapeyron equation 2.43 [63]. P_c on the other hand is defined as

$$\begin{aligned} P_c &= \frac{nR_gT}{V} \\ P_c &= (n_{\mathrm{CO}_2 0} + \Delta n)\frac{R_gT}{V_{b0}+\Delta V_b} \qquad (2.48) \\ \Delta n_{\mathrm{CO}_2} &= 2D_L\pi r_b^2(X_c - C_b^*) \qquad (2.49) \end{aligned}$$

where n_{CO_2} is the number of CO$_2$ in moles, D_L the overall CO$_2$ transfer coefficient from dough to the bubble, and C_b^* is the CO$_2$ equilibrium concentration for the bubble.

The bubble volume expansion is derived from the balance of strain-stress equation:

$$\Delta P = E\epsilon + \mu\epsilon \tag{2.50}$$

$$T_{\mathrm{ret}} = \frac{\mu}{E} \tag{2.51}$$

Differentiating the two sides one gets:

$$\begin{aligned} t_{\mathrm{ret}}\varepsilon_S + \varepsilon_S &= 0 \\ \varepsilon_S &= \frac{\Delta P}{\mu_e} + \left(\varepsilon_{S0} - \frac{\Delta P}{\mu_e}\right) e^{-\Delta t/t_{\mathrm{ret}}} \\ \varepsilon_S &= \frac{\Delta H_e}{H_{ei}} + \varepsilon_{S0} \\ \Delta H_e &= H_{ei}\left[\frac{\Delta P}{\mu_e}\left(1 - e^{-\Delta t/t_{\mathrm{ret}}}\right) + \varepsilon_{S0}\left(e^{-\Delta t/t_{\mathrm{ret}}}\right)\right] \end{aligned} \tag{2.52}$$

where ΔP is the pressure difference, μ_e is the elastic coefficient, μ is the viscosity coefficient, ε_S is the strain, H_e is the height of the cake, ΔH_e is the increased height during time Δt and t_{ret} is the retardation time.

Initially the surface and centre areas are considered uniform. The initial temperature and moisture in the oven and inside the dough are constant and location independent. Also, the initial CO$_2$ concentration is constant on the surface and centre areas. A diagram for Marcotte's algorithm can be found in Figure 2.13.

Lostie et al. [49] define baking as a two stage process composed of a "heating up" stage and a "crust and crumb" stage. Their claim is that the transition from the first period to the second one can be easily observed on experimental baking curves, taking place when the internal temperature and water content curves level off and the curves for the drying rate and volume expansion pass through the maximum.

Their proposal is that of a vaporization front model for the "crust and crumb" stage where the crumb area of the baked good participates in the heat and water vapour transfer affecting the temperature and total pressure and ultimately having an effect on the final shape of the volume.

There are several assumptions made in their model:

- The dough is composed of two layers a dry crust and a wet deformable crumb.
- The heat and mass transfer are unidirectional.
- The gas phase moves through the crust by permeation under a total pressure gradient according to Darcy's law [16]. Furthermore, the gas phase obeys the perfect gas law.
- Heat is transferred through the crust by conduction according to Fourier's law.
- The heat and mass transfer in the mass is ignored causing the dough to have uniform temperature, water content and pressure.
- The crumb is considered as a viscous compressible medium.
- Volume expansion is is given by the vaporization of liquid water and the local deformation rate is proportional to the local overpressure.
- A local thermodynamic equilibrium exists.
- The dough is opaque to infrared radiation.

The vaporization front is located between the crust and crumb layers. The location of the vaporization front will thus move over time as the crust advances during the baking period. The heat arriving at the interface splits into two parts: the first part evaporates water at the interface while the second part serves to raise the crumb's temperature and to evaporate some water within the crumb.

Since the usual continuity equation of heat and mass fluxes cannot be written because there are no state variables gradients in the crumb, Lostie et al. [49] introduce an empirical coefficient that controls the velocity of the interface. This empirical coefficient G_p called the gas phase partition coefficient is considered to be directly generated by the advancing crust and the remainder $(1 - G_p)$ is generated within the crumb. G_p is defined by the following gas phase mass balance in the crumb:

$$\begin{aligned} \frac{dm_g^{wet}}{dt} &= -(1-G_p)\dot{m}_g - \frac{dm_l^{wet}}{dt} - \frac{m_l^{wet}}{m_s^{wet}}\rho_s^{dry}\frac{de_t^{dry}}{dt}C_a \\ &\quad - \frac{m_g^{wet}}{m_s^{wet}}\rho_s^{dry}\frac{de_t^{dry}}{dt}C_a \end{aligned} \tag{2.53}$$

where m represents the mass, G_p the gas phase partition coefficient, $\dot{m}$ the mass flow rate, ρ the density, e the thickness, e_t the thickness, and C_a the cake cross section surface area. The subscripts g, l, and s defines weather the variable belongs to the gas phase, liquid water, and dry matter, respectively. The superscripts wet and dry indicate that the variable is part of the crumb and crust, respectively.

The heat flow rate is given by :

$$\dot{Q} = -\Delta H_{vap}\frac{dm_l^{wet}}{dt} + (m_l^{wet}C_{heatl} + m_s^{wet}C_{heats})\frac{dT^{wet}}{dt} \tag{2.54}$$

where ΔH_{vap} is the water vaporization enthalpy, C_{heat} the specific heat capacity, and T the temperature. In Equation 2.54 the first term is the heat flow rate necessary for water vaporization throughout the cake and the second term is the thermal energy variation rate of the crumb.

The gas mass flow leaving the cake through the baking process is given by:

$$\dot{m}_g = -\frac{dm_g^{wet}}{dt} - \frac{dm_l^{wet}}{dt} - \rho_g^{wet}\epsilon_p^{dry}\frac{de^{dry}}{dt}C_a \tag{2.55}$$

where ϵ_p is the porosity.

The vapour mass variation rate within the crumb is given by :

$$\begin{aligned}\frac{d}{dt}(M_{f_v}^{wet} m_g^{wet}) &= -\frac{dm_l^{wet}}{dt} - \frac{m_l^{wet}}{m_s^{wet}}\rho_s^{dry}\frac{de_t^{dry}}{dt}C_a \\ &\quad - M_{f_v}^{wet}(1-G_p)\dot{m}_g - M_{f_v}^{wet}\frac{m_g^{wet}}{m_s^{wet}}\rho_s^{dry}\frac{de_t^{dry}}{dt}C_a \end{aligned} \tag{2.56}$$

where M_{f_v} is the mass fraction of water vapour.

The dry matter mass is given by:

$$m_s^{wet} = m_{s,ini}^{wet} - \rho_s(1-\epsilon_p^{dry})C_a e_t^{dry} \tag{2.57}$$

where the first term is the initial dry matter mass in the crumb and the second term in the equation is the removed dry matter mass from the crumb by the advancing crust.

The difference between the total gas pressure and the atmospheric pressure that will give way to the change in volume is defined in Equation 2.58. Since the behaviour of the crumb is assumed to be similar to that of a viscous compressible medium, the volume strain is proportional to the gas overpressure.

$$P^{wet} - P^{atm} = -\mu^{wet}\frac{e_t^{wet}}{m_s^{wet}}\frac{d}{dt}\left(\frac{m_s^{wet}}{e_t^{wet}}\right) \tag{2.58}$$

The crumb's volume is given by the sum of the volumes occupied by the gas, liquid water and the dry matter:

$$C_a e_t^{wet} = \frac{m_g^{wet}}{\rho_g^{wet}} + \frac{m_l^{wet}}{\rho_l} + \frac{m_s^{wet}}{\rho_s} \tag{2.59}$$

The gas and heat exchange rate between the cake and the oven air are given by Equations 2.60 and 2.61 respectively.

$$\dot{m}_g = \frac{K_g^{dry}}{\eta_{cg}}\frac{P^{wet} - P^{atm}}{e_t^{dry}}C_a \tag{2.60}$$

$$\dot{Q} = \left(\frac{e_t^{dry}}{\lambda_a^{dry}} + \frac{1}{k_{hext}}\right)^{-1}(T^{oven} - T^{wet})C_a \tag{2.61}$$

where K is the mass permeability,η_c the cinematic viscosity, λ_a the apparent thermal conductivity, and k_{hext} the external heat transfer coefficient.

The gas phase density ρ_g^{wet} and the vapour mass fraction ${M_f}_v^{wet}$ in the crumb are given by the perfect gas state equation and by the partial pressure additivity equation for air and vapour pressure.

Zhang et al. [91] [92] [90] propose a mathematical formulation and numerical implementation of the multiphase system in deformable porous media. To validate the model it is applied to the bread baking process. In this work the dough is considered a viscoelastic material with its mechanical properties defined by its temperature and its deformation is caused by transport phenomena, that is, the changes in temperature, moisture and pressure during the heating process.

Based on thermodynamic relations (i.e. the Gibbs phase rule [75]) when the temperature and moisture content are known, the pressure is also known.

$$P_v = P_v(T, M_w) \tag{2.62}$$

Zhang et al. et al. [91] [92] [90] base the governing equation for deformation on the principle of virtual work. Using a nonlinear mechanics convention in the Updated Lagrange format [87] the equation of deformation can be written as

$$\int_v (\mathbf{S}_K + \Delta\mathbf{S}_K)\delta\varepsilon_G dV = \int_v \mathbf{f}\delta\mathbf{u} dV \tag{2.63}$$

where the force term $\mathbf{f}$ is due to internal pressure and gravity, $\mathbf{u}$ is the displacement increment vector, ε_G the Green strain, and $\mathbf{S}_K$ the Kirchhoff stress that is defined as:

$$\Delta\mathbf{S}_K = \mathbf{D}_e(\Delta\varepsilon_G - \Delta\varepsilon_n). \tag{2.64}$$

The elasticity matrix $\mathbf{D}_e$ can be expressed as

$$\mathbf{D}_e = E\mathbf{D}_{e0}(\mathbf{v}) \tag{2.65}$$

where E is Young's modulus and $\mathbf{D}_{e0}$ is related to Poisson's ratio.

The non-elastic strain ε_n is defined as:

$$\Delta\varepsilon_n = \left(\mathbf{D}_{e(n)}\right)^{-1}\mathbf{S}_K\Delta t \tag{2.66}$$

In this model heat transfer occurs due to convection and radiation. In order to fully model the heat transfer due to radiation the framework would need to take into account the emissivities of bread, gas, oven surface as well as the computation of the geometric relations between bread and oven. To simplify the inclusion of the radiative effect an overall heat transfer coefficient is used.

The CO_2 generation rate is modelled as

$$R_c = R_{c0}\,\text{exp}\,\left(-\frac{T - T_{ref}}{\Delta T}\right)^2 \tag{2.67}$$

where R_c is the rate of CO_2 generation. The parameters of temperature at the reference point T_{ref} and temperature rate of change are chosen so that CO_2 generation occurs s mostly between $20°$ and $60°$C with a peak around $40°$C.

For the creation of this model experimental measurements of the rheological properties were made because the data available for temperature during baking below 60°C were considered inconsistent. The relaxation time is formulated as a function of the temperature [90]:

$$\lambda = 9\left(\frac{2}{\pi}\,\text{arctan}\left(\frac{T-65}{2}\right)\right) + 2 \tag{2.68}$$

Zhang et al.'s [91] [92] [90] numerical implementation uses the Galerkin Finite element method to solve the governing equations which use a triangular mesh to represent the dough's volume as shown in Figure 2.14. The solution procedure for Zhang's algorithm is shown on Figure 2.15.

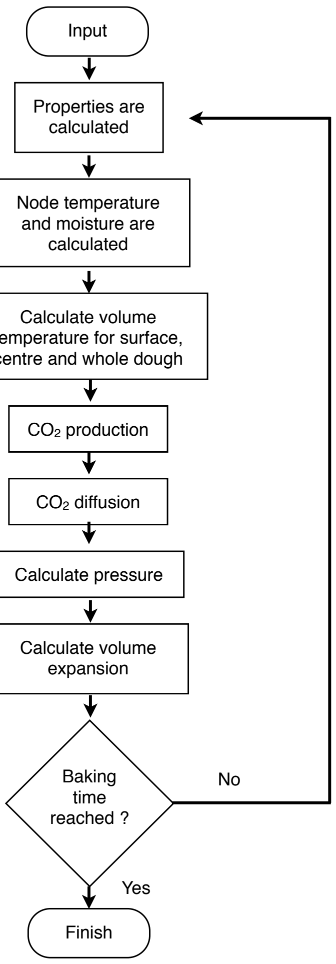

Input
Properties are calculated
Node temperature and moisture are calculated
Calculate volume temperature for surface, centre and whole dough
CO2 production
CO2 diffusion
Calculate pressure
Calculate volume expansion
Baking time reached ?
No
Yes
Finish

Figure 2.13: Marcotte et al. [54] algorithm for cake baking.

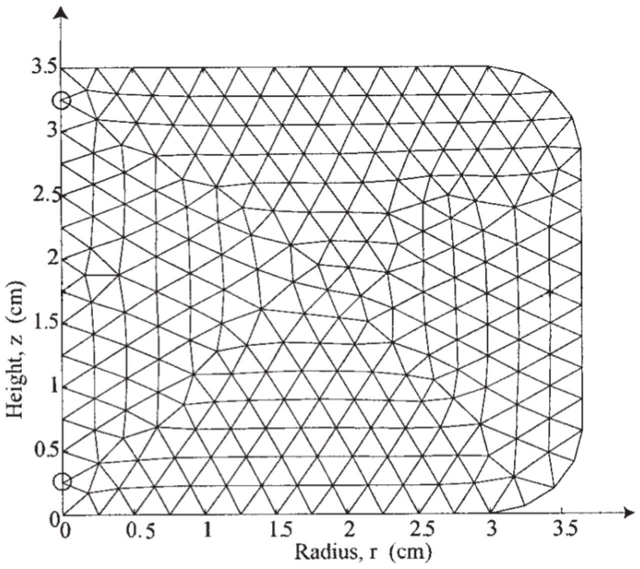


Figure 2.14: Initial mesh for Zhang's simulation from [90].

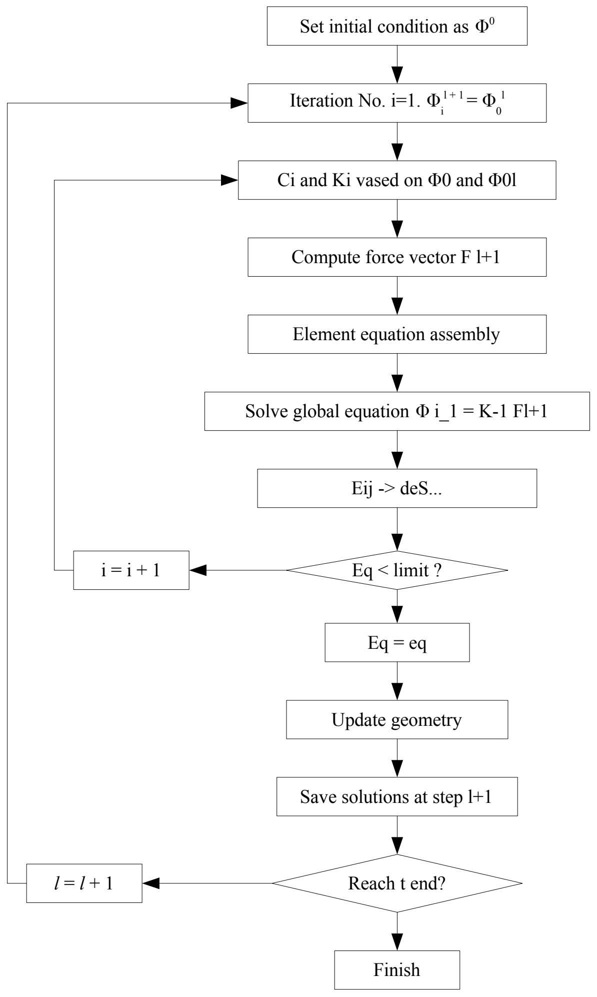

Set initial condition as Φ^0
Iteration No. i=1. $\Phi_i^{l+1} = \Phi_0^l$
Ci and Ki vased on Φ0 and Φ0l
Compute force vector F l+1
Element equation assembly
Solve global equation Φ i_1 = K-1 Fl+1
Eij -> deS...
Eq < limit ?
i = i + 1
Eq = eq
Update geometry
Save solutions at step l+1
Reach t end?
$l = l + 1$
Finish

Figure 2.15: Zhang et al.'s [91] [92] [90] algorithm for the simulation of the baking process.

Chapter 3

Smoothed particle hydrodynamics

3.1 SPH framework

As previously discussed in Chapter 2, SPH is a Lagrangian approach at solving the Navier-Stokes (NS) equation for fluid motion as expressed in Equation 3.1.

$$\frac{d\mathbf{v}}{dt} = -\frac{1}{\rho}\nabla P + \mu\nabla^2\mathbf{v} + \mathbf{f} \tag{3.1}$$

This section describes the proposed framework's implementation of the basic SPH formalism concepts, as well as the evaluation procedures for the properties of the simulated fluids.

3.1.1 Kernel function

The kernel function $W(\mathbf{x}_i - \mathbf{x}_j, h)$ is an important part of the SPH formulation. As explained in Section 2.2.1, this function obtains the weight value that is used to compute particle p_j's contribution to the property of particle i (when computing any property as explained in equation). The kernel function $W(\mathbf{x}_i - \mathbf{x}_j, h)$ is usually a function of the distance between two interacting particles $r_{ij} = |\mathbf{x}_i - \mathbf{x}_j|$, and for short, it can be expressed as $W(r_{ij}, h)$.

The kernel function can take any form as long as it complies with the requirements described on Section 2.2. Specialized kernel functions can be used for computing different properties when required. For example, in Müller's [64] framework for interactive applications he proposes the use of the kernel function W_{poly6} 3.2 for computing all properties in their SPH framework (excluding the pressure force and viscosity properties of the fluid, which will be dealt with shortly). The kernel function W_{poly6} takes the form of:

$$W_{poly6}(r_{ij},h) = \frac{315}{64\pi h^9}\begin{cases}(h^2 - r_{ij}^2)^3, & 0 \leq r_{ij} \leq h \\ 0, & \text{otherwise}\end{cases} \tag{3.2}$$

with its gradient taken the form of:

$$\nabla W_{poly6}(r_{ij},h) = \frac{315}{64\pi h^9}\begin{cases}-6r_{ij}(h^2 - r_{ij}^2)^2, & 0 \leq r_{ij} \leq h \\ 0, & \text{otherwise}\end{cases} \tag{3.3}$$

and its Laplacian:

$$\nabla^2 W_{poly6}(r_{ij},h) = \frac{315}{64\pi h^9}\begin{cases}6(h^2 - r_{ij}^2)(7r_{ij}^2 - 3h^2), & 0 \leq r_{ij} \leq h \\ 0, & \text{otherwise}\end{cases} \tag{3.4}$$

As mentioned before, two cases that need special care are the pressure force and viscosity properties of the fluid. A drawback of using Equation 3.2 to compute the pressure forces is that the particles start to form cluster as they grow closer to each other. This takes place because the gradient of the kernel approaches zero at the centre. To solve this problem, Desbrun and Gascuel [17] propose the use of a spiky kernel with a non-vanishing gradient near the centre. This spiky kernel is also used by Müller [66] and is defined as:

$$W_{spiky}(r_{ij},h) = \frac{15}{\pi h^6}\begin{cases}(h - r_{ij})^3, & 0 \leq r_{ij} \leq h \\ 0, & \text{otherwise}\end{cases} \tag{3.5}$$

and its gradient—which is used to compute the pressure forces—takes the form of:

$$\nabla_i W_{spiky}(r_{ij},h) = \mathbf{N}_{ij}\frac{15}{\pi h^6}\begin{cases}-3(h - r_{ij})^2, & 0 \leq r_{ij} \leq h \\ 0, & \text{otherwise}\end{cases} \tag{3.6}$$

It can be seen from Equation 3.6 that, as the distance between particles i and j decreases, the magnitude of the gradient increases to approach a constant. In other words, as the particles get closer, the repulsive force increases.

The second case that needs special care is the computation of the viscosity forces. Viscosity is a phenomenon caused by the internal friction of the fluid. Thus, it decreases the relative velocity of particles. However, if a standard kernel is used for viscosity the simulation may become unstable because as particles get too close to each other, the Laplacian of the smoothed velocity field (on which the viscosity forces depend) can get negative resulting in an increase in the particles' relative velocities. To avoid this problem Müller [66] uses the kernel function $W_{viscosity}(r_{ij}, h)$ as defined by Equation 3.7:

$$W_{viscosity}(r_{ij}, h) = \frac{15}{2\pi h^3} \begin{cases} -\frac{r^3}{2h^3} + \frac{r_{ij}^2}{h^2} + \frac{h}{2r_{ij}} - 1, & 0 \le r_{ij} \le h \\ 0, & \text{otherwise} \end{cases} \tag{3.7}$$

And its laplacian takes the form of:

$$\nabla^2 W(r_{ij}, h) = \frac{45}{\pi h^6}(h - r_{ij}) \tag{3.8}$$

The proposed framework makes use of two kernel functions. The viscosity is calculated using Müller et al.'s [65] proposed viscosity kernel $W_{viscosity}(r_{ij}, h)$, and the rest of the attributes are calculated using a function known as the spline kernel. The spline kernel $W_{spline}(r_{ij}, h)$ was initially proposed by Monaghan [60] and it takes the form of:

$$W_{spline}(r_{ij}, h) = \frac{1}{\pi h^3} \begin{cases} 1 - 1.5q^2 + 0.75q^3, & 0 \le q < 1 \\ 0.25(2-q)^3, & 1 \le q \le 2 \\ 0, & \text{otherwise} \end{cases} \tag{3.9}$$

where

$$q = \frac{2r_{ij}}{h} \tag{3.10}$$

With its gradient and Laplacian described in Equation 3.11 and Equation 3.12, respectively.

$$\nabla W_{spline}(r_{ij},h) = \begin{cases} -3q + 2.25q^2, & 0 \le q < 1 \\ -0.75(2-q)^2, & 1 \le q \le 2 \\ 0, & \text{otherwise} \end{cases} \tag{3.11}$$

$$\nabla^2 W_{spline}(r_{ij},h) = \begin{cases} -3 + 4.5q, & 0 \le q < 1 \\ 1.5 * (2-1), & 1 \le q \le 2 \\ 0, & \text{otherwise} \end{cases} \tag{3.12}$$

The spline kernel $W_{spline}(r_{ij}, h)$ is used in the present work because it complies with all the requirements presented in Section 2.2 (namely unity, compact support, positivity, decay, delta function property, symmetric property, and smoothness), and it produces stable animations with the performed tests.

3.1.2 Neighbour search

Due to the nature of the SPH framework, locating the set of particles that lie within a distance of a point in 3D space is an operation that is performed frequently. This problem is known as *neighbour search problem*, and for a small number of particles, a brute force method would be preferred. However, a brute force method has a complexity of $O(N_t^2)$ where N_t is the total number of particles, and a SPH framework requires a large amount of particles to achieve realistic looking animations. Since the neighbour search operation is performed at every time step it becomes a good candidate for optimization.

A commonly used method to improve the speed of the neighbour search operation is the use of a grid-based data structure that divides the space occupied by the fluid's particles in cubic cells, each with a volume of h^3 where h is the simulation's smoothing length. All of the particles are registered to the cell where they are located. To obtain the particles that lie within the neighbourhood of a point in space a search is performed on 27 cells: the cell that contains that 3D point and the 26 surrounding cells. This approach has a complexity of $O(N_s N_t)$; m is the number of particles in each search and $N_s \geq N_a$ where N_a is the average number of neighbours of the 3D point, but N_s will still be much smaller than N_t.

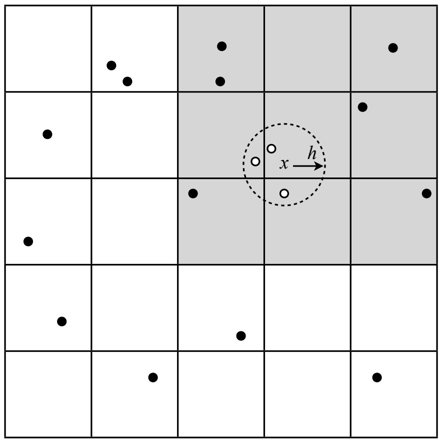


Figure 3.1: A 2D example of the neighbour search operation using a grid-based data structure.

Figure 3.1 illustrates an example of a 2D grid-based structure being used to find the neighbours of the point marked as x. In this 2D example, the search is performed on the 9 shaded cells, each particle's position is tested to see if it falls within the range of h. In this instance, only the three white particles are part of the neighbourhood.

There are other data structures that perform better than the grid-based data structure for the neighbour search operation. One such structure is the kd-tree [7] [27]. The kd-tree is a generalization of octrees, were k represents the number of dimensions being subdivided. The difference between the kd-tree and octree data structures is that where the octree subdivides the space along three dimensions, the kd-tree divides the space along one dimension at a time.

To recursively construct a canonical kd-tree, the required parameters are the complete set of points S that will make up the tree, and an integer d that will represent the depth of the root of the subtree that the recursive call constructs. On the first call the parameter d will be set to zero.

In 3D, the space will generally be divided on the first recursion along the x axis by finding the median of the points x component, followed by the y component, then the z component, finally returning to the x component until the tree is finished. This method results in a balanced kd-tree in which each leaf node is about the same distance from the node.

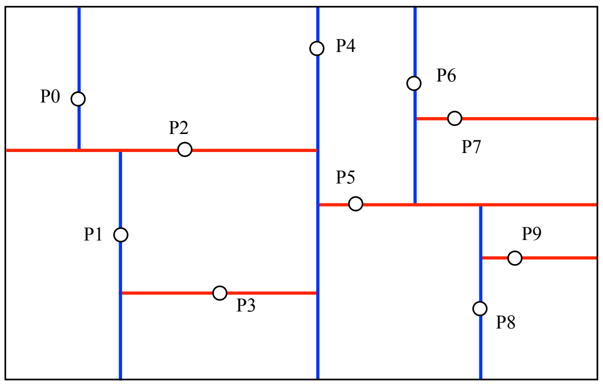
Figure 3.2(a) shows a 2D illustration of a spatial decomposition performed using the kd-tree, and Figure 3.2(b) shows its corresponding tree layout.

Algorithm 1: ConstructKDTree

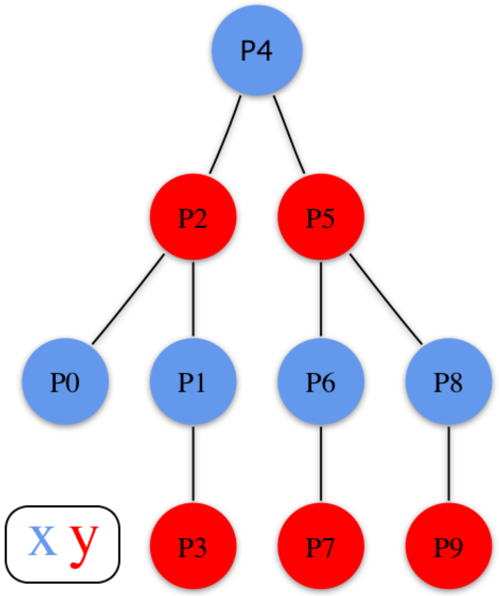
```
Input: Set of points $S$ in $\mathbb{R}^d$, current depth $D_c$
Output: Root node of new $k_h$d-tree
1  if $S$ is empty then
2      return null;
3  else
       // Select axis based on depth so that axis
          cycles through all valid values
4      $splitAxis \leftarrow D_c$ mod $d$;
       // Sort point list and choose a pivot element
5      $split \leftarrow ChooseSplitPoint(S, splitAxis)$;
       // Create node and construct subtrees
6      $node.location \leftarrow split$;
       // child hold the left and right partitions.  0
          = near, 1 = far
7      $node.child[0] \leftarrow ConstructKDTree($points in $S$ before $split, D_c + 1)$;
8      $node.child[1] \leftarrow ConstructKDTree($points in $S$ after $split, D_c + 1)$;
9      return node;
10 end
```

Finding the median is an expensive operation that can potentially make a balanced kd-tree a sub-optimal solution; however, using the median to divide the space is optional. One way to avoid the overhead of sorting all the input points to find the median when splitting the space in half is to sort a fixed number of points randomly chosen from the input set, and find the median from this subset. Another approach is to simply use a random point from the input set; this method offers no guarantee of resulting in a semi balanced tree but in practice it can prove an effective solution.

Ericson [24] describes an elegant solution for visiting all of the nodes in a kd-tree that are overlapped by a sphere. An adaptation of this solution implemented



(a) Spatial decomposition



(b) *k*d-tree layout.

Figure 3.2: A 2D *k*d-tree.

in the proposed framework is presented in Algorithm 2 to perform a range search operation that finds the neighbours of point **x**. As Ericson [24] mentions, an interesting problem when finding the points overlapped by a sphere is to correctly reject the subtree volumes that do not contain a section of the sphere. One solution is to maintain the point inside the volume closest to the sphere centre during traversal. In Algorithm 2 the variable $volNearPt$ is initially set to the query point p. As the traversal recurses the far side of the splitting plane, the point is clamped to the splitting plane to determine the closest point on the volume boundary. With this, the distance between $volNearPt$ and the query point **x** can be used to cull subtrees.

Algorithm 2: RangeSearchKDTree

```
Input: $node$ of $k_h$d-tree, point $\mathbf{x}$ in 3D space, squared of range $r$, point
       $volNearPt$, empty neighbourhood $N$
Output: Neighbourhood $N$ around $\mathbf{x}$
1  if node is null then
2      return;
3  end
4  if $sqrdDistance(node.location, \boldsymbol{x}) < r$ then
5      $N.add(node)$;
6  end
   // Figure out which child to recurse into first
7  $first = \mathbf{x}[node.splitAxis] > node.location[node.splitAxis]$
   // Always recurse into the subtree point x is in
8  $RangeSearchKDTree(node.child[first], \mathbf{x}, r, volNearPt, N)$;
   // Update (by clamping) nearest point on volume
      when traversing far side.  Keep old value on the
      local stack so it can be restored later.
9  $oldValue \leftarrow volNearPt[node.splitAxis]$;
10 $volNearPt[node.splitAxis] \leftarrow node.location[node.splitAxis]$;
11 if $sqrdDistance(volNearPt, \boldsymbol{x}) < r$ then
12     $RangeSearchKDTree(node.child[first \text{ xor } 1], \mathbf{x}, r, volNearPt, N)$;
13 end
14 $volNearPt[node.splitAxis] \leftarrow oldValue$;
```

A range search query in a balanced kd-tree takes $O(N_t^{1-\frac{1}{k}} + N)$ time, where N_t is the total number of points, N the number of the reported points, and k the dimension of the kd-tree. When dealing with a large number of particles, the effi-

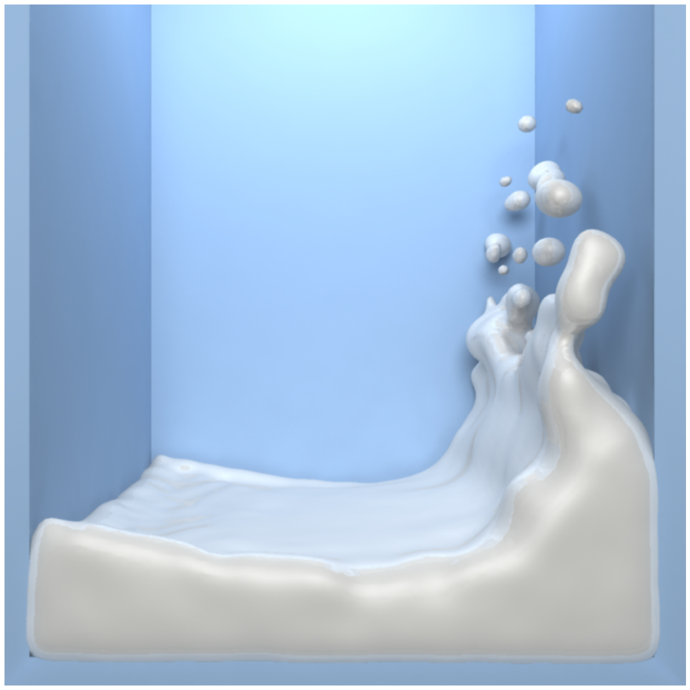


Figure 3.3: The classical dam break test.

cient search algorithm of the kd-tree outweighs the costly creation time making the kd-tree a better choice compared to the grid search structure.

Several versions of the classical dam break animation were made to compare the difference in running time: versions where the input data is sorted are compared against versions where the data is unsorted. The same scene setting was used for all the animations, each of which has a duration of 5 seconds and the fluid particle count was increased for each animation.

Particle count	Sorted data	Unsorted data
$2,920$	6 minutes	7 minutes
$10,180$	41 minutes	42 minutes
$22,890$	1 hour 52 minutes	1 hour 51 minutes
$102,975$	9h 44 minutes	10 hours 1 minute

Table 3.1: Animation running time using kd-tree.

Table 3.1 shows the running times obtained using the proposed framework of dam break simulations with an increasing number of particles for each test (the animations were created in a computer with 2 dual-core 2.21 Ghz AMD Opteron processors with 4 GB of ram). By comparing the results it is possible to see that there is an insignificant difference in running time for creating an animation when sorting the data during the building process of the kd-tree and simply selecting the point stored in the middle of the input data vector. It can be argued that the reason behind these results is that, due to their sequential creation, the input data is sorted when the kd-tree is built. However, as the animation evolves, the positions of the particles change greatly, modifying as well each of the particle's neighbourhood. To test if this was the case, a dam break scene animation of 45 seconds was created with the fluid composed of $2,920$ particles. As the animation evolves changing the particles' positions drastically, so do the particles' set of neighbouring particles; yet, the results are similar with the sorted and unsorted data versions taking each 55 minutes to complete the animation.

3.1.3 Density

The SPH formalism usually computes the density of the particles using the density summation as described on Equation 3.13.

$$\rho_i = \sum_j^N m_j W(\mathbf{x}_i - \mathbf{x}_j, h) \tag{3.13}$$

However, this method has a drawback in that the particles located at the surface of the fluid will have an erroneously calculated lower density than their counter parts inside the fluid, because the outlying particles have fewer neighbours even when the particles are equally spaced apart. To get around this problem Monaghan [61] proposes solving the continuity equation which under the SPH formalism takes the form of:

$$\frac{d\rho_i}{dt} = \sum_j^N m_j (\mathbf{v}_i - \mathbf{v}_i) \nabla W(\mathbf{x}_i - \mathbf{x}_j, h) \tag{3.14}$$

This method requires the initial density ρ_0 of the particles to be initialized to some value, and the density changes are given only by the relative motion of the particles.

The work presented in this thesis uses Equation 3.13 to compute the density of the particles because this method achieves greater stability than the method proposed by Monaghan [61].

3.1.4 Pressure

The purpose of the pressure term of the NS equation of momentum, as shown in Equation 3.1, is to enforce the fluid's incompressibility; however, as Liu and Liu [45] explain, the actual equation of state of the fluid leads to prohibitive time states that are extremely small. Also, although it is possible to include the constraint of the constant density into the SPH formulations, the resultant equations are too cumbersome.

The concept of artificial compressibility comes from the fact that a theoretical incompressible flow is practically compressible. It is possible then to use a quasi-incompressible equation of state to model the incompressible flow.

One way of calculating each particle's pressure is by using the ideal gas law equation

$$P_i = B\rho \tag{3.15}$$

where B is a gas constant that depends on temperature.

Another popular quasi-incompressible equation is suggested by Desbrun et al. [17]:

$$P_i = B(\rho_i - \rho_0) \tag{3.16}$$

where ρ_0 is the rest density. The offset introduced to Equation 3.15 by Desbrun et al. [17] has no mathematical effect on pressure forces since pressure forces, depend on the gradient of the pressure field. However, the offset does make the simulation numerically more stable by influencing the gradient of a field smoothed by SPH.

Premžoe et al. [72] make use of the particle-based formulation developed for computational fluid dynamics known as Moving-Particle Semi-Implicit to animate surface flows. In their work, the pressure term is evaluated in a process to compute the Poisson equation for pressure. Mao and Yang [52] extend Premžoe et al.'s [72] work to be used under the SPH formalism.

Monaghan [61] proposes the use of Tait's Equation [6] to model free surface flows:

$$P_i = B\left(\left(\frac{\rho}{\rho_0}\right)^{\gamma} - 1\right) \tag{3.17}$$

The framework presented in this thesis uses Tait's Equation [6] because it enforces low density variations and is efficient to compute while maintaining the simulation stable. For Tait's Equation [6] the constant B is set empirically and γ takes the value of 7 following the work of Becker et al. [6].

After evaluating the pressure at all particles using Equation 3.17, the pressure term in the NS Equation 3.1 is evaluated as:

$$\frac{1}{\rho_i}\nabla P_i = \sum_{j=1}^{N} m_j \left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2}\right) \nabla W(\mathbf{x}_i - \mathbf{x}_j, h) \tag{3.18}$$

3.1.5 Viscosity

Informally, viscosity can be regarded as a measure the resistance of a material to change in form. A fluid with lower viscosity, such as water, can be considered thin. In contrast, a thick fluid such as honey has high viscosity making it harder for it to flow.

The framework presented in this thesis can produce fluid animations where the viscosity of the fluid evolves locally over time, based on the particle's individual temperature. To achieve this, the viscosity coefficient μ is stored independently at each particle.

The viscosity force at particle p_i is then computed under the SPH formalism following the work by Müller et al. [66] as

$$\mu \nabla^2 \mathbf{v} = \sum_{j=1}^{N} \frac{\mu_i + \mu_j}{2} m_j \frac{\mathbf{v}_j - \mathbf{v}_i}{\rho_j} \nabla^2 W(\mathbf{x}_{ij}, h_{ij}) \tag{3.19}$$

meaning that the individual viscosity coefficients are averaged between interacting particles achieving symmetry in their interaction.

3.1.6 Stress tensor

The mechanical properties of a Newtonian fluid are characterized by a single function of temperature: the viscosity, a measure of resistance of a fluid that is being deformed by either shear or tensile stress. The flow properties of a non-Newtonian fluids, on the other hand, are not only described by its viscosity. Non-Newtonian fluids are very common, examples of them appear in our daily lives: mayonnaise, egg whites, peanut butter, etc. Human bodily fluids like blood and mucus also fall under the non-Newtonian category as well as geological phenomena such as lava, mud flows, and glacier mechanics.

In CFD a fluid is considered non-Newtonian if the stress tensor cannot be expressed as a linear, isotropic function of the velocity gradient; otherwise, the fluid is considered Newtonian [68]. This stress tensor defines the internal stress developed by a fluid as a result of being deformed.

The framework presented in this thesis extends the work by
Mao and Yang [51] [52] on non-Newtonian fluids. As mentioned on Chapter 2, in their work, the constitutive equation used to integrate the stress tensor rate is based on a non-linear Maxwell model as defined on Equation 3.20:

$$\frac{d\mathbf{S}}{dt} = \mathbf{R} + \mu_e D' - \frac{1}{\lambda}\mathbf{S} \tag{3.20}$$

where **S** is the stress tensor, μ_e the elasticity constant, and λ the relaxation time. The rotational tensor **R** is expressed as:

$$\mathbf{R} = \frac{1}{2}(\mathbf{S} \cdot \omega - \omega \cdot \mathbf{S}) \tag{3.21}$$

where ω is the fluid angular velocity expressed as the matrix:

$$\omega = \nabla \mathbf{v} - (\nabla \mathbf{v})^T \tag{3.22}$$

and each matrix element $\omega^{\alpha_e \beta_e}$ is

$$\omega^{\alpha_e \beta_e} = \frac{\partial \mathbf{v}_e^{\beta}}{\partial \mathbf{x}_e^{\alpha}} - \frac{\partial \mathbf{v}_e^{\alpha}}{\partial \mathbf{x}_e^{\beta}} \tag{3.23}$$

$\mathbf{D}'$ is a traceless stress tensor:

$$\mathbf{D}' = \frac{1}{2}\mathbf{D} - \frac{tr(\mathbf{D})}{c} I \tag{3.24}$$

where $\mathbf{D}$ is the velocity gradient, $tr(\mathbf{D})$ the trace of matrix $\mathbf{D}$, and I is the identity matrix. $\mathbf{D}$ is computed as:

$$\mathbf{D} = \nabla \mathbf{v} + \nabla \mathbf{v}^T \tag{3.25}$$

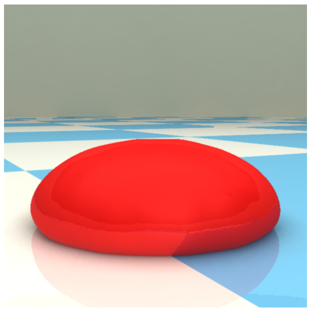
where each matrix element $\mathbf{D}^{\alpha_e \beta_e}$ is

$$\mathbf{D}^{\alpha_e \beta_e} = \frac{\partial \mathbf{v}_e^{\beta}}{\partial \mathbf{x}^{\alpha}} + \frac{\partial \mathbf{v}_e^{\alpha}}{\partial \mathbf{x}_e^{\beta}} \tag{3.26}$$

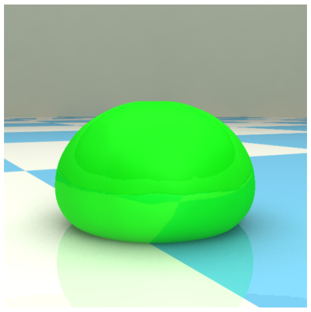
The Greek indices α and β in Equation 3.26 denote 3D spatial coordinates. Under the SPH framework, the partial velocity derivative at position $\mathbf{x}$ is evaluated as:

$$\frac{\partial \mathbf{v}_e^{\alpha}(\mathbf{x})}{\partial \mathbf{x}_e^{\beta}} = \sum_{j=1}^{N} \frac{m_j}{\rho_j} (\mathbf{v}_j^{\alpha_e} - \mathbf{v}^{\alpha_e}) \frac{\partial W(\mathbf{x} - \mathbf{x}_j, h)}{\partial \mathbf{x}^{\beta_e}} \tag{3.27}$$

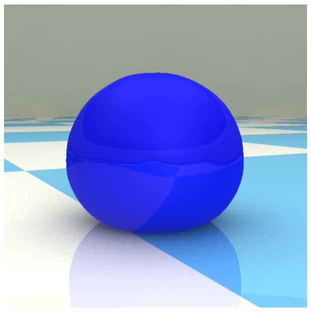
The relaxation time λ that appears on Equation 3.20 is a characteristic of non-Newtonian fluids that defines a memory the fluids have to retain their shape. In essence, the higher the relaxation time, the stronger the non-Newtonian fluid will



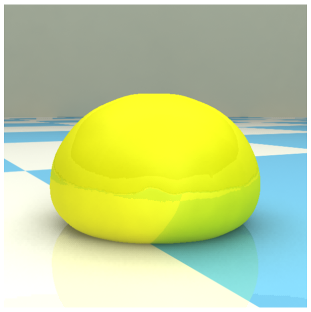
(a) $\lambda = 0.1$, $\mu_e = 1000$



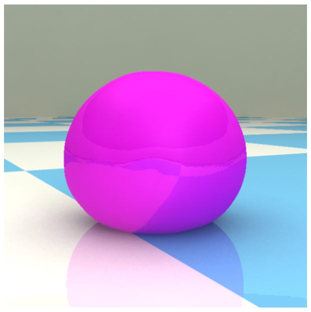
(b) $\lambda = 0.1$, $\mu_e = 10000$



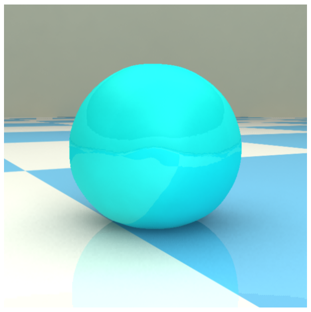
(c) $\lambda = 0.1$, $\mu_e = 100000$



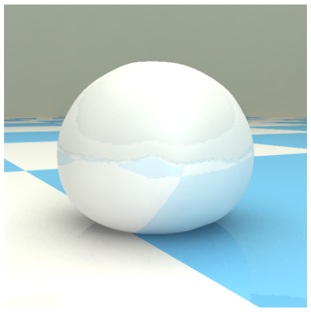
(d) $\lambda = 1$, $\mu_e = 1000$



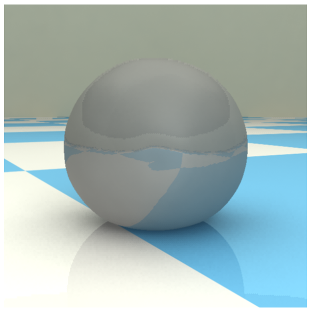
(e) $\lambda = 1$, $\mu_e = 10000$



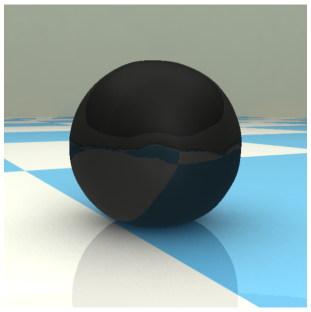
(f) $\lambda = 1$, $\mu_e = 100000$



(g) $\lambda = 10$, $\mu_e = 1000$



(h) $\lambda = 10$, $\mu_e = 10000$



(i) $\lambda = 10$, $\mu_e = 100000$

Figure 3.4: Stress affected by μ_e and λ through animation.

try to keep its original shape. The elasticity constant μ_e defines how the fluid resists to deformation. The higher μ_e is the stronger it will resist.

A more detailed description of how the stress tensor is computed can be found in Reference [51].

Figure 3.4 shows the variations that can be obtained by manipulation the μ_e and λ values of Equation 3.20 while Figure 3.5 shows a side-by-side comparison.

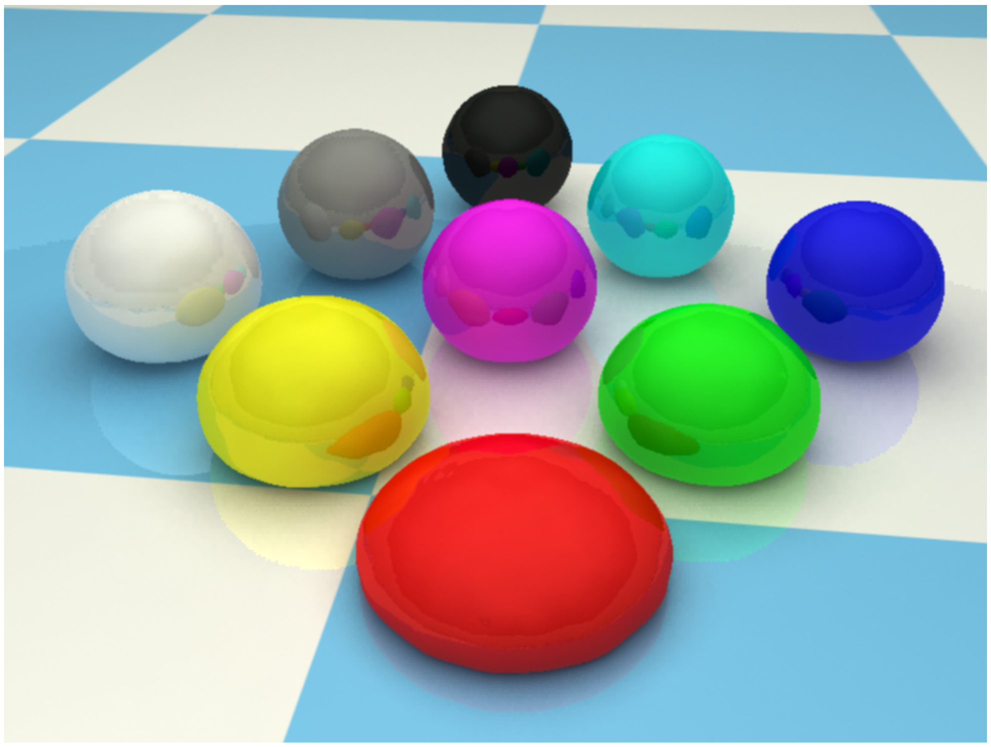


Figure 3.5: Side by side comparison of the effects of the stress tensor.

3.1.7 Surface tension

Surface tension is an effect within the surface boundary of a liquid that causes the boundary to behave as an elastic sheet; it is this force that tends to make the liquid surface smooth. It is because of surface tension that a liquid can resist an external force. As Bridson [12] explains, surface tension is caused because the fluid particles are more attracted to other fluid particles of the same type than to other particles of another type, such as air particles. Thus, the fluid's particles that are near to the surface tend to be pulled in towards the rest of the fluid's particles. This can be thought of like the fluid minimizing its exposed surface area.

Müller et al. [64] model surface tension using the colour parameter. The colour parameter is a property that takes the value of 0 everywhere except at each particle, where it takes the value of 1. Following the SPH formalism the colour property can be evaluated as follows:

$$C_i = \sum_j^N \frac{m_j}{\rho_j} C_j W(r_{ij}, h) \tag{3.28}$$

The surface particles and their normals can be found using the gradient of the colour field:

$$\mathbf{N}_i = \nabla C_i = \sum_j^N \frac{m_j}{\rho_j} C_j \nabla W(r_{ij}, h) \tag{3.29}$$

Particle i can be considered to be on the fluid's surface if the magnitude of the gradient is larger than a certain threshold value.

Finally, the surface tension can be computed as:

$$\mathbf{f}_{tension} = -\frac{\sigma}{\rho_i} \nabla^2 C_i \frac{\mathbf{N}_i}{|\mathbf{N}_i|} \tag{3.30}$$

where σ is the surface tension coefficient between the two fluids. It is worth noting that if the magnitude of $\mathbf{N}_i$ is small numerical instabilities can occur. To avoid this problem the surface tension is only calculated if $|\mathbf{N}_i|$ exceeds a certain threshold value.

3.1.8 Surface particles

Keeping a record of the surface particles during simulation is important because this record can be used either to reproduce different phenomena, or for rendering the surface of the fluid. As described in Section 3.1.7 the gradient of the colour field can be used to detect which particles are at the surface of the fluid. However, a disadvantage of using the colour field to detect the surface particles is that a threshold value has to be estimated by the user to differentiate between the surface and inner particles. A different approach that does not require trial and error is proposed by Mao and Yang [51] using the concept of local convex hull.

The definition of convex hull of a set of points in 2D is the smallest convex polygon that encloses all the points in the set; an often used intuitive example is visualizing the convex hull as a rubber band that tightly surrounds an object assuming its contour. Extending this definition to 3D, the convex hull is defined as the smallest closed triangular mesh that encompasses the points in 3D.

A local convex hull is defined as a convex hull built using a particle and its surrounding neighbours as data set to generate animations of immiscible fluids. This convex hull is built with the purpose of detecting the fluid's particles that lie at the surface of the fluid, as well as for calculating its normals.

The proposed framework also uses Mao and Yang's [51] [52] [53] concept of local convex hull to detect the surface particles. The convex hull is generated using both the particle of interest, and its neighbouring particles; the particle will be considered as being on the surface if the particle in question is closer than some threshold value to one of the vertex or facets belonging to the local convex hull.

Figure 3.6 illustrates an example of the local convex hull in 2D. Here, the particle labeled A would be considered to be on the surface of the fluid. On the other hand, particle B would be considered to be on the inside of the fluid.

The generation of the convex hull is implemented using the QuickHull algorithm [5].

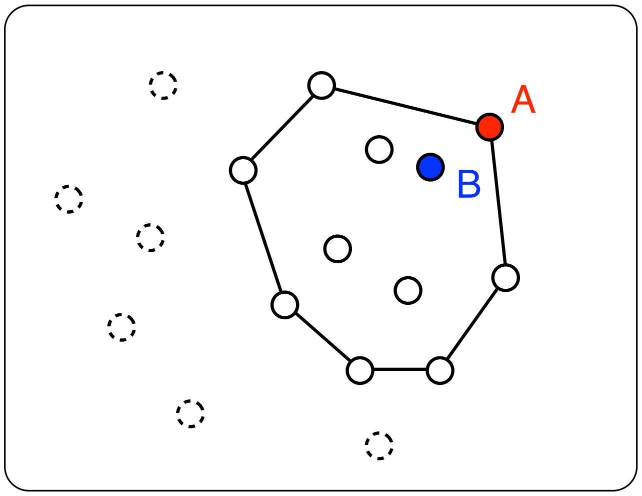


Figure 3.6: Local convex hull.

3.1.9 Collision detection and response

In order to detect collisions between solids and fluid particles, Karabassi et al. [34] propose taking into account the particle's volume and defining the solids as triangular meshes, which reduces the problem to detecting the intersections between cylinders with spherical caps (the particles' trajectory), and triangles.

In Solenthaler et al.'s [79] work, no explicit collision handling methods are applied; instead, the solids themselves are defined using the SPH framework, and the collisions are handled using a high gas constant B in the pressure term of the NS equation. They clarify that this approach does not guarantee an effective collision response when high forces are applied, but it works on most collision problems.

The focus of the work presented in this thesis is to show fluid motion using solids solely as containers or obstacles along the path of the fluid to highlight the fluid's properties. For this reason, the fluid/solid interaction is one-way coupled, meaning that the movement of the fluid will be modified by a solid, but solids will remain immovable when a fluid exerts a force over them. Solids are represented as meshes made up of triangular polygons, and a particle is considered to have collided when its distance to the solid's mesh is less than the length of the particle's radius.

To be able to detect collisions, static particles are placed along the solids facets, and are spatially organized before the simulation process begins with the kd-tree structure. These static particles store to which facet they belong, and are placed at a distance of $h_0/4$ from each other, where h_0 is the initial smoothing length, ensuring that the fluid particles do not penetrate the facet.

To test if particle i has collided, a range search using the kd-tree over the static particles is performed centred at $\mathbf{x}_i$ position with a range of the particle's radius. After the range search has been performed, the subgroup of facets with which the particle could have collided is identified from the information stored on the static particles. If this subgroup is not empty, then the next step is to test if the particle's last movement was in the direction of each of the facets contained in that subgroup. If the test is positive the framework will considered the particle in a collision state and a response will need to be generated.

When testing for collisions using the direction of the particle's movement, it is required to take into account the orientation of the triangles that compose the mesh, otherwise there is the possibility that particles may become stuck or show erratic movements in the collision response stage.

Consider the problem illustrated in 2D in Figure 3.7, where a particle is approaching an edge of a solid, and the range search includes two opposing faces marked as A and B in the illustration. If the orientation (marked by the arrows protruding from the faces) is not considered, both A and B faces will be used for the collision response stage, which could lead to the particle being stuck. A more natural response is achieved by considering the direction of the mesh faces; in the case of the example illustrated in Figure 3.7, the collision test would only be true for face A, avoiding instabilities in the particle's movement due to having wrongly considered a collision with face B.

To perform the direction test, a ray is shot from the particle's previous position $\mathbf{x}_{t-1}$, to its current one $\mathbf{x}_t$. Müller et al.'s [56] method for detecting ray/triangle intersection is used, which produces the location of intersection $\mathbf{x}_c$ if it exists. If

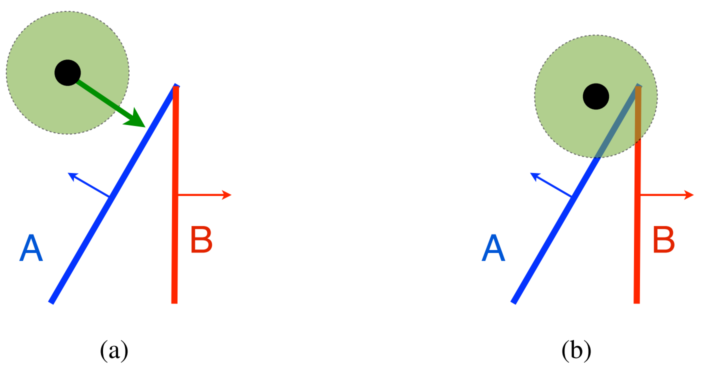


Figure 3.7: Example of possible collision detection and response problem.

such an intersection is found, and the facet is oriented towards the particle, then the particle is considered to be in a collision state, and the facet normal is stored for use during the response stage.

Once a collision has been detected, the position of the particle in question is reverted to its previous position $\mathbf{x}_{t-1}$. Its velocity also needs to be modified to finish the response process. This is done by separating the particle's velocity $\mathbf{v}$ into its normal and tangential components:

$$\mathbf{v}_{normal} = (\mathbf{v} \cdot \mathbf{N}_s)\mathbf{N}_s \tag{3.31}$$

and

$$\mathbf{v}_{tangent} = \mathbf{v} - \mathbf{v}_{normal} \tag{3.32}$$

where $\mathbf{N}_s$ is the normalized sum of the normal vectors belonging to all the facets with which the particle collided. This case occurs when a particle comes close to the mesh seams; this has to be taken into consideration to make sure the solids are water-tight.

The velocity after the collision is computed as:

$$\mathbf{v}_{new} = \beta_c \mathbf{v}_{tangent} - \alpha_c \mathbf{v}_{normal} \tag{3.33}$$

where α_c is the normal velocity coefficient that works as a dampening agent; β_c is

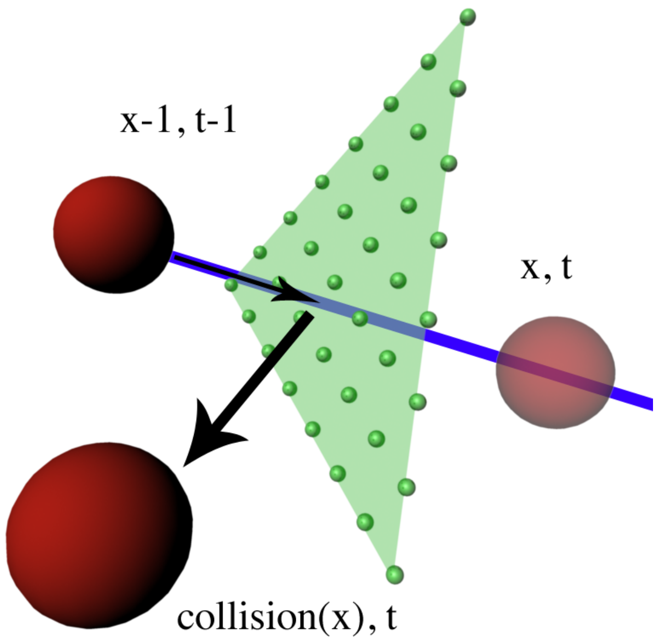


Figure 3.8: Particle collision detection and response.

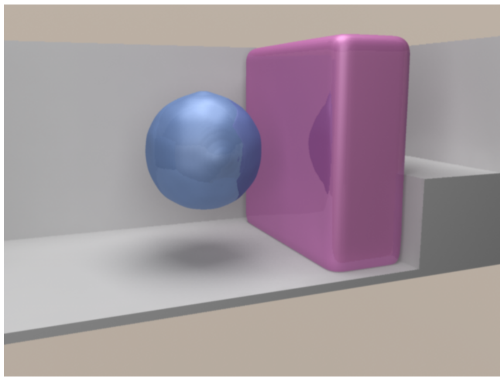
the surface friction parameter that enforces both the non-slip boundary condition when $\beta_c = 0$, and the free slip boundary condition when $\beta_c = 1$

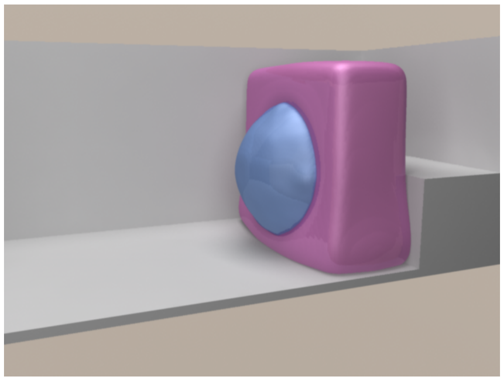
Figure 3.8 illustrates the collision detection and response process.

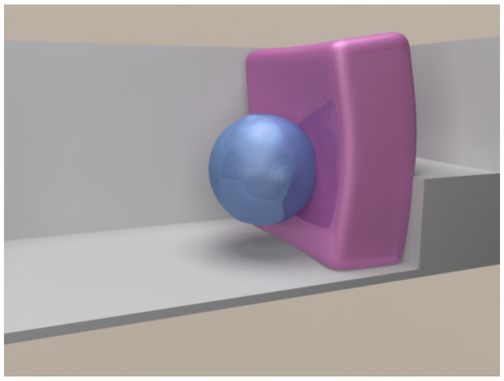
Although the focus of this framework does not deal with immiscible fluids, the system can handle this phenomenon in a similar fashion to Solenthaler et al.'s work by using a high k_{gas} in the pressure term on the NS equation. Figure 3.9 shows frames from two sets of animations with variations in the parameters to show a collision between semisolids (Figure 3.9(a), Figure 3.9(c) and Figure 3.9(f)), viscous fluids (Figure 3.9(b), Figure 3.9(d), and Figure 3.9(f)).

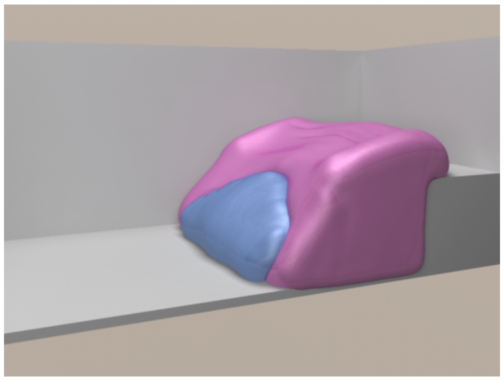
3.2 Numerical stability

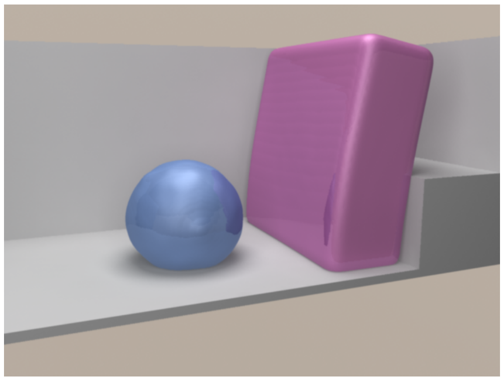
Numerical instabilities can be introduced to a SPH fluid simulation if fluid particles occupy the same space. These numerical instabilities can manifest in the form of small unnatural jittering up to explosions. Although the system treats the fluid


(a) Semisolids, Frame 10


(b) Viscous fluids, Frame 10


(c) Semisolids, Frame 25


(d) Viscous fluids, Frame 25


(e) Semisolids, Frame 60

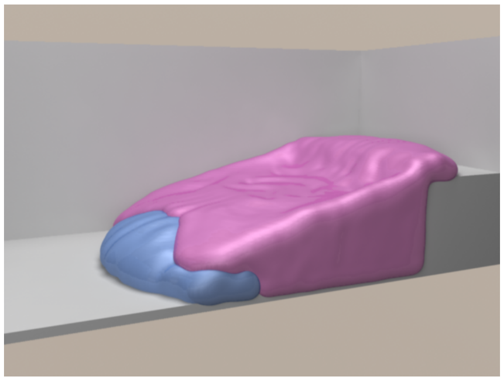

(f) Viscous fluids, Frame 60

Figure 3.9: Animation illustrating immiscible fluids.

as quasi-incompressible using Tait's Equation [6], it may not be enough avoid instabilities due to particles getting too close together to the point that they occupy the same space. For this reason the proposed framework implements two common solutions for this problem in tandem, which are the use of velocity correction by means of XSPH [57], and the use of artificial viscosity [86] .

3.2.1 Velocity correction

In order to avoid the problem of particle inter-penetration, Monaghan [57] proposes the SPH Lagrangian averaged model where the velocity of particle p_i is modified with the XSPH technique for calculating smoothed velocity. This is defined as:

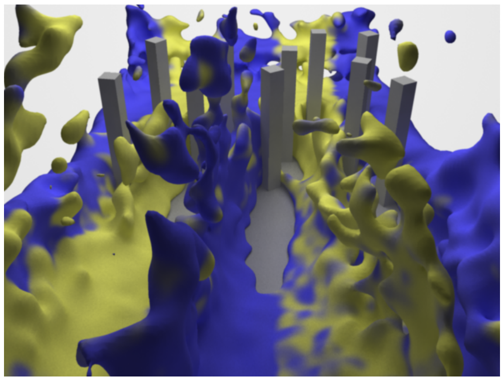
$$\mathbf{v}_i \leftarrow \mathbf{v}_i + \epsilon \sum_{j}^{N} \frac{2m_j}{\rho_i + \rho_j} (\mathbf{v}_j - \mathbf{v}_i) W(\mathbf{x}_{ij}, h) \tag{3.34}$$

where ϵ works as a velocity tuner and its value is clamped in the range of $0 \leq \epsilon \leq 1$. The smoothed velocity brings the particle velocity closer to the average velocity of its neighbourhood, and makes the particles move in an orderly fashion.

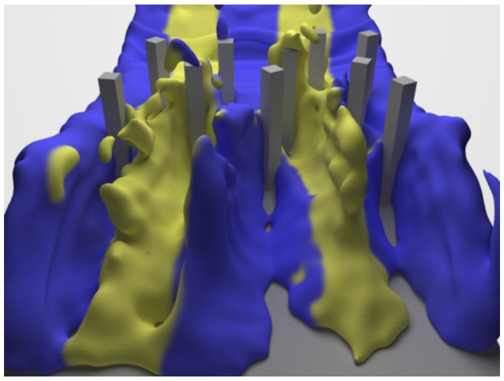
Figures 3.10 and 3.11 show the results of a series of animations where the velocity tuner is set to different values. It is evident from the renderings that, as the value of the velocity tuner ϵ increases, the particles move more orderly due to energy loss.

3.2.2 Artificial viscosity

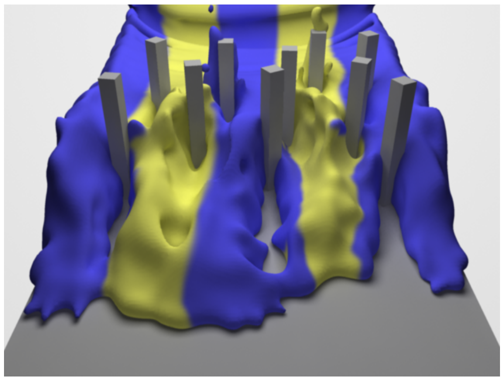
Problems of hydrodynamics found when dealing with shockwaves need special treatment, for simulation can develop unphysical oscillations in the numerical results around the shocked region. For the simulation to remain stable, the kinetic energy needs to be transformed into heat energy in the shock wave front. This transformation of energy can be physically represented as a form of viscous dissipation. This last idea leads to the development of the von Neumann-Richtmyer artificial viscosity [86] which makes the shock transition smoother without giving up the physical correctness.



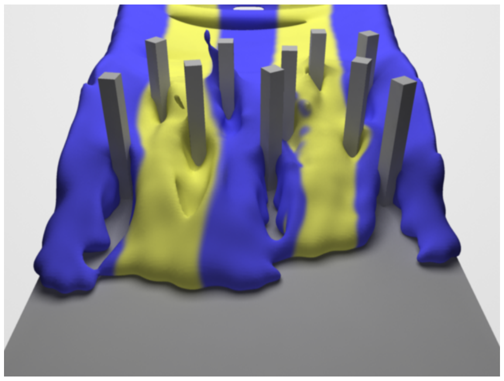
(a) $\epsilon = 0.0$, Frame 100



(b) $\epsilon = 0.25$, Frame 100

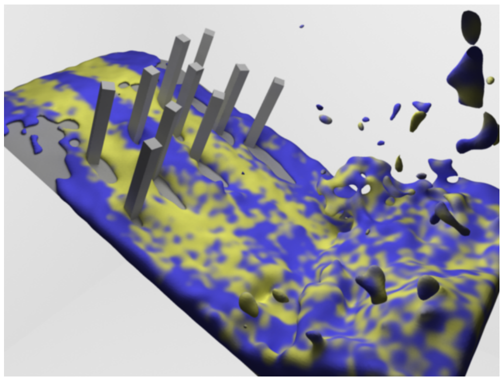


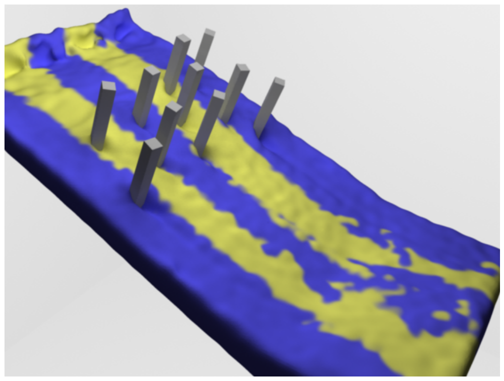
(c) $\epsilon = 0.50$, Frame 100

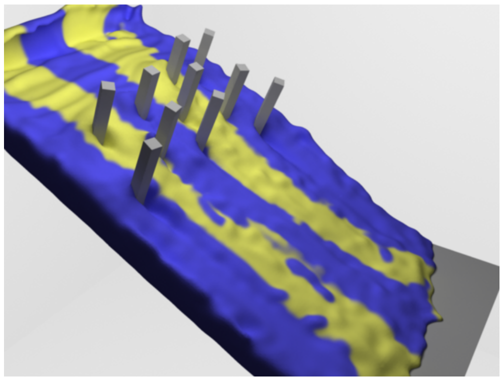


(d) $\epsilon = 0.75$, Frame 100

Figure 3.10: Top-view comparison of the effects of ϵ.


(a) $\epsilon = 0.0$, Frame 200


(b) $\epsilon = 0.25$, Frame 200


(c) $\epsilon = 0.50$, Frame 200

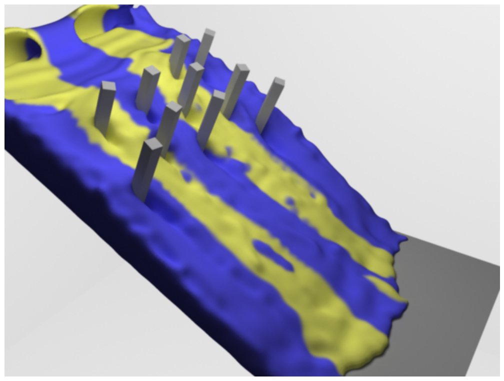

(d) $\epsilon = 0.75$, Frame 200

Figure 3.11: Front-view comparison of the effects of ϵ.

Monaghan et al. [58] [59] apply artificial viscosity under the SPH formalism to facilitate shock simulation by providing the necessary dissipation to convert kinetic energy into heat at the shock front. Furthermore, their approach also prevents unphysical penetration for particles approaching one another [40] [57]. The SPH implementation of the standard artificial viscosity is:

$$\frac{d\mathbf{v}_i}{dt} \leftarrow \frac{d\mathbf{v}_i}{dt} - \sum_{j}^{N} m_j \Pi_{ij} \nabla W(\mathbf{x}_i - \mathbf{x}_j, h) \tag{3.35}$$

where the viscosity tensor Π_{ij} is defined by

$$\Pi_{ij} = \begin{cases} \frac{-\alpha_\Pi \bar{c}_{ij} \mu_{ij} + \beta_\Pi \mu_{ij}^2}{\bar{\rho}_{ij}} & \mathbf{v}_{ij} \cdot \mathbf{x}_{ij} < 0 \\ 0 & \mathbf{v}_{ij} \cdot \mathbf{x}_{ij} \geq 0 \end{cases} \tag{3.36}$$

where

$$\mu_{ij} = \frac{h \mathbf{v}_i j \cdot \mathbf{x}_{ij}}{|\mathbf{x}_{ij}|^2 + \epsilon h^2} \tag{3.37}$$

$$\bar{c}_{ij} = \frac{1}{2}\left(c_i + c_j\right) \tag{3.38}$$

$$\bar{\rho}_{ij} = \frac{1}{2}\left(\rho_i + \rho_j\right) \tag{3.39}$$

$$\mathbf{v}_{ij} = \mathbf{v}_i - \mathbf{v}_j, \mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j \tag{3.40}$$

In Equation 3.35 α_Π and β_Π are constants set to 1.0; ϵ_Π is a constant that takes the value of 0.01, introduced to prevent numerical divergences when $|\mathbf{x}_{ij}| = 0$; c_i and c_j represent the speed of sound at particles i and j respectively; and **v** is the velocity vector.

The viscosity associated with α_Π produces a bulk viscosity while the term associated with β_Π is similar to the von Neumann-Richtmyer viscosity. Because the proposed framework deals with viscosity explicitly using the SPH formalism as described on Subsection 3.1.5, the term associated with α_Π is ignored since it could lead to spurious large sheer viscosity. However, the term associated with β_Π is still used to prevent particle penetration. By setting the value of $\alpha_\Pi = 0$, the artificial viscosity term Π_{ij} (used in the proposed framework) takes the form of:

$$\Pi_{ij} = \begin{cases} \frac{\beta_\Pi \mu_{ij}^2}{\bar{\rho}_{ij}} & \mathbf{v}_{ij} \cdot \mathbf{x}_{ij} < 0 \\ 0 & \mathbf{v}_{ij} \cdot \mathbf{x}_{ij} \geq 0 \end{cases} \tag{3.41}$$

3.3 New phenomena

This section introduces the new heat-based phenomena developed using the SPH framework, including fluid/solid phase changes based on the work by Mao and Yang [51] [52] [53] and the newly proposed phenomenon of simulating volume expansion given by bubbles trapped inside a non-Newtonian fluid.

3.3.1 Heat transfer

The simulation of heat transfer is a very important component of the presented framework: the change in the fluid's temperature is the main driving force in the animation of the solid-fluid phase change and volume expansion. For simplicity's sake, only two types of heat transfer are considered: the heat transfer from external sources such as air, and the internal heat transfer. This assumption distances the proposed framework from a more physically accurate model that considers radiation and heat from the water phase changes inside the dough while still keeping the framework within the realm of physical-plausibility in generating a temperature gradient throughout the baking process.

External Sources Heat Transfer

The external heat transfer takes place at the surface of the fluid on the fluid-air, or fluid-solid interfaces. The present framework models this transfer following the work by Stora et al. [82] using Equation 3.42:

$$\left(\frac{dT}{dt}\right)_{ext_i} = k_{hext}(T_i - T_{ext})\frac{r_i^2}{\rho_i} \tag{3.42}$$

which takes into account the surface area of the particles located at the interface of the fluid. In Equation 3.42 k_{hext} is the external heat transfer coefficient, T_{ext} is the temperature from the external object with which the heat transfer is occurring (be it on the fluid-air or fluid-solid interfaces), and finally T_i, ρ_i, and r_i are the temperature, density, and radius (respectively) belonging to particle p_i.

The radius of particle i can be computed using the following Equation:

$$\frac{4}{3}\pi r_i^3 = \frac{m_i}{\rho_0} \tag{3.43}$$

The external heat transfer takes place only on those particles determined to be on the surface of the fluid using the local convex hull method that is described on Section 3.1.8.

Internal Heat Transfer

A popular method for computing the internal heat transfer that takes place in both grid [13] and particle-based fluid models [82] [62] [70] is using the general heat equation:

$$\frac{\partial T}{\partial t} = k_{hint}\nabla^2 T \tag{3.44}$$

where k_{hint} is the internal heat transfer coefficient.

However, this approach assumes that the fluid's density is a constant, an assumption that is difficult to maintain under the SPH formalism since, regardless of how the pressure term is computed, some variation in the particle's density value will occur. To avoid this problem Mao and Yang [52] [51] propose a more flexible approach as described in Equation 3.45:

$$\frac{\partial T}{\partial t} = k_{hint}\frac{1}{\rho}\nabla^2 T \tag{3.45}$$

The assumption of a constant density is left behind and all that is needed to obtain the rate of change of temperature is to compute the temperature Laplacian for particle i which is computed under the SPH formalism as:

$$\frac{1}{\rho_i}\nabla^2 T = \sum_{j=1}^{N} \frac{4m_j}{(\rho_i + \rho_j)^2}(T_j - T_i)\nabla^2 W(r_{ij}, h) \tag{3.46}$$

Figure 3.12 shows screenshots taken from the realtime view of the proposed framework implementation illustrating the results from Mao and Yang's proposal for simulating internal heat transfer. Each screenshot shows the fluid's particles at increasing time periods with the changes in colour representing the changes in each particle's temperature, showing the temperature gradient in the dough.

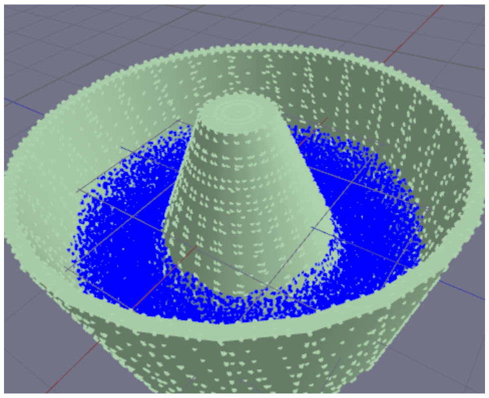
3.3.2 Phase changes

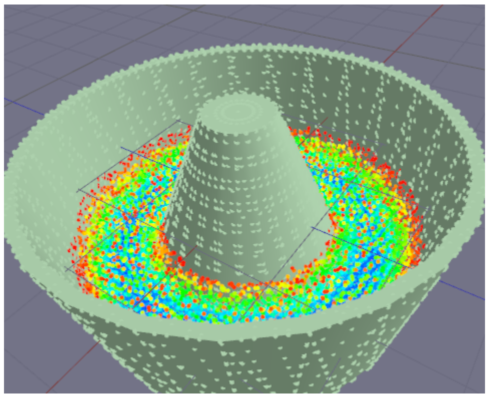
Phase transition is an interesting phenomena that has been studied before under the SPH formalities by Mao and Yang [51] [52] [53], and Paiva et al. [70]. The framework presented in this thesis builds upon the work by Mao and Yang with Non-Newtonian fluids for describing phase transitions, and extends it by implementing non-uniform viscosity throughout the fluid, making it temperature-dependent. This framework also makes use of the XSPH velocity tuner to absorb energy, making the fluid-solid phase transition animation more realistic.

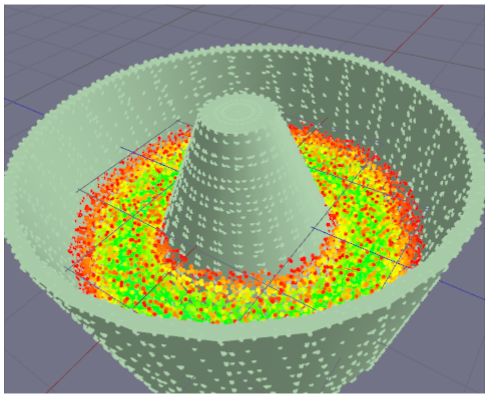
Elasticity

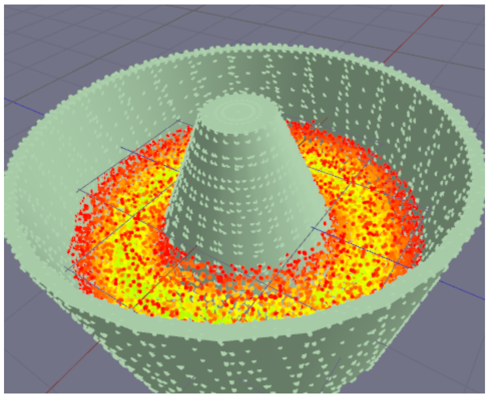
Mao and Yang [51] [52] present a method for creating animations of phase changes by modifying the μ_e and λ values of the stress equation introduced to create animations of Non-Newtonian fluids. Figure 3.13 shows frames of an animation where a solidification effect can be achieved by varying μ_e and λ based on temperature change that takes place after the fluid comes into contact with the floating structure.

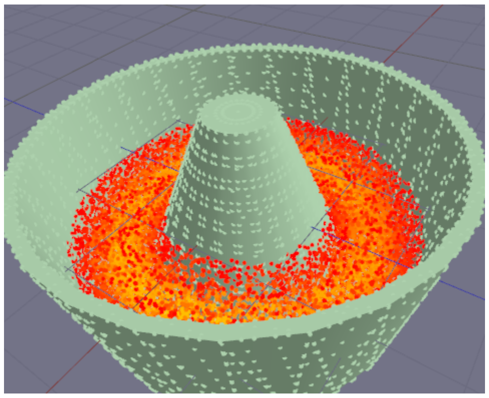
Figure 3.14 illustrates the opposite effect where μ_e and λ change due to the increase in temperature from the floor, and the bunny structure melts away turning into a fluid.


(a) Frame 72


(b) Frame 122


(c) Frame 172


(d) Frame 222


(e) Frame 272

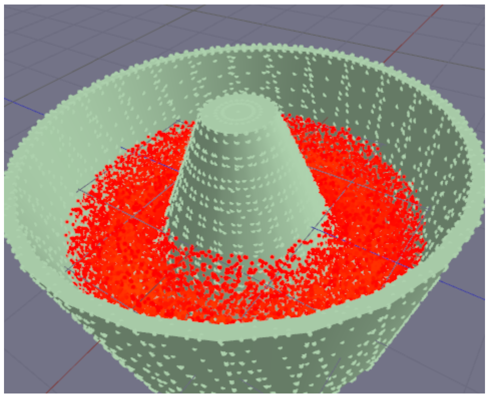
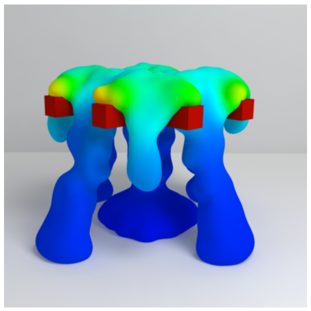
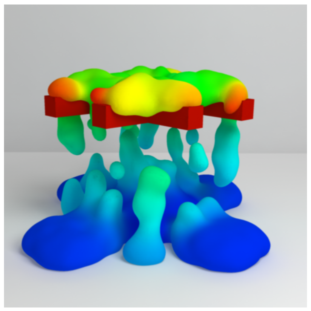
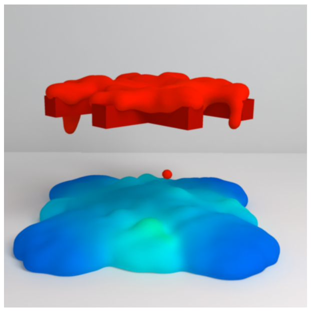
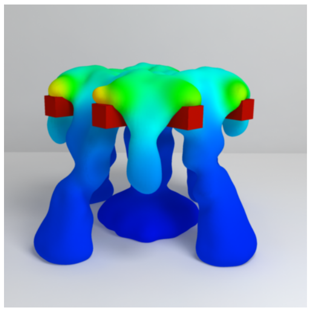

(f) Frame 372

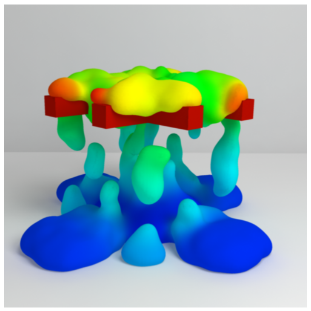
Figure 3.12: Heat transfer screenshots from real-time view.

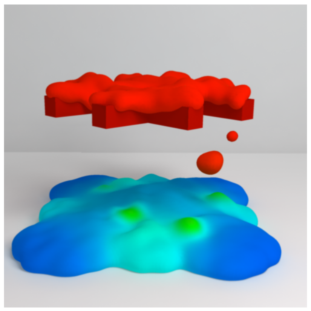

(a) Constant λ and μ_e, Frame 95

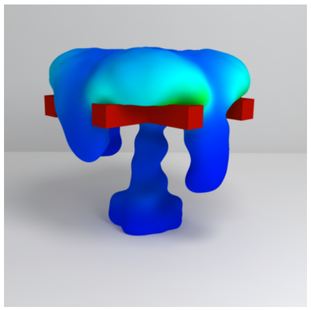

(b) Constant λ and μ_e, Frame 135

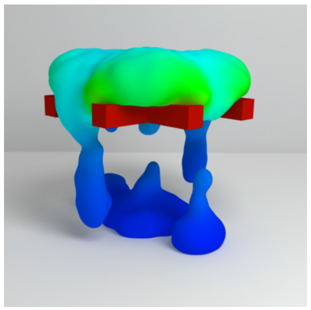

(c) Constant λ and μ_e, Frame 400

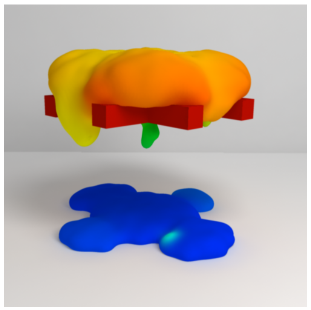

(d) Temperature dependent λ, Frame 95

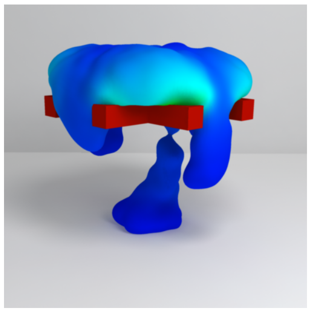

(e) Temperature dependent λ, Frame 135

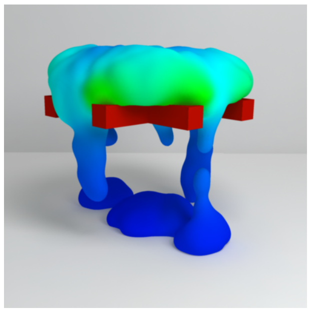

(f) Temperature dependent λ, Frame 400


(g) Temperature dependent μ_e, Frame 95


(h) Temperature dependent μ_e, Frame 135


(i) Temperature dependent μ_e, Frame 400


(j) Temperature dependent λ and μ_e, Frame 95


(k) Temperature dependent λ and μ_e, Frame 135

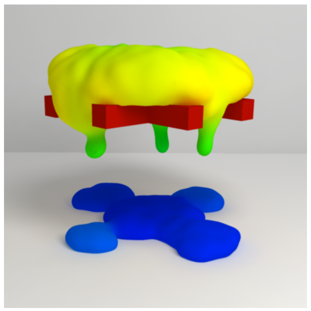

(l) Temperature dependent λ and μ_e, FSrame 400

Figure 3.13: Effects of varying λ and μ_e based on temperature.

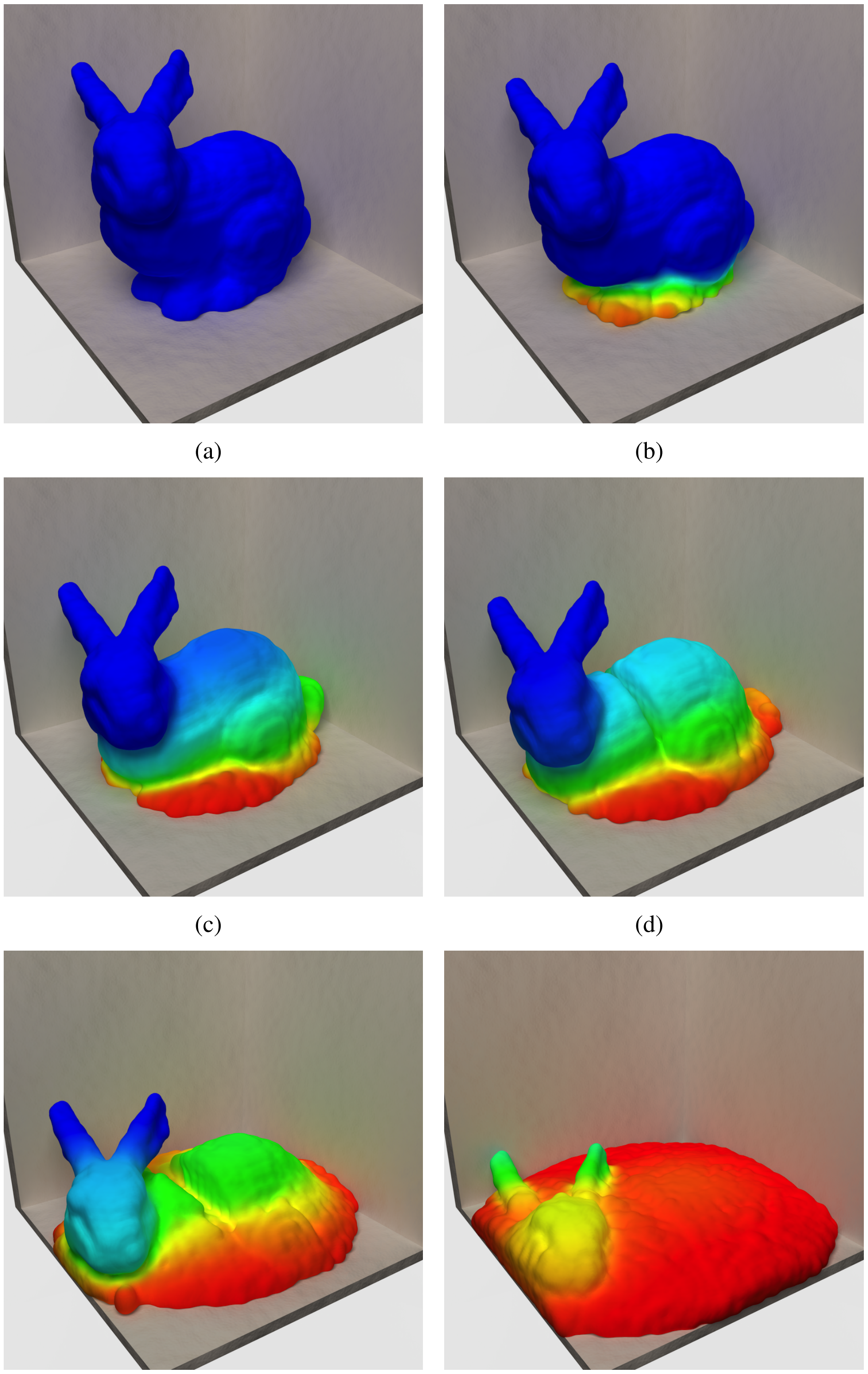

(a) (b) (c) (d) (e) (f)

Figure 3.14: A melting animation illustrating the effects of varying λ and μ_e based on temperature.

Viscosity

Viscosity is another parameter that is very important during the fluid-solid phase transition. As a matter of fact, in Paiva et al.'s [70] model the solids are nothing more than highly viscous fluids. In order to model the fluid with non-uniform viscosity, the present framework implements the method proposed by Müller et al. [64] where the viscosity at particle i is defined by Equation 3.47:

$$f_i^{viscosity} = \mu_i \sum_j \frac{m_j}{\rho_j}(\mathbf{v}_j - \mathbf{v}_i)\nabla^2 W(\mathbf{x} - \mathbf{x}_j, h) \tag{3.47}$$

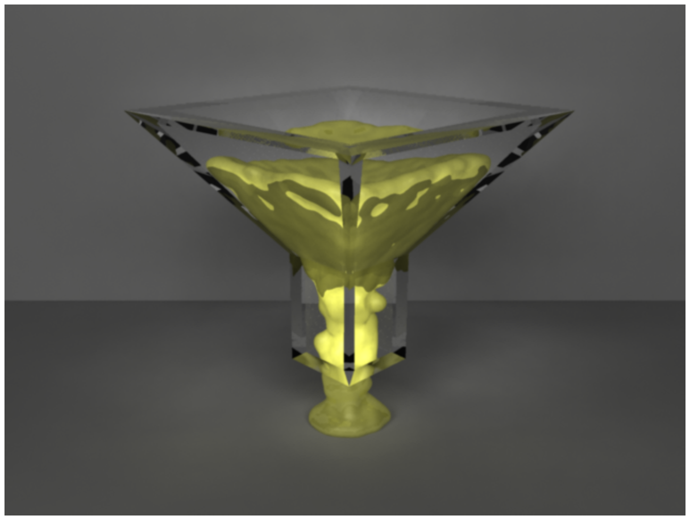
where μ_i is the viscosity at each particle p_i, and its value can be modified depending on the temperature of particle p_i. This approach is used by Stora et al. [82] with the purpose of animating lava flows.

The changes in the fluid's motion can be too subtle to notice due to gradually varying viscosity. In order to better illustrate the effects of the viscosity parameter, the used examples contain fluids with very different homogenous viscosity values. Figure 3.15 shows a fluid with low viscosity that travels through the glass funnel very easily. In contrast, Figure 3.16 shows a very viscous fluid: a closer look will reveal that at the end of the animation there is a significant amount of fluid residue still in the funnel.

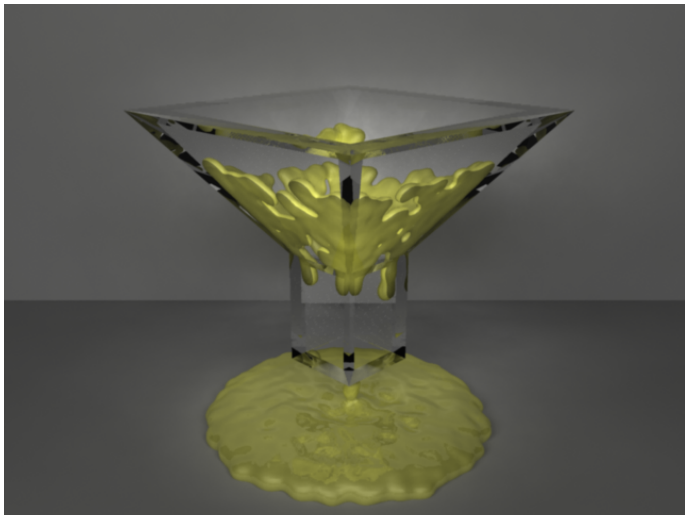
Velocity tuner

As discussed in Section 3.2.1 the proposed framework uses the XSPH technique to make the fluid particles move in an orderly fashion. Monaghan [62] describes this version of XSPH, and mentions that it may not be optimal in some cases because it introduces a loss in energy while correcting the particles' velocities. However, a goal of the proposed framework is to create phase transition in fluids; this energy loss is actually required to achieve a relaxed state in the particles, and it can be harnessed to create a more varied range of animations.

Figures 3.10 and 3.11 illustrate the effects of modifying the velocity tuner uniformly for the whole fluid; however, this parameter can also be modified in a non-

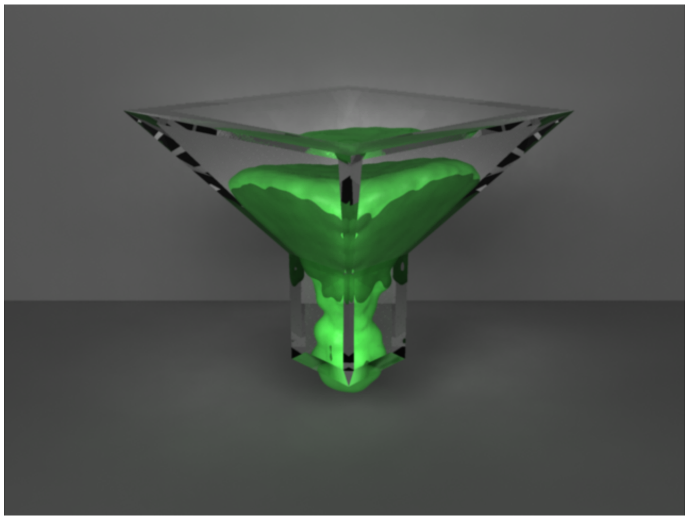


(a) v = 1, frame 66



(b) v = 1, frame 360

Figure 3.15: Animation of a fluid with low viscosity.


(a) v = 1×10^5, frame 66

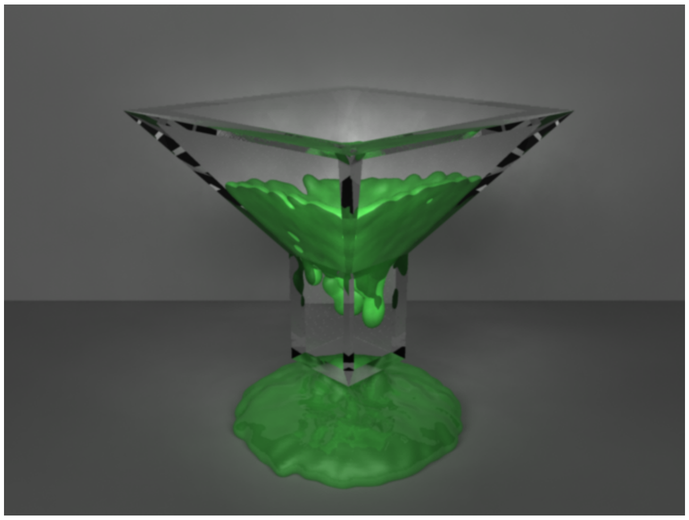

(b) v = 1×10^5, frame 360

Figure 3.16: Animation of a fluid with high viscosity.

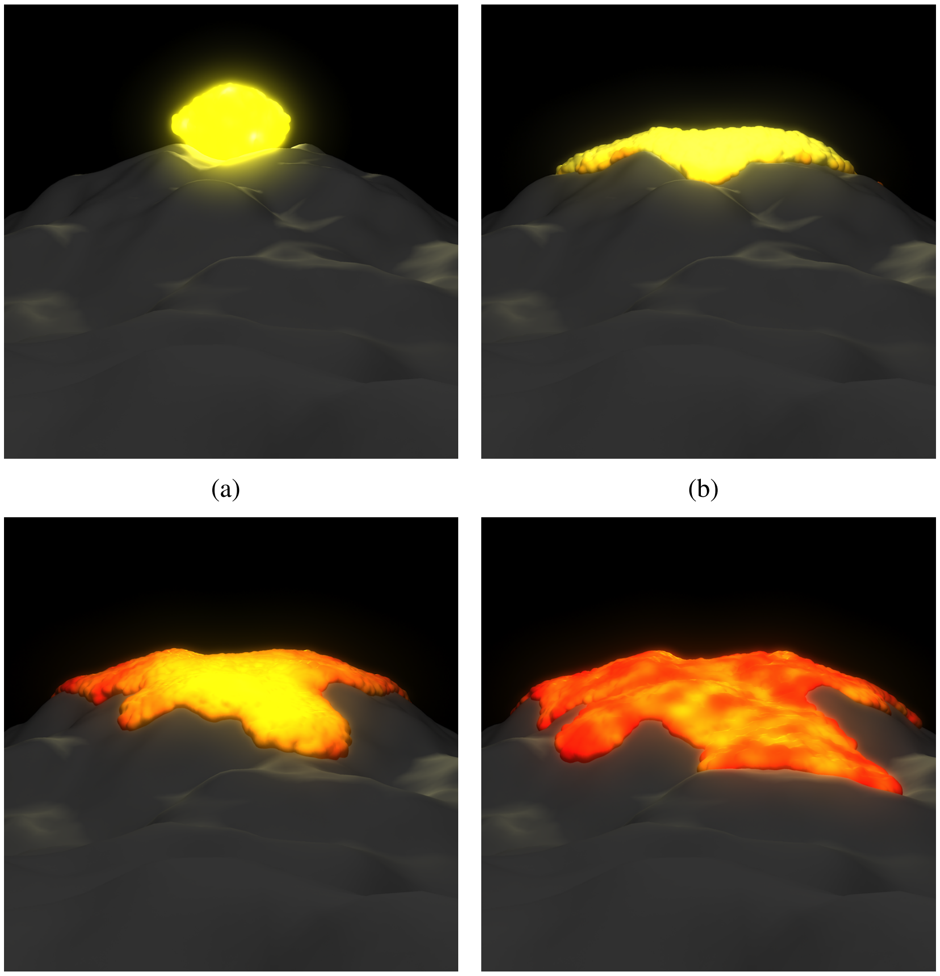


Figure 3.17: Animation illustrating the effects of variable velocity tuner.

uniform way throughout the fluid. Figure 3.17 shows a scene commonly used to describe varying viscosity under the SPH formalism. Nevertheless, in this case the viscosity remains fixed at a value, and we achieve a similar result by linearly interpolating the value of the velocity tuner based on the particles' temperatures. In the illustration it is noticeable how the lava that flows to the left and right of the volcano cools of quickly, and stops flowing, whereas the lava that flows from the front of the volcano keeps its high temperature for a longer period of time, and manages to flow further down the volcano.

3.3.3 Volume expansion

Pressure changes

The pressure term from the NS equation of motion as described in Equation 3.1 is used to regulate the overall volume of the fluid, keeping the fluid in a quasi-incompressible state by 1) adjusting the distance between the particles, exerting a positive force between particles as they come closer to drive them apart, and 2) a negative force that drives the particle together as the distance between them grows too large. This pushing and pulling between the particles is in an effort to keep the particle's density as close as possible to its resting density value. As described in Subsection 3.1.4 the proposed framework uses Tait's Equation [6] to handle the pressure term of the NS Equation. However, the pressure term can be manipulated to make the fluid increase in volume in a controlled manner.

To achieve volume expansion the particles are tagged as being simple fluid particles, or particles that will be treated as bubbles of air trapped in the fluid. This process make the fluid increase in volume due to temperature changes that augment the overall volume of the fluid. For all intents and purposes, fluid and bubble particles will be treated in the exact same way except when dealing with the particles' pressure. Initially, the bubble particles will use Tait's Equation [6] to handle pressure, but when the temperature begins to change Equation 3.48 is used to handle the bubble particle pressure:

$$P_{bi} = B\frac{\rho_i}{\rho_0}\left(\frac{\rho_0}{\rho_{bi}} - 1\right) \tag{3.48}$$

where B is a constant that scales the effects of the changes in pressure, $\frac{\rho_i}{\rho_0}$ is a throttling factor that reduces the effect of the pressure changes as the particles grow apart. This throttling factor makes the animation more stable than relying only on the SPH framework to attenuate the applied forces due to the distance between the particles. The term $\left(\frac{\rho_0}{\rho_{b1}} - 1\right)$ is the driver for the increase in pressure, and it increases due to ρ_{b_i}, which is the bubble density which is defined as:

$$\rho_{bi} = \frac{\rho_0}{V_{bi}} \tag{3.49}$$

the framework assumes the bubble to follow the ideal gas state law, and it assumes that the process is of an isobaric nature so the bubble volume of the particle is defined as:

$$V_{bi} = \frac{V_{b0} T_i}{T_0} \tag{3.50}$$

where V_{b0} is the particles' initial volume; for the purposes of our simulation this parameter is set to 1.

By substituting Equation 3.49 and Equation 3.50 into Equation 3.48 we get the final form of the pressure equation for bubble particles:

$$P_{bi} = B \frac{\rho_{bi}}{\rho_0} \left(\frac{\mathbf{V}_0 T_i}{T_0} - 1 \right) \tag{3.51}$$

However, Equation 3.51 for bubble pressure cannot be used indefinitely since the volume will continue to increase. In order to avoid this, the pressure term changes back to the method defined by Monaghan [61] as described in Equation 3.17.

Changing back to Equation 3.17 for pressure allows a nonaggressive transition from the increasing volume to a relaxed state avoiding instabilities in the animation in the form of jittering in the particles movement. However, to keep the changes in volume, Tait's Equation [6] needs to be modified, or else, the fluid will return to its initial state. The modification needed is quite simple: while the volume is increasing by using Equation 3.48, the framework keeps track of the particles' density storing it as ρ_{f_i}, and the final pressure equation takes the form of:

$$P_i = B \left(\left(\frac{\rho}{\rho_{f_i}} \right)^{\gamma} - 1 \right) \tag{3.52}$$

Adaptive smoothing length

Due to the changes in volume, the system must compensate for the distance between particles so that they are "connected" by a dough medium and do not fall apart as the volume increases. To avoid a breaking effect in the fluid, Mao and Yang [52] [51] propose up-sampling the fluid particles as their positions evolve over time, achieving a more realistic animation. The problem with this approach is that, without their proposed method of calculating the particles' pressure using the projection method under the SPH formulation to conserve volume, the newly created particles will present instabilities in the form of jittering motions in the initial frames while the system stabilizes.

Liu and Liu [45] describe a method to adapt each particle's individual smoothing length h with the purpose of maintaining the number of particle's neighbours relatively constant. The method they describe is based on updating the smoothing length using the particle's density value as described in Equation 3.53:

$$h_i = h_0 \left(\frac{\rho_0}{\rho} \right)^{1/\mathrm{d}} \tag{3.53}$$

where h_0 and ρ_0 are the initial smoothing length and initial density respectively, and d is the number of dimensions. This method however becomes unstable for particles located at the surface of the fluid if the adaptive smoothing length is not restricted to a given range due to the problem described on Section 3.1.3, where the density of the outlying particles is erroneously calculated.

A different approach to calculate the smoothing length is proposed by Sigalotti et al. [78] who use an adaptive smoothing length to enforce constant density at the surface of the fluid. Their adaptive smoothing length method is similar to the one described by Liu and Liu and takes the form of:

$$h_i = k_g \left(\frac{\rho_i}{\bar{g}} \right)^{-\epsilon_g} h_0 \tag{3.54}$$

where k_g is a scaling factor of the order of unity, ϵ_g a sensitivity parameter that exists in the interval of $0 \leq \epsilon_g \leq 1$, and $\bar{g}$ the geometric mean of the initial estimates given by:

$$\log \bar{g} = \frac{1}{N} \sum_{j}^{N} \log \rho_j. \tag{3.55}$$

As the value of the sensitivity parameter ϵ_g approaches 1, the smoothing length h_i becomes more sensitive to the density distribution, creating a longer smoothing length as the density ρ_i of the particle decreases. On the contrary, when the value of ϵ_g approaches 0, this method reduces the adaptive smoothing length to the fixed smoothing length approach. Sigalotti et al.'s [78] approach produces much more stable results by maintaining a medium between the particles as the fluid expands, while also solving the problem of incorrect particle densities at the surface of the fluid; for this reason their method is used in this work.

Because we are dealing with a varying smoothing length per particle depending on each particle's density a symmetric interaction must be enforced in some way; otherwise, there will not guarantee that when particle i exerts force on particle j, an equal and opposite reaction will occur, since the area of effect of each particles may not be the same, violating Newton's third law.

To achieve the symmetric interaction the framework averages the smoothing lengths of the pair of interacting particles as described by Benz [8]:

$$h_{ij} = \frac{h_i + h_j}{2}. \tag{3.56}$$

3.4 Numerical integration

After the properties of all the particles have been computed at the end of each time step all that is needed to proceed is to integrate the movement of particles.

The proposed framework adopts the Euler integration method due to its ease of implementation and low memory consumption. According to the Euler integration method, the particle's velocity is integrated as:

$$\mathbf{v}_i^{b+1} = \mathbf{v}_i^b + \mathbf{a}_i^b \Delta t \tag{3.57}$$

where b is the bth time step, Δt the time step itself, and $\mathbf{a}_i$ the particle's acceleration computed using the SPH framework as described in Section 3.1. After integrating the particle's velocity, the corresponding position is computed as:

$$\mathbf{x}_i^{b+1} = \mathbf{x}_i^b + \mathbf{v}_i^{b+1} \Delta t \tag{3.58}$$

3.5 Surface reconstruction

After generating the fluid animation the results are in the form of points moving in space rendered in real time. However, this is not the best solution for representing the fluid since it does not show the volume of the fluid as it moves. In order to see a more detailed description of the fluid, a mesh is created taking the points that define the fluid as input.

Several methods were tested for creating such a mesh trying to avoid the common problem that occurs with the marching cubes algorithm when used with certain field functions. In these cases, the resulting surface will often go beyond the volume defined by the fluid particles, penetrating the solid container. Furthermore, when reconstructing the fluids surface, the volume expansion needs to be taken into account so that the mesh of the fluid expands with the particles as they grow apart, and we do not get particles suspended in space.

Figure 3.18 illustrates an extreme scenario with a very coarse particle distribution where the volume of effect of the field function is underestimated. The full animation originally includes a fluid inside a container, but for clarity the container is omitted. As it can be seen from the illustration, even though the initial volume is rendered correctly (Figure 3.18(a)), the fluid breaks apart and levitates in an unrealistic way as the volume expands (Figure 3.18(b)).

The opposite case would be to naively use a large volume of effect for the field function avoiding the levitating fluid as shown on a second animation in Figure 3.19.

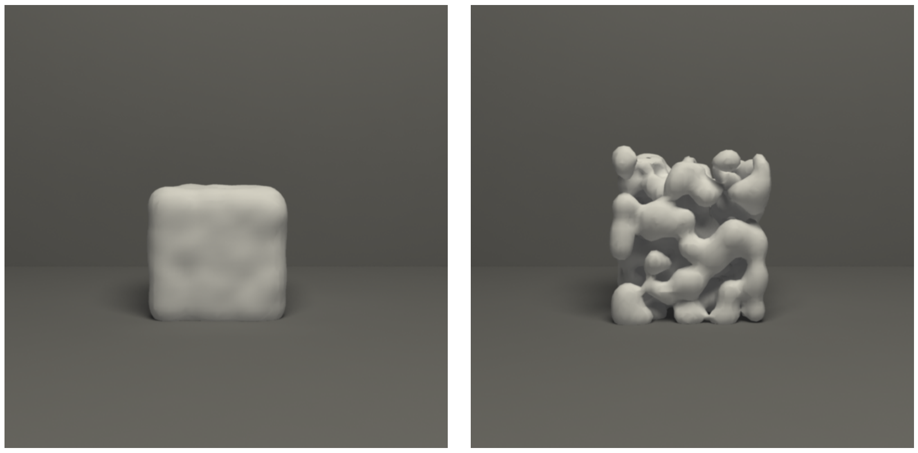

(a) Frame 26 (b) Frame 586

Figure 3.18: Underestimating the volume of effect for field function.

In this frame, the volume has already expanded to its fullest. Although the fluid does not levitate in this test, and the quality of the surface may be sufficient, if the volume of effect is too large there is the risk that the fluid will penetrate the solid in which it is contained, and this is not acceptable.

The methods tested emphasize on creating a mesh whose vertices are very close to the points of the surface of the input data set. One of the several algorithms tried is the Poisson surface reconstruction method [35]; this algorithm takes as input only the points that lie on the surface of the fluid along with the normals at such points to create a scalar field. The scalar field is eventually used to get the mesh of the fluid using the marching cubes algorithm. This method turned out to be robust enough when creating single frames, and it is very robust to outlier points. However, this method is not optimal when creating a sequence of frames to make up the animation because of the nature of the algorithm, which makes it non-deterministic, creating frames that were not temporally coherent. Even more, when running this method several times with the same data set, slightly different results are obtained. This temporal discrepancy is shown in Figure 3.20 where four meshes are generated from a single frame—each colour coded—and when they are superimposed their shapes to not match and the colours show which is the mesh that extends the furthest.

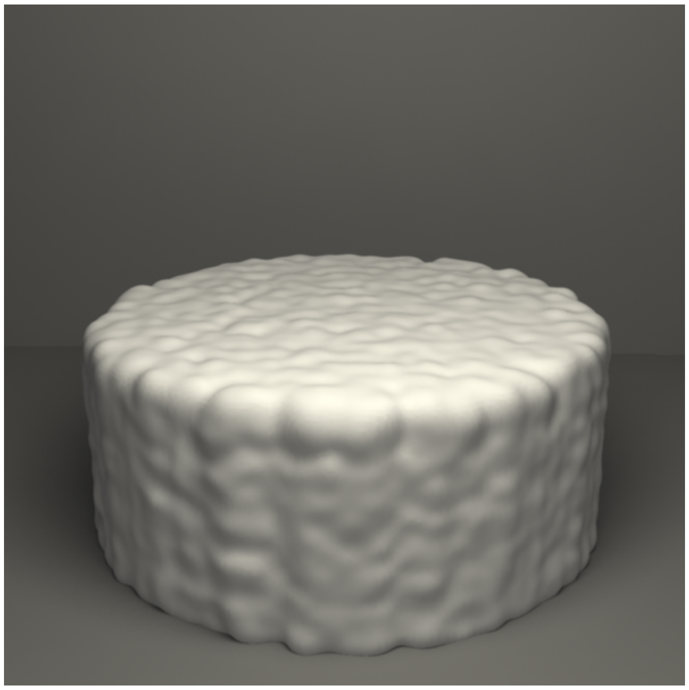


Figure 3.19: Overestimating the volume of effect for field function.

Another method tested to create the fluid's surface is the robust implicit moving least squares method (RIMLS) approach [83]. Similarly to the Poisson reconstruction method, this method takes as input the positions of the particles on the surface of the fluid along with their respective normals. However, contrary to the Poisson method, this method did give temporal coherent results, but unfortunately the RIMLS approach is very sensitive to the changes in particle positions and their normals. Figure 3.21 illustrates this problem with an early animation test using the RIMLS approach. The surface reconstruction behaves as expected creating a smooth expanding volume until it steps from frame 158 (Figure 3.21(a)) to frame 159 (Figure 3.21(b)). Here the normals of the surface particles change enough that a drastic change takes place in the resulting animation, resulting in an unrealistic jump on the surface of the fluid. To better illustrate the problem comparing static frames, Figure 3.21(c) shows the rendering of both surfaces superimposed without the container, applying a red material on the surface of frame 158, and a blue ma-

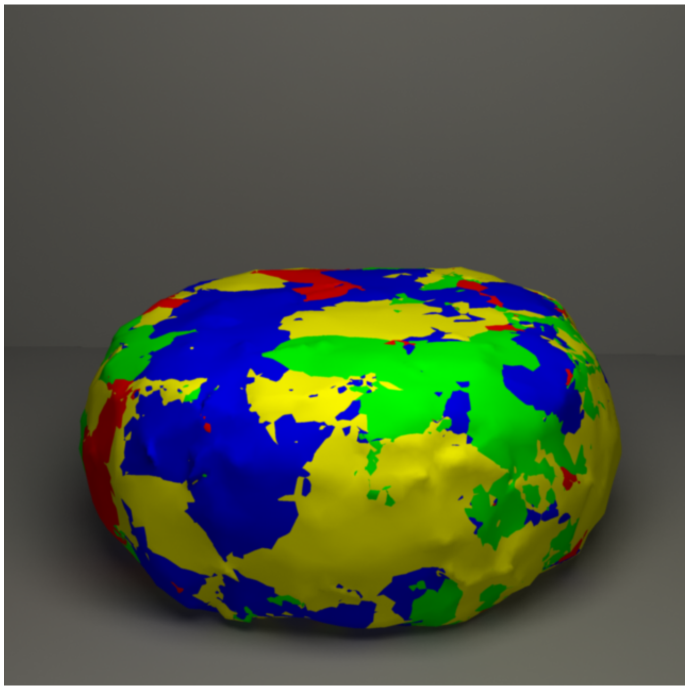
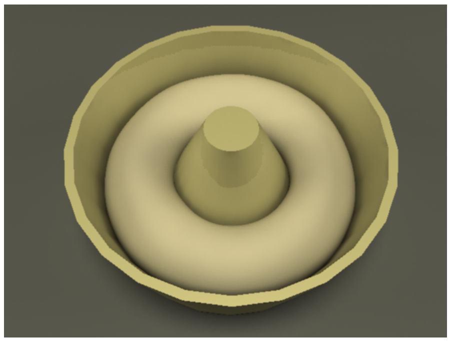


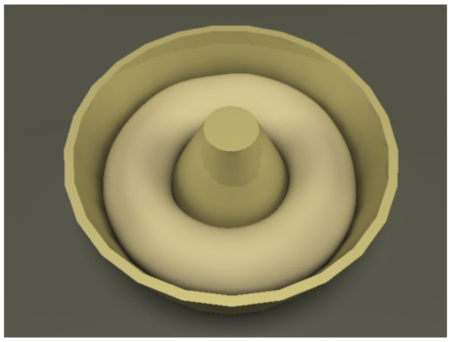
Figure 3.20: Result of the Poisson reconstruction method showing temporal discrepancy.

terial on the surface of frame 159. It is evident that there is a big shift in the shape of the surface in this frame transition. However, since meshes come from different frames a change must be visible the animation comes from the slow increase in volume of the fluid and the change should be smoother; instead the result shows a big deformation in the fluid which in the animation looks as the aforementioned unrealistic jump from one frame to the another, and although the rest of the sequence is without fault, this problem makes this approach less than ideal for animation purposes.

Another method tried in this work, which steps away from reconstructing the scalar field, is the Alpha-shapes algorithm [20] [19] [21] . Due to the nature of this method, the vertices of the resulting mesh share the same positions as those from the input data set, and although the resulting mesh can be coarse, it can be smoothed out using a smoothing technique such as the Loop subdivision algorithm [47]. However, this method had to be discarded as well because of the need of setting the alpha parameter. This alpha parameter can be set to define the number of solid objects we want based on the work of Bernardini and Bajaj [9]. Nevertheless, if this approach is used for the entire animation, we would need to define the number of solid objects the input set would create for every frame: creating a natural looking animation using this approach would be very difficult, and the alternative is to calculate this value for the first frame, then continue to creating the rest of the frames reusing this value. This approach is, again, not without problems: after the particles start to move around there may be the case when the particles grow apart (such as in the proposed volume expansion method), and the holes inside the alpha shape could create other objects that create intersections with the main fluid mesh, or even holes in the surface of the fluid. This last extreme example is illustrated on Figure 3.22, where Figure 3.22(a) shows a complete mesh (left without smoothing to better illustrate the results) and the last frame on Figure 3.22(b), where holes have appeared and the notion of a continuous surface is gone.

As stated before, neither the Poisson reconstruction or RIMLS methods for cre-


(a) Frame 158


(b) Frame 159

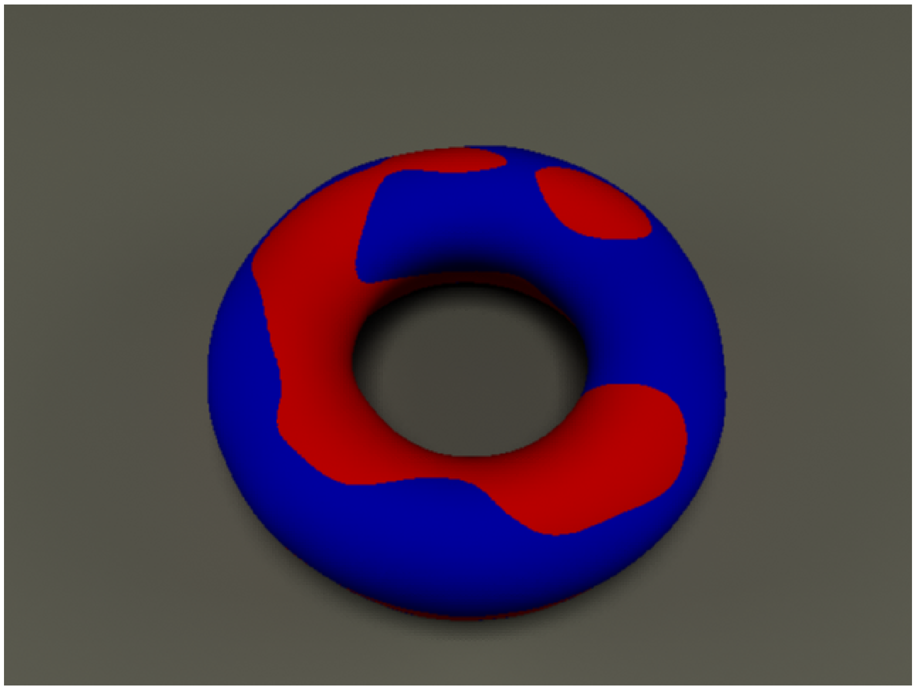
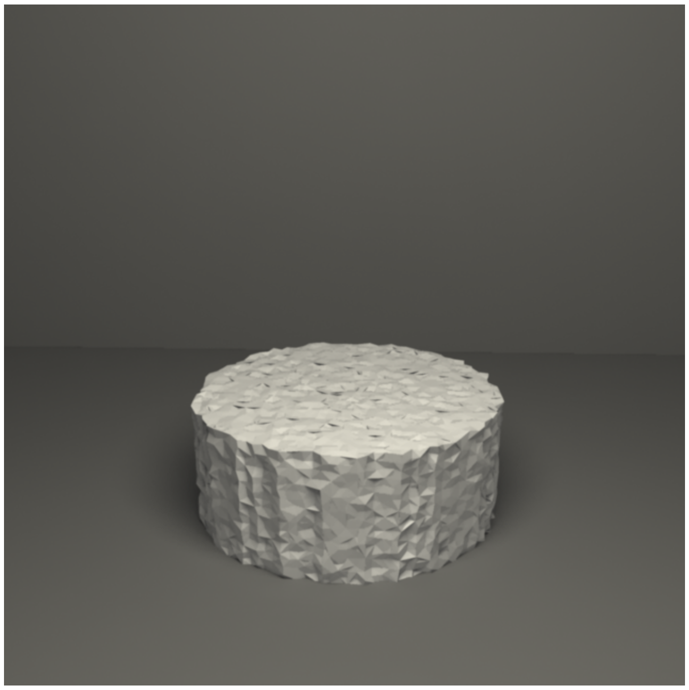

(c) Side by side comparison

Figure 3.21: Subsequent frames showing temporal discrepancy using the RIMLS algoritm.


(a) Frame 1

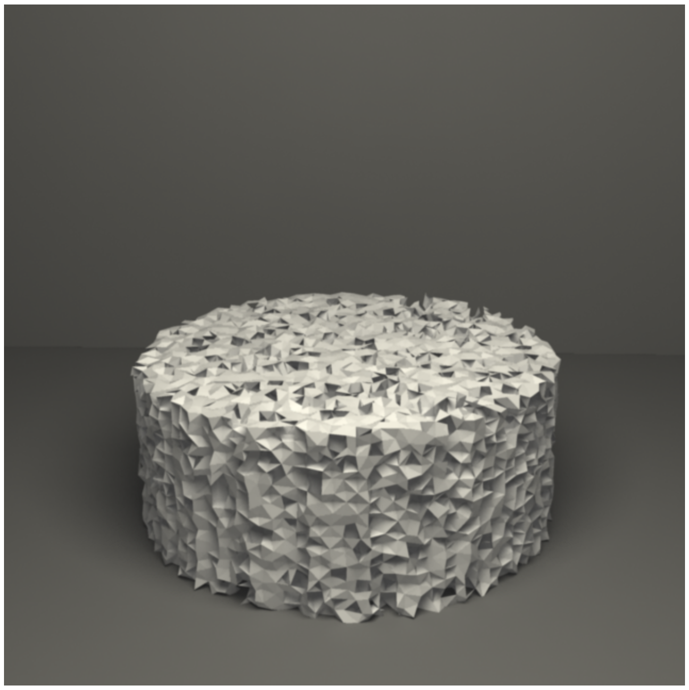

(b) Frame 1700

Figure 3.22: Results of the alpha-shape algorithm from using the same alpha value for the whole animation.

ating the scalar field for the marching cubes algorithm nor the alpha-shapes method for creating the surface mesh resulted in a high quality temporal consistent mesh that could be used for the purpose of creating animations. It is for this reason that the proposed method uses the popular field function attributed to the work by Wyvill et al. [88] to create the scalar field as described in Equation 3.59.

$$\phi(r_\phi) = \begin{cases} a_\phi \left(1 - \frac{4r_\phi^6}{9b_\phi^6} + \frac{17r_\phi^4}{9b_\phi^4} - \frac{22r_\phi^2}{9b_\phi^2}\right), & r_\phi \leq b_\phi \\ 0, & r_\phi > b_\phi \end{cases} \tag{3.59}$$

However, this approach alone does not take into account the issue that arises from the newly proposed phenomenon of volume expansion; as the particles grow apart the fluid's mesh will disappear if the volume of effect of each particle is not dynamically adapted as illustrated in Figure 3.18(b).

To solve this problem, the proposed framework uses the information available from the SPH formalism to modify the particle's volume of effect based on the changes in each particle's density, as described by Equation 3.60:

$$r_{\phi_t} = \alpha_\phi + \left(\beta_\phi \frac{\rho_0}{\rho_i}\right) \tag{3.60}$$

where r_{ϕ_t} is the radius used to define the volume of effect around particle i at frame t, α_ϕ is the base radius when the particle's density is in the relaxed state. From Equation 3.60 it can be seen that the particle's radius of the volume of effect is inversely proportional to the changes in the particle's density ρ, that is, as ρ increases the radius of the volume of effect r_ϕ decreases in length approaching α_ϕ; on the contrary, as ρ decreases, r_ϕ increases in length.

This approach of modifying the volume of effect of each particle allows the surface to grow as the fluid expands through the open areas of the container, focusing on the areas where the particles become more isolated with respect to their neighbouring particles, while maintaining the volume of effect small where the particles are close together (such as in the containers surface where particles are being pushed together avoiding the problem where the fluid's mesh penetrates the con-

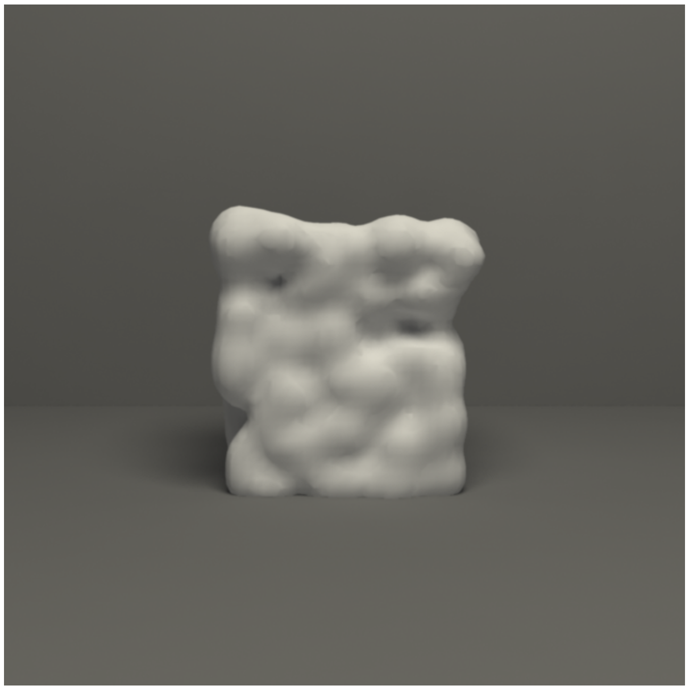


Figure 3.23: Result using a good marching cubes distance function.

tainer). The applicability of this approach is dependent on the particle count of the simulation: that is, if there are too few particles there may not be enough to create a complete shape avoiding levitating fluids. With Figure 3.23 we can compare the results to those obtained with the static field function: starting with the same data set this approach obtains a much better result.

Finally, Figure 3.24 shows the results from applying this approach to an animation with a higher particle count. The shape obtained is not as bulbous as the one shown in Figure 3.19, and it has a rough texture present in many baking goods.

3.6 Summary

In this chapter the implementation of the SPH framework is presented showing how the work by Mao and Yang [51] [52] [53] was expanded to create more flexible and stable phase transitions by adding a viscosity model which allows varying viscosity throughout the fluid, as well as the XSPH velocity smoothing technique

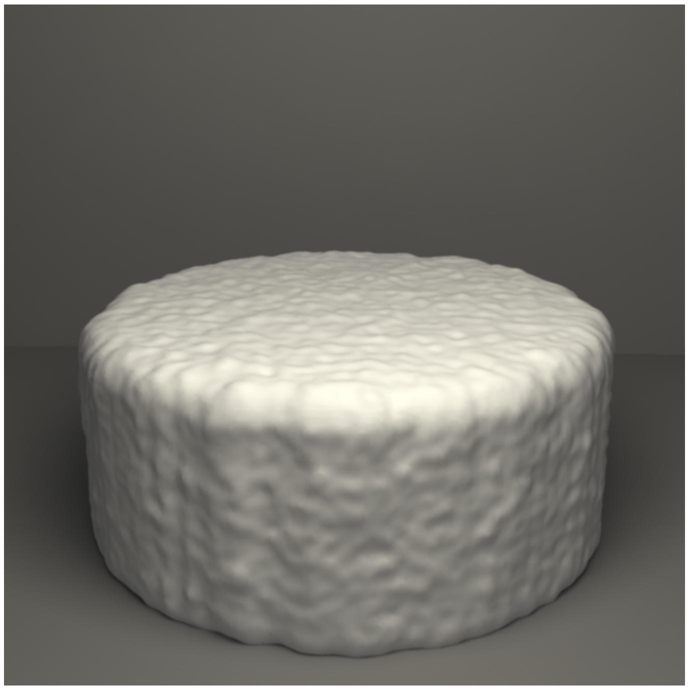


Figure 3.24: Result using a good marching cubes distance function.

which introduces the velocity tuner ϵ parameter that gives greater flexibility to the fluid-solid phase transition. Furthermore, the framework was made more stable by adding artificial viscosity which helps prevent particle inter-penetration.

The SPH framework presented in this thesis can produce quality animations representing thin and highly viscous fluids, fluids that try to maintain their original shape, fluid-solid phase transitions including melting and solidification as well as fluids that expand in volume due to changes in inner pressure.

One of the drawbacks of the SPH framework is that it requires a very large number of particles in order to create quality results. The animations presented in this chapter handle tens of thousands of particles producing results in manageable times by improving the efficiency of the range search function by means of the kd-tree data structure, as well as the overall computational efficiency of the implementation by means of parallel computing using the OpenMP API [2]. However, there is more room for improvement, currently the framework uses the Euler integration method which leads to the need of small time steps; by changing the integration method to the Leap-Frog scheme [46] the time step could be increased which will in turn decrease the amount of time required to generate the animations along with the possibility of using more particles. Furthermore, larger time steps could also allow the use of a wider value range for the simulation parameters while keeping the animations stable.

Chapter 4

Baking model

This chapter contains a description of results obtained from a combination of the phenomena discussed in Chapter 3 to model the baking process by using data available from academic experimental results. Furthermore, this section contains the resulting animations and the data collected during their creation, which was used to perform qualitative comparisons of results presented from other mathematical baking models.

The baking process is quite complex due to the many physical and chemical changes that take place during that process. In order to reproduce the baking phenomena with the goal of creating animations, this framework simplifies the baking process to the following items (described in detail in the next subsections):

- Heat transfer
- Volume expansion
- Solidification
- Surface colouring

Figure 4.1 shows screenshots of a time-lapse recorded by Kolias [37] that show the baking process for an angel food cake. The screenshots illustrate the four items mentioned above that the present framework aims to simulate: the dough is inside a conventional oven, and as Figure 4.1(b) shows, the dough begins to rise as time passes and the temperature increases; gradually, the dough begins to show the characteristics of a solid. Then, the surface of the dough begins to brown as it is subject to heating; finally, as the baking process comes to an end, there is a recession in the size of the dough.

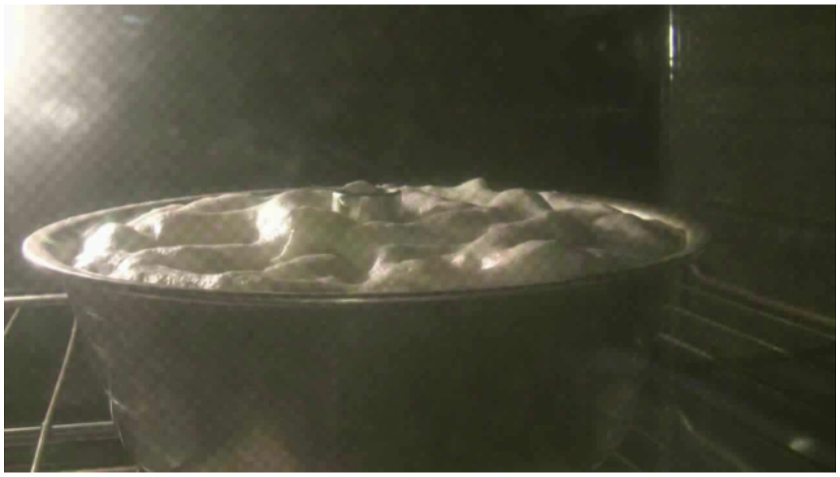
4.1 Heat transfer

As mentioned in Section 3.3, an accurate simulation of heat transfer is important in this framework because it is the driving force behind the phenomena that will come together to animate the baking process.

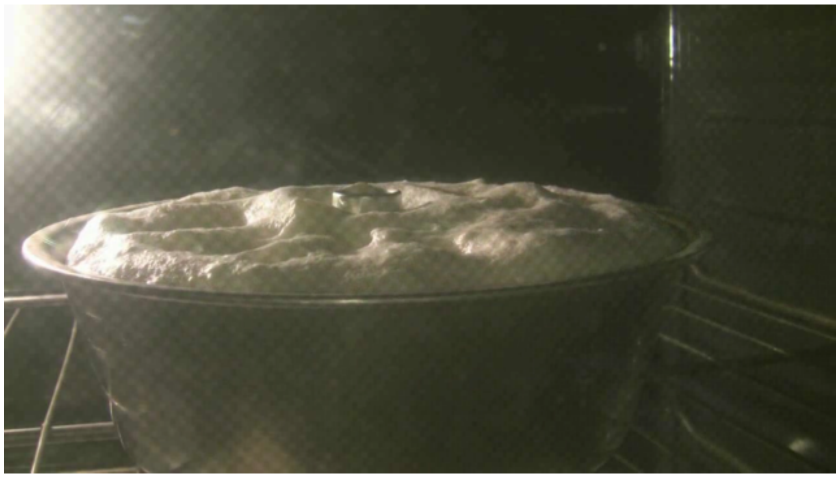
In the same fashion, heat plays a very important role in the different baking models described in Section 2.4. For the sake of simplicity, the authors of these models add different assumptions to their approaches; one of these assumptions is related to the way the dough's temperature is treated, which can affect the outcome of the simulation. For instance, a common assumption made in the mathematical models proposed by Fan et al. [25], Lostie et al. [49], and Marcotte et al. [54] is that temperature is considered uniform throughout the dough's volume.

Although the method proposed by Zhang et al. [91] [92] [90] models heat transfer differently from the approach presented in this thesis (Zhang et al.'s [91] [92] [90] approach takes into account thermal conduction, convection and phase change—evaporation-condensation—while the proposed framework only considers thermal conduction and convection), both approaches consider the naturally occurring temperature gradient that takes place during the the baking process in the crumb section of the dough.

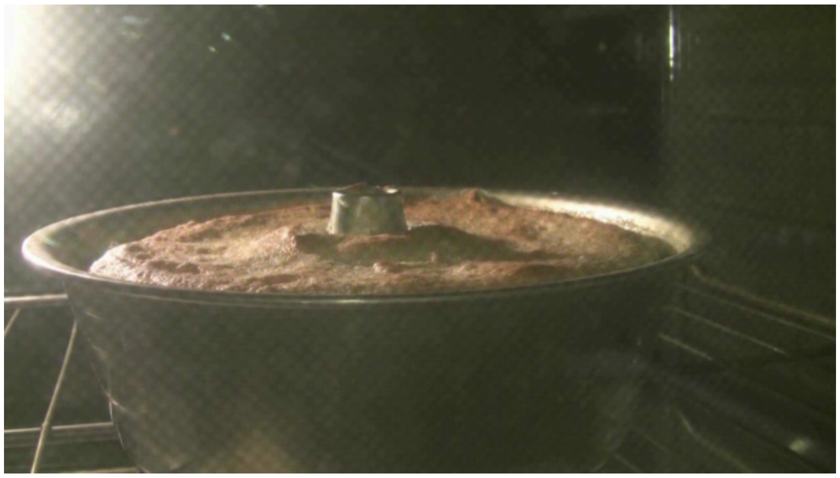
The proposed framework uses the heat transfer method described in Section 3.3.1 to simulate the temperature changes which are the driving force behind the baking process. Figure 4.2 illustrates the temperature changes during a simulation of the dough-baking process with a cylindrical shape container using the developed system. As the graph shows, the results differ from those obtained by Zhang et al. [92] in that there is no difference in the final temperatures at the crumb and crust section



(a)



(b)



(c)

Figure 4.1: Baking time lapse for an angel food cake from [37].

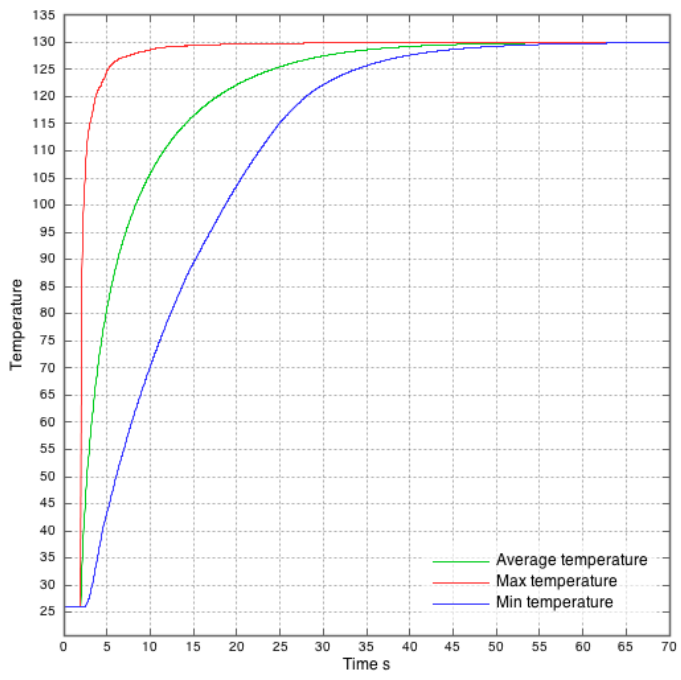


Figure 4.2: Temperature profile of a bread roll during the baking process.

of the baked good. However, this does not affect the final outcome in the animation, and the rates of change of temperature can be considered qualitatively similar to those obtained by Zhang et al. [92] for the different sections of the dough.

It is important to note that even Zhang et al. [92] consider the temperature data obtained in their observations to be of a qualitative nature; this is due to the difficulties in taking accurate measurements inside the dough during the baking process.

4.2 Volume expansion

The dough's final volume is one of the critical attributes used to measure the quality of the baked good, giving it a particular look and overall texture.

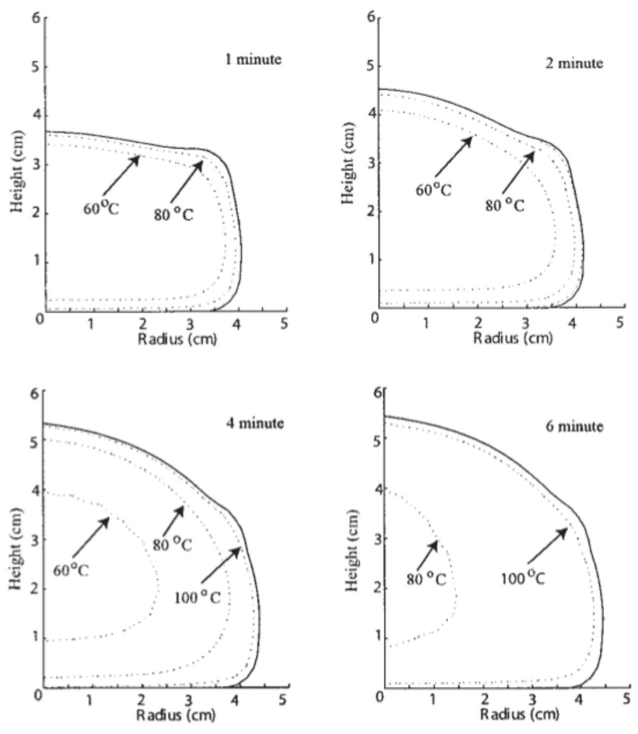
There are different leavening agents that can be used depending on the desired result to foment the dough's increase in volume during the baking process, this increase in volume is known as *oven rise*. Some leavening methods include chemical leaveners, biological leaveners, and mechanical leavening.

Baking powder is a common rise agent used for baking that works by releasing CO_2 inside the dough by means of a chemical reaction either by reacting in the wet dough with baking soda at room temperature, or by being triggered with the increase in temperature. Another commonly used leavener is yeast, a biological leavener classified as a Fungus that produces CO_2 during its life cycle. Finally, an example of mechanical leavening is creating foam by using a whisk tool on a non-Newtonian fluid that can hold bubbles (such as egg whites): this is a common technique when baking sponge cake and angel food cake.

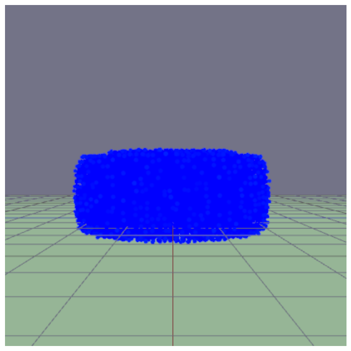
Models geared more towards a realistic simulation generate the final dough expansion and deformation based on CO_2 and water vapour generation; two examples are presented by Fan et al. [25], and Zhang et al. [91] [92] [90]. In order to simplify the proposed model, the bubbles are considered to be already trapped inside the dough as was described in Section 3.3.3: the volume expansion is given by the pressure buildup due to the increase in temperature during the baking process, causing the volume of the bubbles themselves to increase. However, even under this assumption, results obtained show that the proposed framework is capable of producing realistic looking animations by modifying 1) the bubble-to-dough particle ratio, 2) the amount of force the bubbles exert on the dough as they increase in volume, and 3) the bubble particle distribution inside the dough.

Figure 4.3 shows a side by side comparison between the results obtained from Zhang et al.'s [92] mathematical model of bread baking, and the framework proposed in this thesis. A noticeable difference between the two sets of results is that the oven rise obtained from the simulation of this framework is much bigger than that of Zhanget al.'s [92] model. Where they report an increase in relative volume of 170%, the sequence presented on Figure 4.3 shows an increase in relative volume of 350%, which shows that the proposed model can handle very large volume expansions.

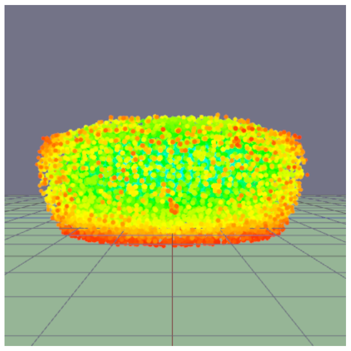
The animation from Figure 4.3 was created using particles in the $20,000$ range, out of which 40% were treated as bubbles distributed evenly throughout the dough.



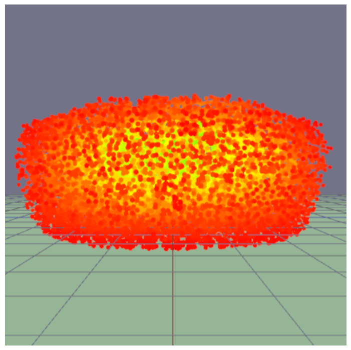
(a) Zhang et al.'s [92] oven rise results.



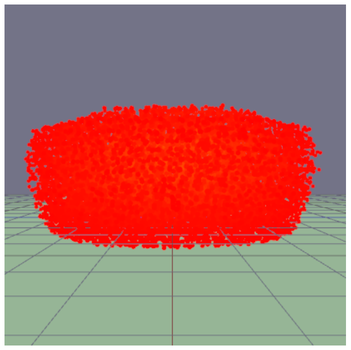
(b) Frame 48



(c) Frame 180

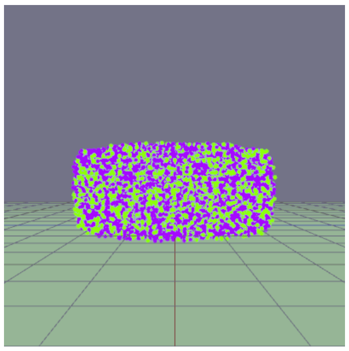


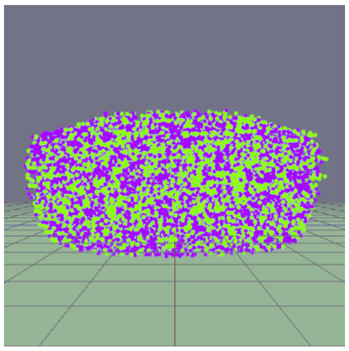
(d) Frame 360

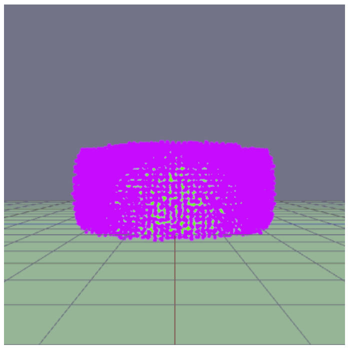


(e) Frame 720

Figure 4.3: Qualitative comparison of volume expansion.


(a) Bubbles evenly distributed, frame 48


(b) Bubbles evenly distributed, frame 940


(c) Bubbles distributed in the center, frame 48

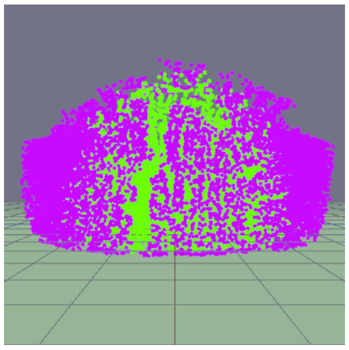

(d) Bubbles distributed in the center, frame 940

Figure 4.4: Results from different bubble particle distribution.

However, it is important to note that as in real life, the distribution of the bubble particles has a big impact on the final result of the baked good. Figure 4.4 illustrates such a difference; here, the frames are from two animations set up in a similar fashion; both of the animations have the dough start with a cylindrical shape, but the initial bubble distribution is set up differently, in order to better visualize the difference the dough particles are coloured in magenta while the bubble particles have a light green tone. It can be seen in Figures 4.4(a) and 4.4(b) that the bubbles are distributed evenly, whereas Figures 4.4(c) and 4.4(d) show the dough with the bubbles distributed at the centre of the dough.

4.3 Solidification

The solidification phenomenon is simulated with the approach described in Section 3.3.2. In order to correctly simulate the mechanical properties of the dough as it bakes, the proposed framework modifies the solidification parameters based on models created from empirical data. However, it is important to note that this data comes from observations conducted on a specific type of dough. In order to make the framework described in this thesis more flexible, the original equations have been modified to give the end user the ability to scale behaviours of rates of change, resulting in more control over the resulting animation.

Changes in dough viscosity are an important factor when simulating the baking process. The proposed framework takes the rate of change for viscosity proposed by Fan et al. [25], in which the viscosity decreases at the beginning of the baking process, but steadily increases after a turning point in temperature is met. This behaviour is reproduced with Equation 4.1:

$$\mu = \begin{cases} \mu_0 + (T - T_0)\frac{\mu_c - \mu_0}{T_c - T_0}, & T \leq T_c \\ \mu_c + (T - T_c)\frac{\mu_f - \mu_c}{T_f - T_c}, & T > T_c \end{cases} \tag{4.1}$$

where T is the current temperature, μ_0, μ_f, and μ_c are the initial, final and turning point viscosity coefficients respectively, and T_0, T_f, and T_c are the initial, final, and turning point temperatures respectively during the baking process.

Although the elasticity of the dough is one of its many properties that changes during baking phenomena, it has largely been ignored in many attempts at modelling the baking process. Zhang [90] mentions that elasticity is another variable that could be added to his proposed model. Due to the lack of experimental data, changes in elasticity in the proposed framework follow the same rate of change given by the viscosity changes described in Equation 4.1, albeit with different range in parameters. The final form of the changes for the elasticity parameter μ_e from Equation 3.20 is defined in Equation 4.2:

$$\mu_e = \begin{cases} \mu_{e0} + (T - T_0)\frac{\mu_{ec}-\mu_{e0}}{T_c-T_0}, & T \leq T_c \\ \mu_{ec} + (T - T_c)\frac{\mu_{ef}-\mu_{ec}}{T_f-T_c}, & T > T_c \end{cases} \tag{4.2}$$

with μ_{e0}, μ_{ef}, and μ_{ec} as the initial, final and turning point elasticity coefficients.

In Zhang et al.'s [91] [92] [90] model, the relaxation time is based on the works of McGee [55], Pan [71], and Steffe [81] (as defined in Equation 2.68). This approach—modified for scaling—takes the form described in Equation 4.3:

$$\lambda = C_\lambda \left[\frac{2}{\pi} \arctan\left(\frac{T - 65}{2}\right) + 1\right] + 2 \tag{4.3}$$

where C_λ is a user defined scaling factor.

As a final adjustment, the velocity tuner ϵ_i is linearly interpolated based on each particle's temperature as illustrated on Equation 4.4:

$$\epsilon_i = \epsilon_0 + (T_i - T_0)\frac{\epsilon_f - \epsilon_0}{T_f - T_0} \tag{4.4}$$

where ϵ_0 and ϵ_f are the initial and final values for the velocity tuner. The velocity tuner makes the particles move in an orderly fashion, and thus, it has a big effect on the result of the baking process simulation when determining whether the dough will become solid. As a result of the tests performed, a value of $\epsilon_f = 0.5$ is suitable to allow for the phase transition to fully take place.

Table 4.1 presents an example of the possible values that the scaling parameters can take in order to create animations of fluids with a fluid-solid phase transition (such as the one illustrated on Figure 4.5). The profiles of these parameters in relation to temperature increase are illustrated on Figure 4.6.

Because the viscosity and elasticity parameters follow a similar rate of change, they can be combined thereby reducing the number of parameters to define the fluid-solid transition phase from 12 to 9.

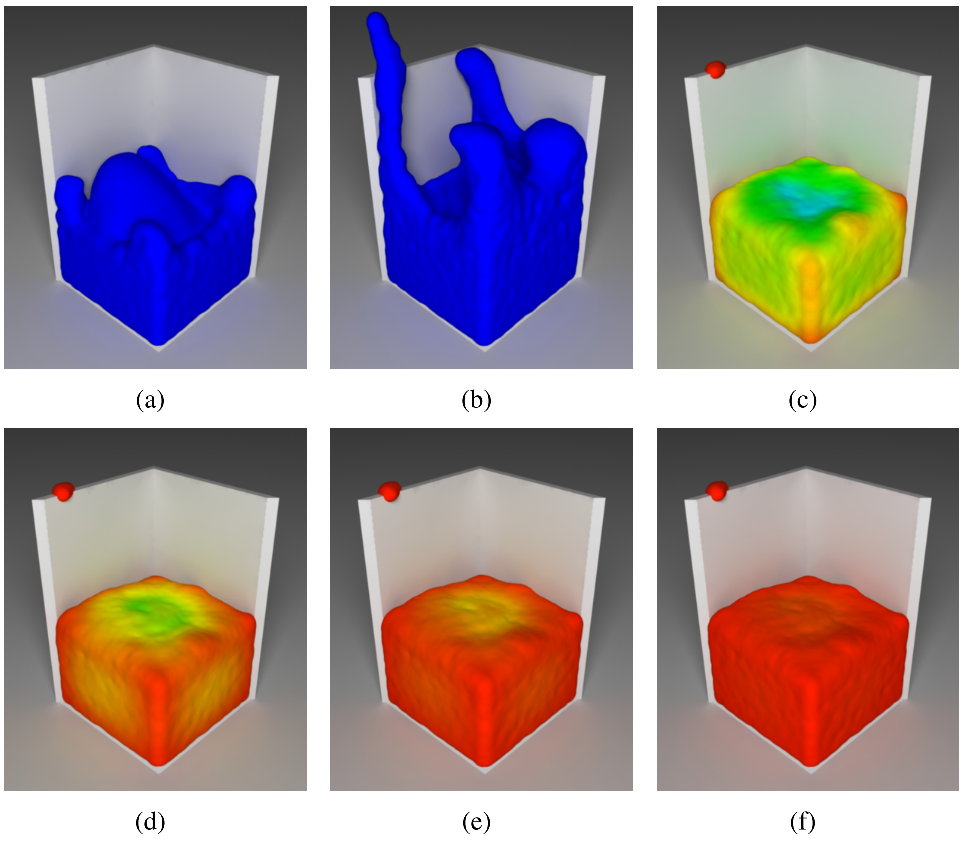
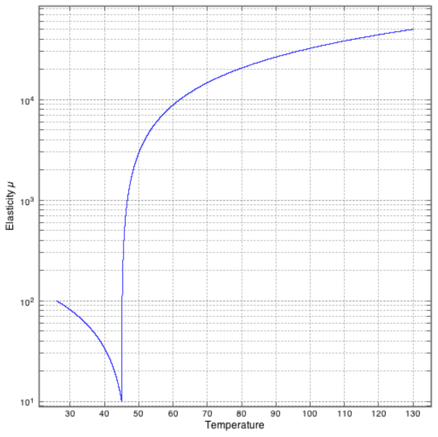


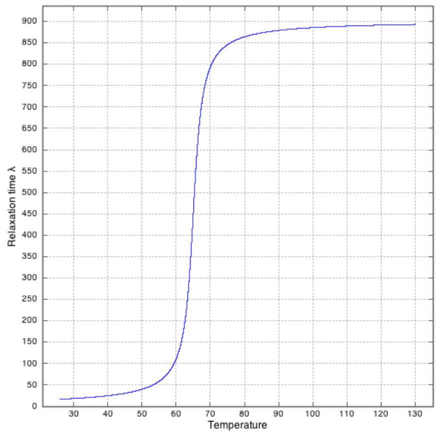
Figure 4.5: Fluid-solid phase transition.

Parameter	**Value**
T_0	26
T_c	45
T_f	120
μ_0	0.1
μ_c	1
μ_f	5×10^3
μ_{e0}	1
μ_{ec}	10
μ_{ef}	5×10^4
C_λ	450
ϵ_0	0.05
ϵ_f	5

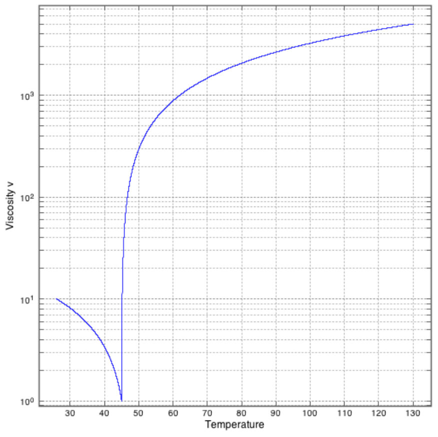
Table 4.1: Solidification parameters.



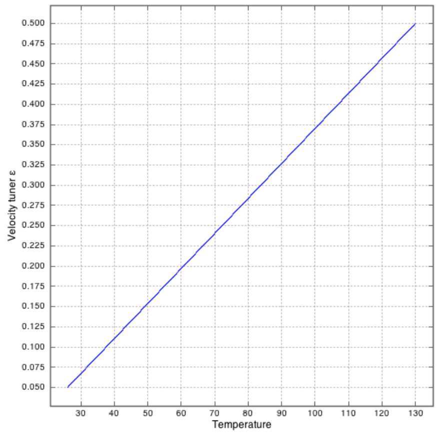
(a) elasticity



(b) relaxation time



(c) viscosity



(d) velocity tuner

Figure 4.6: Temperature profiles of parameters.

4.4 Surface colouring

Simulating the changes in dough surface colour is an important characteristic necessary to achieve a more realistic animation of the baking process. As Zanoni et al. [89] point out, changes in the surface colour depend both on the physicochemical characteristics of raw dough (i.e. water content, pH, reducing sugars, and amino acid content), as well as on the operating conditions applied during baking (i.e. temperature, air speed, relative humidity, and modes of heat transfer).

Zanoni et al. [89] propose a mathematical model to predict the browning kinetics of bread crust during baking. The surface browning is described as the colour difference between the raw dough and a sample that is subject to heating. This colour difference is represented as the Euclidean distance in 3D space with each dimension representing the brightness, redness, and yellow difference between the two samples. The colour difference is considered a function of temperature history, so in order to obtain the surface colour, the transport and deformation stages are modelled. After this, the temperature history is obtained, and the surface is coloured.

The purpose of the framework presented in this thesis is to create animations geared towards visual effects; for this reason, a very important objective is giving the user flexible control over the result of the surface colouring, rather than following realistic browning by colouring the surface in any way desired as the baking process takes place. This flexible level of control is achieved using the SPH formalism along with texture mapping to colour the vertices of the generated surface mesh. The use of textures allows the user to create different results that would be hard or difficult to reproduce in real life.

The surface colouring is made by applying a texture to the created mesh. This texture is a diagonal gradient that contains the range of colours the surface will acquire due to the change in temperature. In order to use this texture, every vertex is assigned a UV coordinate that maps the lowest temperature to the lower-left section of the texture, and the highest temperature to the top-right part of the texture.

To achieve smooth colouring transitions, the U and V coordinates of the texture map for each vertex will have the same value, meaning that they will lie on a diagonal across the UV plane. However, the sets of particles could be treated in an alternative manner by arranging the UV coordinates in different areas of the UV map, thereby creating more interesting effects.

To get the values of the vertex UV coordinates, a range search is performed using the kd-tree data structure at each vertex of the mesh gathering the fluids neighbouring particles. With the information of the neighbouring particles, the UV value pair can now be obtained using Equation 4.5, following the SPH formalism.

$$U_i = \sum_{j=1}^{N} \frac{m_j}{\rho_j} G(x_j) W(\mathbf{x}_i - \mathbf{x}_j, h) \tag{4.5}$$

where U_i is the value that will be used for the UV coordinates at the ith vertex, and $G(\mathbf{x}_j)$ is a function that determines the value that particle j will contribute to the final position of the UV coordinate. Different results can be obtained by modifying the definition of $G(x_j)$. For instance, defining $G(\mathbf{x}_j)$ as

$$G(x_j) = 0.9\left(\frac{T_j - T_0}{T_f - T_0}\right) + 0.05 \tag{4.6}$$

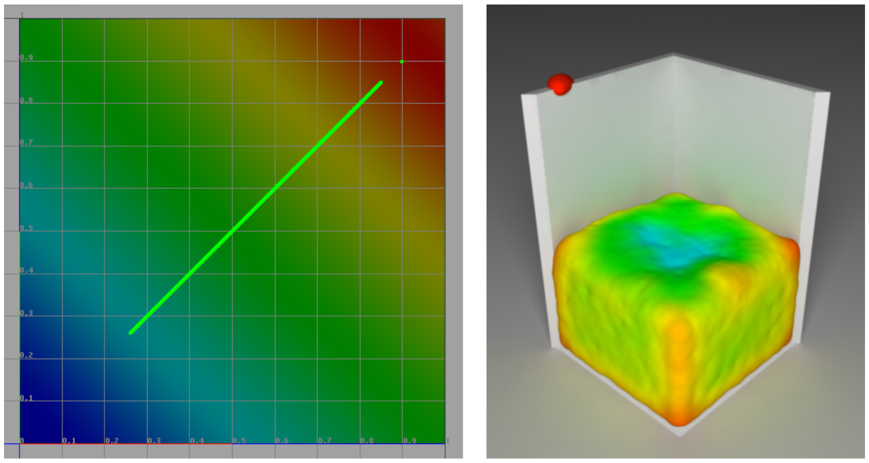
creates a gradient colouring effect dependent on the particle's temperature. On the other hand, if the function $G(x_j)$ is made to differentiate between bubble and dough particles (as defined in Equation 4.7):

$$G(x_j) = O(\mathbf{x}_j) + 0.8\left(\frac{T_j - T_0}{T_f - T_0}\right) + 0.05 \tag{4.7}$$

where

$$O(\mathbf{x}_j) = \begin{cases} 0, & \text{if particle is bubble} \\ 0.1, & \text{if particle is dough} \end{cases}$$

the areas in the fluid with a dense bubble particle distribution—which can be regarded as the crumb of the baked good—can be coloured differently.



(a) UV map (b) Coloured mesh

Figure 4.7: Result from calculating UV coordinates using the SPH formalism.

Figure 4.7(a) highlights in green the UV coordinates generated using SPH formalism placed along the diagonal from bottom left to top right while Figure 4.7(b) shows the result of applying a gradient texture to the mesh using the generated UV map.

4.5 Results

In this section different animations produced with the proposed framework are presented to show the framework's flexibility in simulating the baking process for different and peculiar types of bread. Due to the lack of real-life measurements for the mechanical properties of these types of bread, the values used for the simulations were estimated based on empirical observations with the goal of creating appealing, or dramatic-looking animations.

Bread roll animation

Results presented here show the final renderings of the classic bread roll used in many mathematical models of the baking process. Figures 4.8(c) and 4.8(d) show renderings from different viewpoints of the particle information illustrated in Figure 4.4(b), while Figures 4.8(e) and 4.8(f) are similar renderings of the the particles shown in Figure 4.4(d).

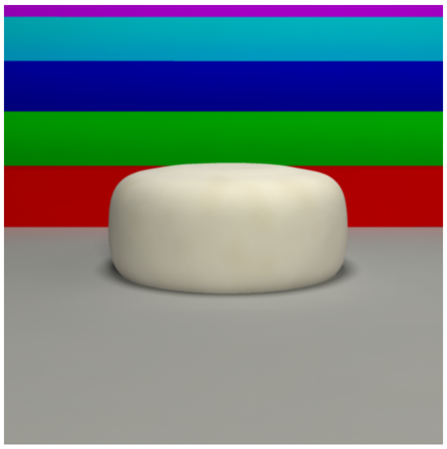
The characteristic colour of the surface was obtained by using the bubble and dough differentiating method described by Equation 4.7. An interesting naturally occurring phenomenon is highlighted in Figure 4.9, where the difference between crust and crumb can be easily observed in the areas where the crumb was torn apart during the baking process. This happens because the bubble particle distribution was not even throughout the dough as illustrated on Figure 4.8(f); when the dough has an even distribution of bubble and dough particles the generated surface will have uniform colouring as illustrated on Figure 4.8(d).

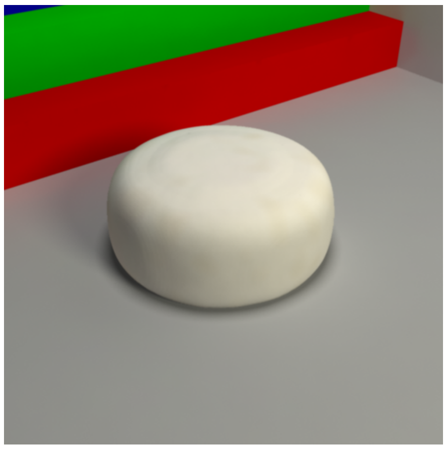
Figure 4.10 shows images of real bread rolls. Their initial shape was not completely cylindrical and thus their resulting shapes are different, however some key features are comparable to the results obtained with the proposed framework. For instance, Figure 4.10(a) shows a baked dinner roll with an uniform volume expansion and surface browning. Figure 4.10(b) on the other hand shows the result from baking no-knead bread which breaks at the top of the crust showing the inner crumb.

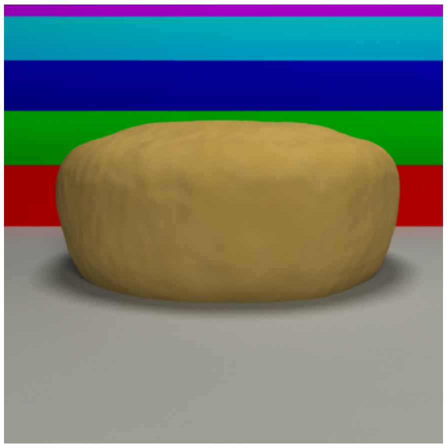
Angel food cake animation

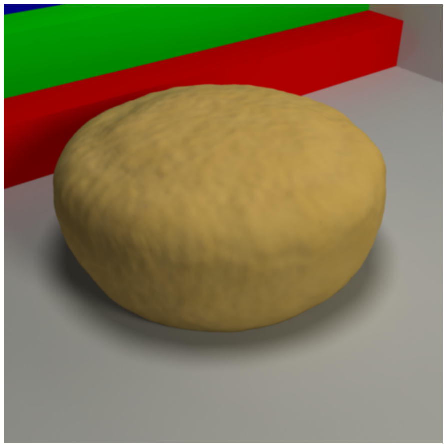
Angel food cake is a type of sponge cake with an peculiar baking method. It requires a special type of container that has a tube structure in its centre. This tube adds more surface area to the dough, making it possible to heat it faster; also, it works as an aid for the cake's oven rise because the dough can cling to it as the oven rise takes place. Another interesting fact about the angel food cake is that after it has been fully baked it must be turned over, otherwise the cake deflates loosing much of its light texture. Figure 4.11 illustrates an animation sequence showing a failed attempt at baking this cake when it is left in its container at the end of the baking process until it collapses.

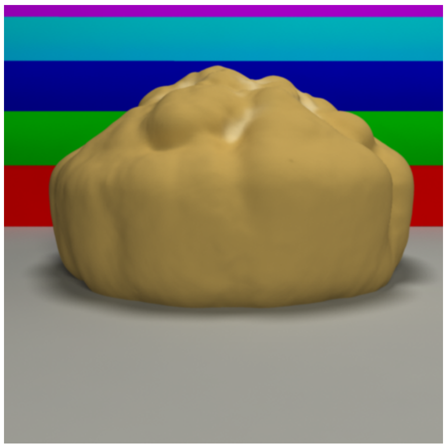
As the animation progresses, the angel food cake is baked but the container is not turned over during its cooling period, causing the dough to loose part of the volume it gained during the oven rise. This animation is generated—as usual—by


(a) Initial dough state, front view


(b) Initial dough state, side view


(c) Result from evenly distributed bubbles


(d) Result from evenly distributed bubbles, side view


(e) Result from bubbles distributed in the centre

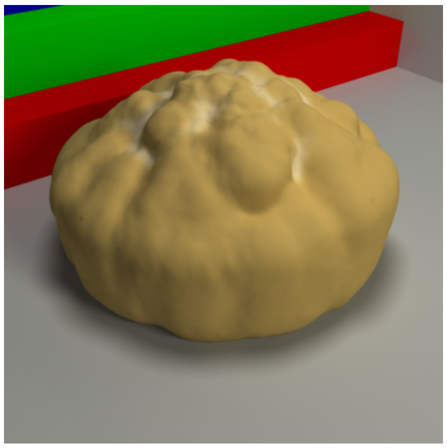

(f) Result from bubbles distributed in the centre, side view

Figure 4.8: Different stages of baking a bread roll.

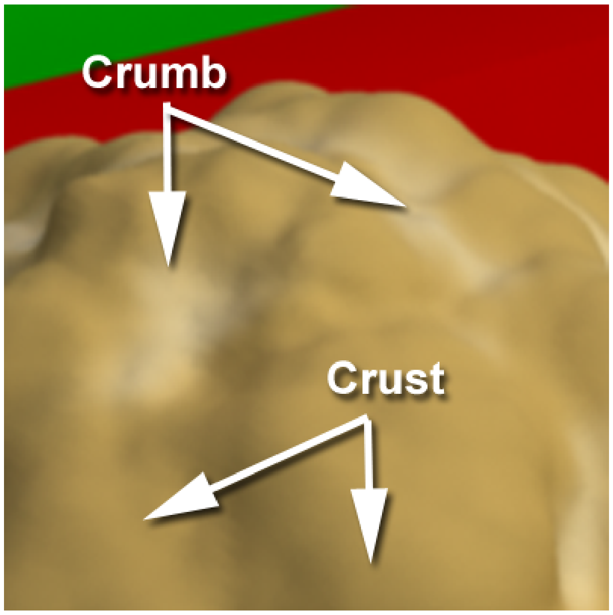


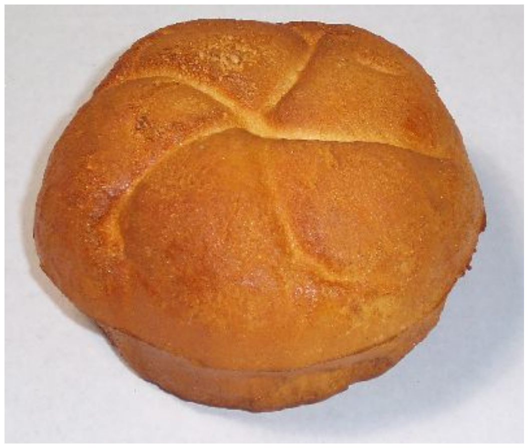
Figure 4.9: Close up highlighting crust and crumb from Figure 4.8(f).

raising the temperature of the particles, and after the cake is completely baked the value for ρ_f in the pressure Equation 3.52 is increased and the elasticity variable μ_e along with the relaxation time μ from Equation 3.20 are reduced to break the bread's structure.

Bread of the dead animation

The "bread of the dead" or *Pan de muerto* is a traditional Mexican bread that is made during the Day of the Dead festivities. The name of the bread is also related to its peculiar shape: a round mass is the bulk of the bread, and at the top, extra dough pieces (shaped in a bone-like fashion) are placed to simulate crossed bones. This is finished with another skull-like dough piece placed in the top centre.

In order for this shape to be held and maintained throughout baking, the dough needs to be prepared in a way that it is smooth and elastic. Figure 4.12 illustrates frames from an animation sequence showing the baking process for this type of bread; it is possible to see that the raw dough holds this shape, and maintains it even when the volume begins to increase as the baking process takes place.


(a) Bread roll from [33]

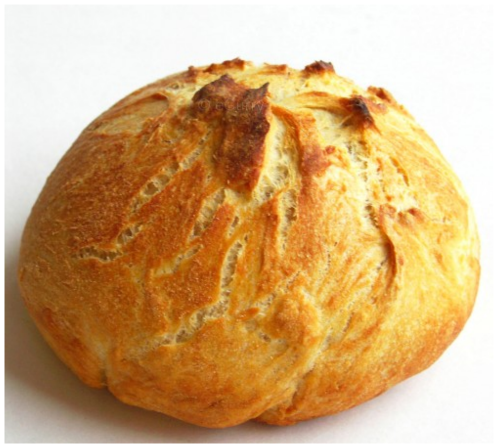

(b) No knead bread from [80]

Figure 4.10: Images from baked bread rolls.

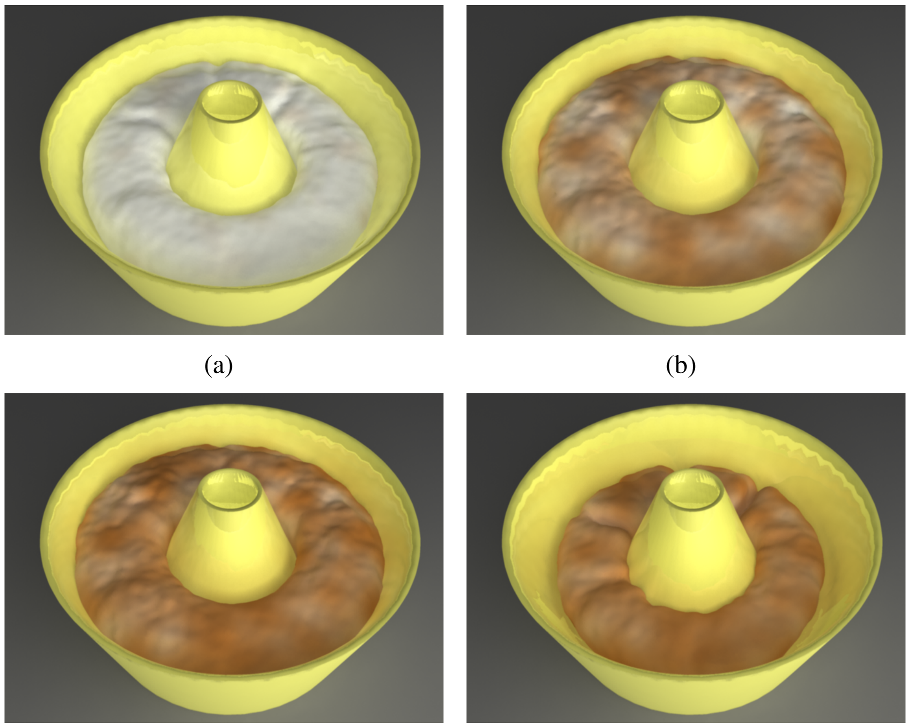
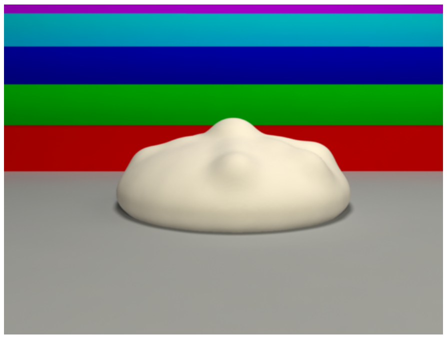
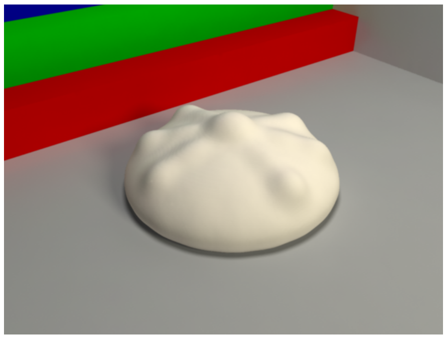
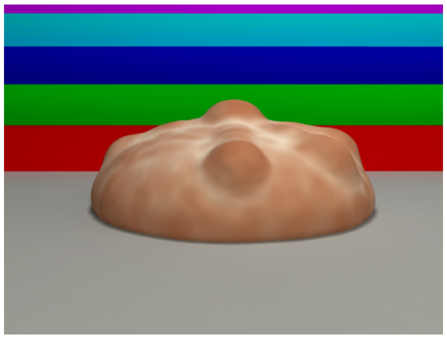
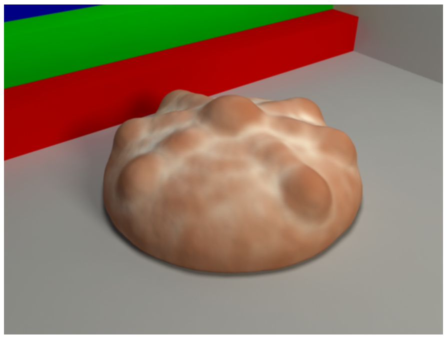

(a) (b) (c) (d)

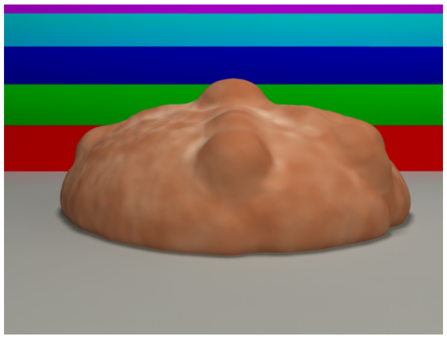
Figure 4.11: Animation sequence baking an angel food cake.

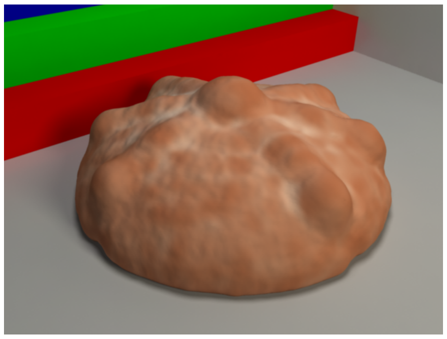

(a) Frame 28

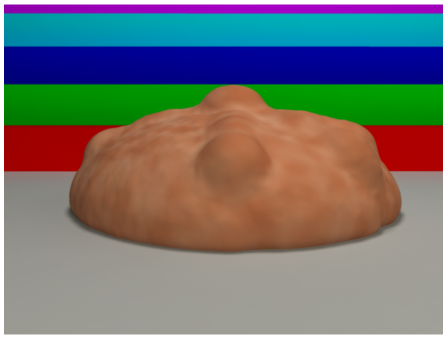

(b) Frame 28


(c) Frame 148


(d) Frame 148


(e) Frame 376


(f) Frame 376


(g) Frame 1290

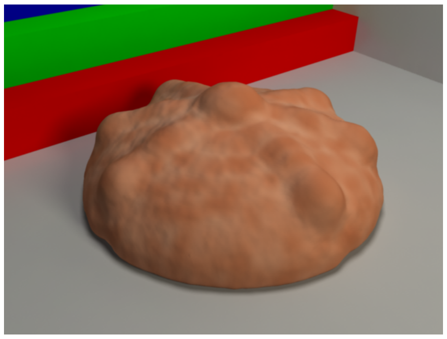
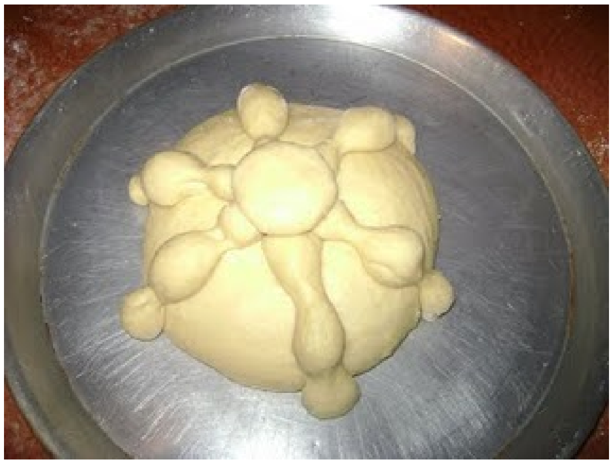
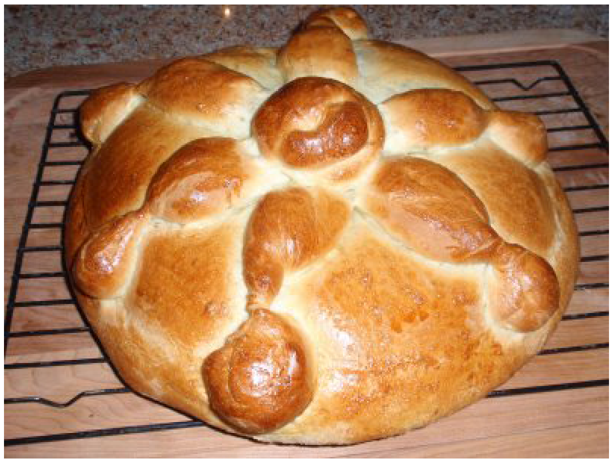

(h) Frame 1290

Figure 4.12: Animation sequence for baking the bread of the dead.



(a) Raw bread of the dead from [44].



(b) Baked bread of the dead from [1].

Figure 4.13: Bread of the dead.

To compare between the animations obtained from the framework an its real-life counterpart Figure 4.13 shows images of real bread of the dead before and after baking.

4.6 Summary

This chapter describes the results of applying SPH formalism to model the baking process. These results show that the proposed framework is capable of reproducing the baking process in a dramatic way, generating interesting animations that remain physically plausible and are interesting to watch.

Furthermore, the results demonstrate that the proposed framework can create animations for a wide array of bread types, by carefully selecting the parameters that control the solidification and volume expansion phenomena in order to create baked goods with different types of deformations. By manipulating the solidification parameters the framework can simulate doughs that can hold their shapes even when raw (like the bread of the dead), or thin batter that easily flows, (such as the one used for pancakes). The oven rise is easily reproduced with the proposed method for volume expansion; moreover, results achieve size increments that would not be possible in real life.

However, with this level of flexibility comes the problem of parameter tuning; as it currently stands, the number of parameters may be overwhelming for creating a specific animation. Also, the solidification and volume expansion phenomena are too tied together with each trying to achieve the opposite thing. This can lead to overestimating a parameter that can, for example, solidify the bread too soon while ignoring completely the contribution of volume expansion. On the contrary, the force applied to expand the volume may be too much for the fluid to handle, causing it to become unstable. The current method for dealing with these problems is by trial-and-error, which is very time consuming. Further studies should be made on how to better handle these two opposing forces, and on developing methods for parameter estimation.

The final touch that gives the resulting animations a realistic look is the colour transitions that take place during the baking period. The method used to colour the surface of the mesh is a simple one, but it can handle the colouring of the dough achieving realistic looking colour transitions as the bread bakes.

Chapter 5

Conclusions and Future Work

In order to simulate the baking process, the work by Mao and Yang [51] [52] [53] was extended in a number of ways. First, the viscosity property is made to be dependent on individual particles' temperature, allowing for greater control over the fluid-solid phase transition. The surface heat transfer is also modified following the method proposed by Stora et al. [82]; after implementing this method, heat transfer takes into account the area represented by each particle at the surface of the fluid. Furthermore, the newly proposed method to generate the volume expansion is developed, which takes into account the changes in volume of particles that are treated as bubbles inside the dough. Such a change is driven by the change in temperature.

Other improvements include a modification to the fluid-solid collision detection and response, which is changed from defining the solids with a limited set of predefined structures to a more general approach. In particular, this new method for detecting and responding to collisions can handle solids of any shape and size by using particles to outline the surface of the solids. Although this is a slower approach, the range of possible animations is much increased.

The running speed of the implementation is decreased by changing the neighbour search function from using the grid-based data structure to the more efficient kd-tree data structure. Also, the numerical stability of the simulation is improved by adding XSPH to correct the particles' velocities, and by adding artificial viscosity to avoid particle inter-penetration.

The resulting animation sequences from the simulation of the baking process prove to be qualitatively similar to their real world counterparts. The dough changes from a fluid to a solid state, while it also gains volume as its temperature rises gradually. Moreover, the method used to colour the surface is capable of creating a nice browning effect that is similar to the one obtained in real baked goods. Furthermore, the results show how the framework is capable of handling exaggerated versions of the baking process such as generating increases in volume. Although these results may look as a desirable product, they are next to impossible to create in the real world.

There are different ways the proposed framework can be further improved or extended to better simulate the baking process. For example, the running time needed to create animations is lowered by adding support for shared-memory parallel programming via the OpenMP API to the framework implementation; however, the running time for the simulations could be shortened further still by using the Graphics Processing Unit (GPU) to perform the computation of heavy sections of the code. Although the implementation of such a feature is not trivial, Harris [31] demonstrated that it is possible to achieve real-time animations with 10 times the number of particles that were used to create the animations presented in this thesis; Harris' work yields animations of much higher quality at an even faster pace. This increase in speed would be beneficial because there would be a faster turnover for animation tests.

Another way to improve the speed of the framework implementation is to simply change the integration method from the Euler approach to the Leap-frog scheme. This would allow for much larger time steps in the animation that, as a result, would decrease the running time of the implementation, and perhaps, make it more stable.

The final look of the rendered bread can be improved by using texture synthesis techniques to render characteristics pertaining to baked goods, such as cracks in the crust as the dough increases in volume, as well as rendering the bubbles themselves as part of the crumb whenever it becomes visible. Also, more advanced techniques could be used to generate the surface mesh; part of the problem was that the generated surface would extend well outside the volume defined by the particles, or it would not fill the containers fully. And although a solution is proposed, there is still much room for improvement: a possible direction of research would be to look into the work by van der Laan et al. [85] that constructs a surface of particles that is as smooth as possible respecting the particle positions. However, the question of how to better handle the expansion of the volume would still need to be looked into with more detail.

Due to the nature of SPH it would be easy to create a more realistic simulation of the baking process by modelling the transport of water; this can be done using the standard SPH formalism. The water content can be stored as a property of each particle, and it could be modified as a function of temperature.

Another way for improving the simulation would be to add more sources for the pressure buildup inside the dough by following Zhang et al.'s [91] [92] [90] model. Pressure from the evaporating water, and the creation of CO_2 could be used to generate a more realistic simulation.

An interesting avenue of research is to look into the possibility of harnessing a recurring problem that took place while simulating volume expansion. When the parameters of solidification and volume expansion—for the particles tagged as bubbles—were not set correctly, there would sometimes be explosions in the fluid due to the quick release of high amounts of force. This, coupled with heat transfer among particles, could result in very attractive effects for simulating explosions coming from specific objects handling cracks in solids, and dispersion in granulous materials.

Bibliography

[1] There is always thyme for... `http://thereisalwaysthymefor.blogspot.com/2008/11/blue-monday-and-weekend-recap.html`, November 2008.

[2] The OpenMP API specification for parallel programming. `http://openmp.org/`, November 2010.

[3] ALEXA, M., BEHR, J., COHEN-OR, D., FLEISHMAN, S., LEVIN, D., AND T. SILVA, C. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics 9*, 1 (2003), 3–15.

[4] BARAFF, D. *An Introduction to Physically Based Modeling: Rigid Body Simulation IUnconstrained Rigid Body Dynamics.* SIGGRAPH, 1997.

[5] BARBER, C. B., DOBKIN, D. P., AND HUHDANPAA, H. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw. 22*, 4 (1996), 469–483.

[6] BECKER, M., AND TESCHNER, M. Weakly compressible sph for free surface flows. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2007), Eurographics Association, pp. 209–217.

[7] BENTLEY, J. L. Multidimensional binary search trees used for associative searching. *Commun. ACM 18*, 9 (1975), 509–517.

[8] BENZ, W. Smooth particle hydrodynamics - a review. In *Numerical Modelling of Nonlinear Stellar Pulsations Problems and Prospects* (1990), J.˜R.˜Buchler, Ed.

[9] BERNARDINI, F., AND BAJAJ, C. L. Sampling and reconstructing manifolds using alpha-shapes. In *In Proc. 9th Canad. Conf. Comput. Geom* (1997).

[10] BLINN, J. F. A generalization of algebraic surface drawing. *ACM Trans. Graph. 1*, 3 (1982), 235–256.

[11] BLOKSMA, A. H., AND NIEMAN, W. The effect of temperature on some rheological properties of wheat flour doughs. *Journal of Texture Studies 6*, 3 (1975), 343–361.

[12] BRIDSON, R. *Fluid Simulation.* A. K. Peters, Ltd., Natick, MA, USA, 2008.

[13] CARLSON, M., MUCHA, P. J., VAN HORN, B. R., AND TURK, G. Melting and flowing. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation* (New York, NY, USA, 2002), ACM Press, pp. 167–174.

[14] CLEARY, P. W., PYO, S. H., PRAKASH, M., AND KOO, B. K. Bubbling and frothing liquids. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers* (New York, NY, USA, 2007), ACM Press.

[15] DA, F., AND YVINEC, M. 3d alpha shapes. `http://www.cgal.org/Manual/latest/doc_html/cgal_manual/Alpha_shapes_3/Chapter_main.html`, June 2010.

[16] DAROY, H. *Les fontaines publiques de la ville de Dijon.*

[17] DESBRUN, M., AND GASCUEL, M. P. Smoothed particles: a new paradigm for animating highly deformable bodies. In *Proceedings of the Eurographics workshop on Computer animation and simulation '96* (New York, NY, USA, 1996), Springer-Verlag New York, Inc., pp. 61–76.

[18] DESOUZAMENDES, P., DUTRA, E., SIFFERT, J., AND NACCACHE, M. Gas displacement of viscoplastic liquids in capillary tubes. *Journal of Non-Newtonian Fluid Mechanics 145*, 1 (August 2007), 30–40.

[19] EDELSBRUNNER, H. Weighted alpha shapes. Tech. rep., Department of Computing Science, University of Illinois, Urbana-Champagne, IL, 1992.

[20] EDELSBRUNNER, H., KIRKPATRICK, D. G., AND SEIDEL, R. On the shape of a set of points in the plane. Tech. rep., University of British Columbia, Vancouver, BC, Canada, Canada, 1981.

[21] EDELSBRUNNER, H., AND MÜCKE, E. P. Three-dimensional alpha shapes. In *VVS '92: Proceedings of the 1992 workshop on Volume visualization* (New York, NY, USA, 1992), ACM, pp. 75–82.

[22] ELLERO, M. Viscoelastic flows studied by smoothed particle dynamics. *Journal of Non-Newtonian Fluid Mechanics 105*, 1 (July 2002), 35–51.

[23] ENRIGHT, D., MARSCHNER, S., AND FEDKIW, R. Animation and rendering of complex water surfaces. *ACM Trans. Graph. 21*, 3 (2002), 736–744.

[24] ERICSON, C. *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology).* Morgan Kaufmann, January 2005.

[25] FAN, J., MITCHELL, J. R., AND BLANSHARD, J. M. V. A model for the oven rise of dough during baking. *Journal of Food Engineering 41*, 2 (August 1999), 69–77.

[26] FISCHER, K. Introduction to alpha shapes. `http://www.inf.ethz.ch/personal/fischerk/pubs/as.pdf`.

[27] FRIEDMAN, J. H., BENTLEY, J. L., AND FINKEL, R. A. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw. 3*, 3 (1977), 209–226.

[28] GINGOLD, R. A., AND MONAGHAN, J. J. Smoothed particle hydrodynamics: Theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society 181* (November 1977), 375–389.

[29] GOKTEKIN, T. G., BARGTEIL, A. W., AND O'BRIEN, J. F. A method for animating viscoelastic fluids. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers* (New York, NY, USA, 2004), ACM, pp. 463–468.

[30] GUENNEBAUD, G., AND GROSS, M. Algebraic point set surfaces. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers* (New York, NY, USA, 2007), ACM, p. 23.

[31] HARRIS, M. Fast fluid dynamics simulation on the gpu. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses* (New York, NY, USA, 2005), ACM Press.

[32] HUBER, P. J., AND RONCHETTI, E. M. *Robust Statistics (Wiley Series in Probability and Statistics)*, 2 ed. Wiley, February 2009.

[33] JONES, M. Easy greens. `http://tinyurl.com/ygq7mar`, February 2010.

[34] KARABASSI, E.-A., PAPAIOANNOU, G., THEOHARIS, T., AND BOEHM, A. Intersection test for collision detection in particle systems. *journal of graphics, gpu, and game tools 4*, 1 (1999), 25–37.

[35] KAZHDAN, M., BOLITHO, M., AND HOPPE, H. Poisson surface reconstruction. In *SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing* (Aire-la-Ville, Switzerland, Switzerland, 2006), Eurographics Association, pp. 61–70.

[36] KEISER, R., ADAMS, B., GASSER, D., BAZZI, P., DUTRE, P., AND GROSS, M. A unified lagrangian approach to solid-fluid animation. pp. 125–148.

[37] KOLIAS, F. Angel food cake - time lapse. `http://vimeo.com/2764927`, January 2009.

[38] KOLLURI, R. Provably good moving least squares. *ACM Trans. Algorithms 4*, 2 (2008), 1–25.

[39] LARKIN, W. K. Applied neuroscience institute. `http://www.appliedneuroscienceinstitute.com/index.php/blog/2010/05/`, May 2010.

[40] LATTANZIO, J. C., MONAGHAN, J. J., PONGRACIC, H., AND SCHWARZ, M. P. Controlling penetration. *SIAM Journal on Scientific and Statistical Computing 7*, 2 (1986), 591–598.

[41] LEFFLER, R. Lafleur de paris: the daily musings of an american in paris. `http://lafleurdeparis.blogspot.com/2009/11/peanut-butter-and-jell-brie-children.html`, November 2009.

[42] LENAERTS, T., ADAMS, B., AND DUTRÉ, P. Porous flow in particle-based fluid simulations. *ACM Trans. Graph. 27* (August 2008), 49:1–49:8.

[43] LEVIN, D. Mesh-independent surface interpolation. *Geometric Modeling for Scientific Visualization* (2003), 37–49.

[44] LIMON, L. La cocina de leslie. `http://www.lacocinadeleslie.com/2009/11/pan-de-muerto.html`, November 2009.

[45] LIU, G. R., AND LIU, M. B. *Smoothed particle hydrodynamics: a meshfree particle method*. World Scientific Publishing, 2003.

[46] LIU, H., AND SHI, P. Meshfree particle method. pp. 289–296 vol.1.

[47] LOOP, C. Smooth subdivision surfaces based on triangles. Department of mathematics, University of Utah, Utah, USA, Aug 1987.

[48] LORENSEN, W. E., AND CLINE, H. E. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1987), ACM, pp. 163–169.

[49] LOSTIE, M., PECZALSKI, R., AND ANDRIEU, J. Lumped model for sponge cake baking during the "crust and crumb" period. *Journal of Food Engineering 65*, 2 (November 2004), 281–286.

[50] LUCY, L. B. A numerical approach to the testing of the fission hypothesis. *Astronomical Journal 82* (December 1977), 1013–1024.

[51] MAO, H. *Physical-Based Non-Newtonian Fluid Animation Using SPH*. PhD thesis, University of Alberta, 2006.

[52] MAO, H., AND YANG, Y.-H. Particle-based non-newtonian fluid animation with heating effects. Tech. rep., Department of Computing Science, University of Alberta, 2005.

[53] MAO, H., AND YANG, Y.-H. Particle-based immiscible fluid-fluid collision. In *GI '06: Proceedings of the 2006 conference on Graphics interface* (Toronto, Ont., Canada, Canada, 2006), Canadian Information Processing Society, pp. 49–55.

[54] MARCOTTE, M., AND CHEN, C. A computer simulation program for cake baking in a continuous industrial oven. In *2004 ASAE Annual Meeting* (2004), ASAE.

[55] MCGEE, H. *On Food and Cooking: The Science and Lore of the Kitchen*. Scribner, November 1984.

[56] MÖLLER, T., AND TRUMBORE, B. Fast, minimum storage ray/triangle intersection. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses* (New York, NY, USA, 2005), ACM, p. 7.

[57] MONAGHAN, J. On the problem of penetration in particle methods. *Journal of Computational Physics 82*, 1 (May 1989), 1–15.

[58] MONAGHAN, J., AND GINGOLD, R. Shock simulation by the particle method sph. *Journal of Computational Physics 52*, 2 (November 1983), 374–389.

[59] MONAGHAN, J., AND J., P. Artificial viscosity for particle methods. *Applied Numerical Mathematics 1*, 3 (May 1985), 187–194.

[60] MONAGHAN, J. J. Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics 30*, 1 (September 1992), 543–574.

[61] MONAGHAN, J. J. Simulating free surface flows with sph. *J. Comput. Phys. 110*, 2 (1994), 399–406.

[62] MONAGHAN, J. J. Smoothed particle hydrodynamics. *Reports on Progress in Physics 68*, 8 (August 2005), 1703–1759.

[63] MOORE, W. J. *Physical Chemistry*, 4 ed. Prentice Hall College Div, 1972.

[64] MÜLLER, M., CHARYPAR, D., AND GROSS, M. Particle-based fluid simulation for interactive applications. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2003), Eurographics Association, pp. 154–159.

[65] MÜLLER, M., KEISER, R., NEALEN, A., PAULY, M., GROSS, M., AND ALEXA, M. Point based animation of elastic, plastic and melting objects. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2004), Eurographics Association, pp. 141–151.

[66] MÜLLER, M., SOLENTHALER, B., KEISER, R., AND GROSS, M. Particle-based fluid-fluid interaction. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation* (New York, NY, USA, 2005), ACM, pp. 237–244.

[67] NAVIER, C. L. M. H. Memoire sur les lois du mouvement des fluides. *Mem. Acad. Sci. Inst. France 6* (1822), 389–440.

[68] OWENS, R. G., AND PHILLIPS, T. N. *Computational Rheology*. World Scientific Publishing Company, July 2002.

[69] OZTIRELI, A. C., GUENNEBAUD, G., AND GROSS, M. Feature preserving point set surfaces based on non-linear kernel regression. *Computer Graphics Forum 28*, 2 (April 2009), 493–501.

[70] PAIVA, A., PETRONETTO, F., LEWINER, T., AND TAVARES, G. Particle-based non-newtonian fluid animation for melting objects. *Computer Graphics and Image Processing, 2006. SIBGRAPI '06. 19th Brazilian Symposium on* (2006), 78–85.

[71] PAN, B., AND CASTELL-PEREZ. Textural and viscoelastic changes of canned biscuit dough during microwave and conventional baking. *Journal of Food Process Engineering 20*, 5 (1997), 383–399.

[72] PREMOŽE, S., TASDIZEN, T., BIGLER, J., LEFOHN, A., AND WHITAKER, R. T. Particle-based simulation of fluids. *Computer Graphics Forum 22*, 3 (2003), 401–410.

[73] Ramesh, N. S., Rasmussen, D. H., and Campbell, G. A. Numerical and experimental studies of bubble growth during the microcellular foaming process. *Polymer Engineering & Science 31*, 23 (1991), 1657–1664.

[74] Roy, T. Physically-based fluid modeling using smoothed particle hydrodynamics. Master's thesis, University of Illinois at Chicago, Chicago, Illinois, 1995.

[75] Sandler, S. I. *Chemical and Engineering Thermodynamics*, 3 ed. Wiley, August 1998.

[76] Shen, C., O'Brien, J. F., and Shewchuk, J. R. Interpolating and approximating implicit surfaces from polygon soup. *ACM Trans. Graph. 23*, 3 (2004), 896–904.

[77] Shepard, D. A two-dimensional interpolation function for irregularly-spaced data. In *ACM '68: Proceedings of the 1968 23rd ACM national conference* (New York, NY, USA, 1968), ACM, pp. 517–524.

[78] Sigalotti, L., Daza, J., and Donoso, A. Modelling free surface flows with smoothed particle hydrodynamics. *Condensed Matter Physics 9*, 2(46) (2006), 359–366.

[79] Solenthaler, B., Schläfli, J., and Pajarola, R. A unified particle model for fluid-solid interactions. *Computer Animation and Virtual Worlds 18*, 1 (February 2007), 69–82.

[80] Soma. ecurry. `http://www.ecurry.com/blog/breads/no-knead-bread/`, April 2009.

[81] Steffe, J. *Rheological methods in food process engineering*. Freeman Press, 1996.

[82] Stora, D., Agliatti, P. O., Cani, M. P., Neyeret, F., and Gascuel, J. D. Animating lava flows. *Proceedings of Graphics Interface 2*, 4 (1999), 203–210.

[83] Takeda, H., Farsiu, S., and Milanfar, P. Kernel regression for image processing and reconstruction. *IEEE Transactions on Image Processing 16* (2007), 349–366.

[84] Thürey, N., Sadlo, F., Schirm, S., Fischer, M. M., and Gross, M. Real-time simulations of bubbles and foam within a shallow water framework. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2007), Eurographics Association, pp. 191–198.

[85] van der Laan, W. J., Green, S., and Sainz, M. Screen space fluid rendering with curvature flow. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2009), I3D '09, ACM, pp. 91–98.

[86] VONNEUMANN, J., AND RICHTMYER, R. D. A Method for the Numerical Calculation of Hydrodynamic Shocks. *Journal of Applied Physics 21* (Mar. 1950), 232–237.

[87] WANG, X., AND SHAO, M. *Principles and numerical methods in finite element method*, second ed. Tsinghua University Press, Beijing, China, 1999.

[88] WYVILL, G., MCPHEETERS, C., AND WYVILL, B. Data structure for soft objects. *The Visual Computer 2*, 4 (August 1986), 227–234.

[89] ZANONI, B., PERI, C., AND BRUNO, D. Modelling of browning kinetics of bread crust during baking. *Lebensmittel-Wissenschaft und-Technologie 28*, 6 (1995), 604–609.

[90] ZHANG, J. *Multiphase heat and mass transfer with large deformation in porous media*. PhD thesis, Cornell University, January 2003.

[91] ZHANG, J., AND DATTA, A. Mathematical modeling of bread baking process. *Journal of Food Engineering 75*, 1 (July 2006), 78–89.

[92] ZHANG, J., DATTA, A. K., AND MUKHERJEE, S. Transport processes and large deformation during baking of bread. *AIChE Journal 51*, 9 (2005), 2569–2580.

Appendix A

Animation information

Name	Reference	Particle count	Simulation time	Animation time	System
Velocity tuner test 1	Figure 3.10(a)	28,072	6h 30m	14s	A
Velocity tuner test 2	Figure 3.10(b)	28,072	4h 10m	14s	A
Velocity tuner test 3	Figure 3.10(c)	28,072	4h 05m	14s	A
Velocity tuner test 4	Figure 3.10(d)	28,072	4h 22m	14s	A
Elasticity test 1	Figures 3.13(a), 3.13(b), and 3.13(c),	8,982	1h 48m	30s	B
Elasticity test 2	Figures 3.13(d), 3.13(e), and 3.13(f)	8,982	1h 46m	30s	B
Elasticity test 3	Figures 3.13(g), 3.13(h), and 3.13(i)	8,982	1h 49m	30s	B
Elasticity test 4	Figures 3.13(j), 3.13(k), and 3.13(l)	8,982	1h 51m	30s	B
Melting bunny	Figure 3.14	14,872	16h 52m	2m	B
Viscosity test 1	Figure 3.15(a)	5,017	28m	15s	B
Viscosity test 2	Figure 3.15(b)	5,017	28m	15s	B
Volcano	Figure 3.17	20,856	9h 33m	59s	B
Fluid container	Figure 4.5	19,958	5h 25m	33s	B

Table A.1: Test animations

System specifications:

System A 2x Dual-Core AMD Opteron @ 2.21 GHz (4 cores in total), 4GB RAM

System B 2x Intel Core i7 CPU 920 @ 2.67GHz (8 cores in total), 6GB RAM

Name	**Reference**	**Dough particle count**	**Bubble particle count**	**Total particle count**	**Simulation time**	**Animation time**	**System**
Bread roll: Uniform bubble distribution	Figures 4.8(c) and 4.8(d)	11,586	8,224	19,810	3h 56m	39s	B
Bread roll: Centred bubble distribution	Figures 4.8(e) and 4.8(f)	11,953	7,857	19,810	3h 47m	39s	B
Angel food cake	Figure 4.11	12,370	8,575	20,945	4h 39m	32s	B
Bread of the dead	Figure 4.12	12,880	8,890	21,770	5h 59m	48s	B

Table A.2: Baking animations