Utrecht University, Game and Media Technology

Virtual Characters and Dynamic Fluid Interactions

Simulation methods for creating convincing bleeding effects in real-time

ing. R. van Oeveren, 3788172

April 2014

Supervised by dr. N.G. Pronost

Acknowledgements

We would like to thank a couple of persons whose input proved invaluable for this experimentation project. Foremost, the continous support, feedback and advice from dr. Nicolas Pronost has helped greatly in developing the methods proposed in this report. Furthermore, the author of the fluid simulation library used in our framework, Jackson Lee, has been very forthcoming with implementation details and code improvements. Last but not least we would like to thank Selma Hilgersom for all her support, and the final revision+editting that has certainly improved the quality of this report.

Contents

1 Introduction

This report describes the experimentation project undertaken by Reinier van Oeveren, as part of the master programme Game and Media Technology at Utrecht University. The focus of the experiments is real-time interaction between virtual characters and physics-based fluids, with an emphasis on simulating bleeding effects. The main goal of the experiments is to determine a suitable framework for the fluid simulations and obtaining the parameters, physical entities (e.g. forces) and interactions needed to achieve specific bleeding effects. The bleeding effects we want to achieve are (deep) cuts and puncture wounds (such as bullet wounds). Simulating convincing and realistic bleeding effects can be useful in many applications, ranging from (serious) games to virtual reality systems and even animated movies. The main criteria for the effects we attempt to achieve is how natural they look compared to their real-life counterparts, while taking into account the (real-time) performance of the methods and techniques behind the effects.

As we show in this report, creating convincing fluid effects poses some challenges that must be overcome. The main difficulties for this project are correctly detecting collisions between the fluid and the virtual character and determining methods that enhance the realistic behavior of the bleeding effects without great loss of performance. To enhance the fluid behavior we propose an *adhesion force method* that aims at reproducing the tendency of fluids to stick to the virtual character. Additionally, we propose several methods that increase the natural appearance of the bleeding effects based on the type of wound that is being simulated. A small portion of this report is dedicated to fluid rendering, although this is not the main focus of the project.

2 Background

Computational fluid dynamics is a well-known research topic within the field of computer graphics. The performance of computer hardware has greatly increased over the last decade. Especially with the introduction of graphics processing units (GPUs) and the subsequent increase in computational capabilities, more recently developed fluid simulation methods have taken a vast leap forwards in terms of realistic visual representations compared to older methods. Fluid simulations performed in computer graphics range heavily in complexity, from relatively simple animated particle systems to complex physics-based models. For our experiments we will use the same physics-based fluid simulation method that was also used in the recently published paper by Xu et al. [1]. Their work focuses on simulating fluid behaviour that is driven by character motion. We use ideas and techniques from this paper as a reference for our experiments with the aim to develop new methods that build upon the work in the reference paper. The main goal is to incorporate actual collisions between the fluid and virtual characters and to experiment with methods that improve the visual quality of the fluid behaviour as it interacts with the virtual character. This chapter provides the necessary background information on the fluid simulation technique and methods proposed in the reference paper.

2.1 Smoothed Particle Hydrodynamics

Traditional particle systems are mainly used in real-time applications, since they only consider approximate and independent particle motion. Traditional particle systems involve substantially less computational costs compared to physics-based particle systems. For this experimentation project we have chosen to implement *smoothed particle hydrodynamics* (SPH), a physics-based fluid simulation method that is slowly becoming more widely used in real-time applications such as games. Smoothed particle hydrodynamics is a mesh-free Lagrangian method where the fluid consists of a set of discrete elements that are commonly called particles. Dating back to 1977, smoothed particle hydrodynamics was originally developed as a probabilistic particle method by Gingold and Monaghan [2] but also independently by Lucy [3]. Originally, its intended usage was simulating astrophysical problems. Nowadays it is applied to many research problems involving incompressible fluid flows. Limited use of smoothed particle hydrodynamics in real-time applications – even until now – can be accounted to its increased complexity in both implementation and computation. Especially the increased computational costs are often considered a significant bottleneck in real-time applications like games and virtual reality systems where performance is a decisive factor, making SPH a less popular choice.

Implementations of Lagrangian fluids can either be mesh-based or particle-based, while Eulerian methods always correspond to mesh-based solvers. The simulation domain for Eulerian methods is discretized by a fixed mesh, meaning that the fluid properties are only calculated at discrete grid points of the domain. In the Lagrangian view, the fluid is described as a discrete number of particles that 'carry' the fluid quantities. Each particle is specified by a state list of properties, namely position, velocity, mass, force, pressure and density. The particles are able to move freely through the entire domain, although they remain dependent on the acting forces in the simulation scenario at all times. Particles have a spatial distance in the SPH-method, also called the *smoothing length,* over which the particle properties are smoothed out by a kernel function. By summing the relevant properties of all particles that lie in the range of the kernel, the physical properties of any particle can be obtained. The performance of particle-based solvers highly depends on the efficiency of implemented computational algorithms and the actual number of simulated fluid particles.

2.2 Reference work

The work done for this experimentation project is based on the research conducted by Tianchen Xu, Wen Wu, and Enhua Wu who published their work in the research paper 'Real-time generation of smoothed-particle hydrodynamics-based special effects in character animation' [1]. In this paper, the authors present an approach to generate real-time smoothed particle hydrodynamics fluid flows that are driven by character motion. Their novel method involves the constraint of the fluid particles based on the geometric properties of the character motion trajectory. To achieve their goals, the authors developed methods for efficient placement of particle emitters and tracking of character motion to predict particle velocities. In the following subsections we elaborate on the main techniques and methods that we base our experimentation project on. The authors also dedicate a large part of their work to the rendering of the fluid. In subsection 2.2.3 we will give a brief introduction into the proposed rendering methods that we partly implement in our experiments. The main focus however remains on methods that are relevant to the fluid behavior itself.

2.2.1 Particle Emitters

During the simulations, particle emitters are responsible for the emittance of new or existing particles. Particle emitters serve as the starting position of a new particle that is emitted into the simulation scene. Positioning the particle emitters can be handled in different ways. The simplest approach for emitter distribution is appointing the mesh vertices as the positions for the emitters. However, this approach is not ideal as vertex layouts are often non-uniform, causing uncontrollable emission states. In the previous work by Xu et al. [4], the vertices of the skinned character are unwrapped in to the UV space and stored in a position buffer in order to track the emitters. The main drawback to this approach is that different UV atlases may be generated in varying geometric scales for different subparts of the mesh, leading to various emitter densities. In Xu et al. [1] the proposed method of positioning particle emitters is based on per-triangle sampling to obtain the correct distribution that uniformly divides the set of emitters over the entire virtual character mesh, or subparts of the character mesh (such as arms or legs). Each mesh triangle $\triangle ABC$ is aligned with the pair of orthonormal tangent vectors that are constructed along the principal directions of the triangle centroid. By parametrizing the vertices A, B and C into the tangent-aligned space (u, v), the triangle can be point-sampled at a customized resolution. The sampled points are mapped into the normalized location space (α, β) of each triangle domain. Sampled points whose calculated domain location values $\alpha \text{ and } \beta$ lie within the range of $[0, 1]$ are regarded as valid emitter positions. Using this technique, combined with bilinear interpolation, emitter positions become fast and easily updated. For details and formulas, please refer to section 3.1 of Xu et al. [1].

2.2.2 Particles and Motion Tracking

As mentioned earlier, particles are discrete elements that define the mesh-free simulation fluid. Particles are bound to their lifetime parameter, which indicates for how long the particle has existed in the simulation. Upon exceeding the particle's maximum lifetime, the particle is typically removed from the fluid. However, instead of removing and adding particles throughout the simulation it is more efficient to relocate 'expired' particles to the position of a randomly selected existing emitter, thus handling both the addition and removal of individual particles in one single operation. In the case of an animated mesh, particles that are added or relocated are assigned an initial velocity based on the trajectory and motion of the designated emitter.

The reference paper uses a probabilistic method to estimate the new particle velocities based on the motion and corresponding positions of the vertices of the character mesh. These vertices are tracked and recorded over the current frame at time t and three historical frames at $(t - \Delta t)$, $(t - 2\Delta t)$ and $(t - 3\Delta t)$. The motion of the emitter trajectory is then calculated based on the interpolated vertex positions that determine the emitter positions at each time frame. The subsequently derived emitter trajectory curve is then used to calculate the emission velocity of the spawning particle. The magnitude is obtained from the length of the trajectory segment during interval Δt, while its direction is based on the tangent vector of the trajectory curve.

2.2.3 Fluid Rendering

A large portion of the work proposed in Xu et al. [1] focuses on the visual representation of the fluid, in other words how the fluid is rendered onto the screen. The technique used to render the fluid is fairly basic and revolves around the well-known method of billboarding, which is essentially the use of 2D sprites in a 3D environment such that the sprite is always facing the camera. Most 3D engines are able to render sprites faster than conventional 3D objects, which results in a substantial performance advantage – especially when there is a large amount of particles in the simulation that require rendering. Additionally, the work in Xu et al. [1] extensively describes how to optimize the fluid rendering through the use of GPU buffers in order to achieve optimal real-time rendering performance. This is not covered in the scope of this experimentation project.

3 Experiment Setup

This chapter gives a detailed description of the implementation details of the simulation framework we have created to conduct our experiments. The implemented methods and techniques differ - in several areas - from the reference paper. In section 3.1 we describe the experiments in general and the main differences from the paper methods. The altered implementations are generally better suited for conducting our bleeding effect experiments in such a way that the desired experimental results can be achieved. Occasionally, techniques mentioned in the reference paper involve highly optimized implementations that do not fall within the scope of our project. Instead, we have implemented simplified or adapted techniques to achieve similar effects while preventing the need for the development and implementation of time-consuming methods. The setup of the fluid simulation is subsequently described in detail in section 3.2., including the pipeline, workflow and important simulation variables. The description of the pipeline and workflow includes more technical background concerning the simulation framework and its internal workings. The simulation variables are specific parameters that influence the fluid simulation during runtime. These variables are divided into two categories; static variables that remain constant for all experiments and non-static variables that may vary for each individual experiment, depending on achieving the desired effect. The general description and the function of these variables are given in this chapter, the actual values and effects are described in chapter 4.

3.1 General Description

The techniques and methods used to achieve desired fluid effects are based on the assumption that, during simulation runtime, the virtual character will emit fluid from its surface. Examples of this type of fluid emission are bleeding and sweating, however the effect can be generalized to many forms of fluid emission. Placing the particle emitters over the entirety of the character mesh surface can make the character appear to be consisting of a certain fluid such as mud or slime. By limiting the placement of emitters to certain subparts of the character mesh, it is possible to achieve specific effects. For instance, to simulate a crying effect, the particle emitters should generally be placed near the eyes of the virtual character. An other example is bleeding effects that can be accomplished by placing the particle emitters in a straight line, defining a deep or superficial cut. There are two main distinct factors that will have the largest impact on achieving certain fluid effects: the properties of the material(s) that comprise the virtual character and the properties of the fluid itself. With regard to the material properties, there are several variables that are important, such as friction and energy restitution. The fluid-related properties range from well-known common parameters such as compressibility and viscosity to more intricate fluid-related traits such as adhesion, which can be seen as the tendency of dissimilar materials to stick to one another. The adhesion effect between particles and the character surface plays an important role in most of the fluid effect methods implemented in the experiments. The exact simulation variables that involve material properties, as well as full details and descriptions are described further in subsection 3.3.2.

3.2 Implementaton Details

This section describes important implementation details of the different aspects and methods that form the basis of the experiments. Some of the implemented methods are closely related to techniques described in the reference paper, such as particle spawning. Other methods differ substantially from their counterparts in terms of technique, implementation or both. Methods we developed for our experiments that involve collision detection and additional fluid forces - such as adhesion - function as extensions on the original paper and aim to improve the realism of the fluid effects by mimicking physical properties that exist in the real world.

3.2.1 Implementation Framework

The framework that we use for our experiments is built upon the existing framework called RAGE that is developed specifically for Utrecht University. RAGE makes use of a mix of the programming languages C++ and Python. The rendering is done using OGRE [5], an open-source 3D engine that is freely available on the internet. For the physics simulations we make use of the Bullet Physics Library [6] in conjunction with Bullet-FLUIDS [7], an independently developed library aimed at the simulation of SPH fluids.

3.2.2 Virtual Character Mesh

For the virtual characters we use during our experiments, we use models provided by the OGRE engine. These models consist of triangle meshes that can be animated by loading animation routines. We can discard any attached animations so that we have static meshes at our disposal. For simplification purposes we only use one single virtual character per scene for each experiment. The virtual character is animated using a skeleton-based approach where individual bones determine the animation pose. The visual representations of all used virtual characters consist of triangle meshes, which are very commonly used in computer graphics. Using triangle meshes allows us to retrieve the shared vertex data during each frame of the (skeletal-based) animation of the virtual characters. The mesh vertices are used as reference points needed for storing initial particle emitter positions during the initialization phase and tracking emitter positions during runtime based on the current animation pose. Based on the tracked emitter positions, we calculate and apply initial velocities to particles that spawn at emitters that are linked to animated characters.

3.2.3 Collision Shapes

Without any interference, the fluid particles will initially be able to penetrate the surface of the virtual character. In order to be able to correctly apply our fluid methods, such as the adhesion force, we need to introduce collision detection and resolution to our virtual character and fluid particles. A common approach in computer graphics is to represent the virtual character using primitive collision shapes such as spheres and boxes for the head, arms and legs. The reason behind this can be found in performance of the collision detection that is much more efficient when performed on primitive shapes. For our experiments we need a highly detailed collision shape that approaches the original character mesh as close as possible. To achieve our goal we have implemented the use of bounding volume hierarchy triangle meshes shapes as collision shape for the character, which is a built-in functionality of the Bullet Physics Library. Triangle mesh shapes take the character triangle mesh vertices as input to create a collision shape that is (nearly) identical in shape to the original mesh, with added support of an accelerated search structure. The triangle mesh collision shape can be generated at initialization of the framework and serialized to disk for future access, but this is only relevant for static meshes. During the character animation the shape of the character mesh potentially changes, which requires rebuilding the collision shape at each frame of the animation. The collision shape update process can act as a severe performance bottleneck as we explain in subsection 4.2. Because collision shapes are responsible for the physics interactions in our simulation, some parameters that deal with material properties (such as friction and energy restitution) are applied directly onto the collision shape instead of the character mesh.

3.2.4 Emitter Placement

In contrary to the uniform particle emitter placement proposed in the reference paper, our framework requires manual placement of particle emitters on the character mesh. This approach ensures the ability to experiment with various different scenarios in which the specific placement of emitters is crucial. The focus of our experiments mainly lies on bleeding effects, which in turn are generally dependent on the type of injury that is being reproduced. Each injury type displays different fluid behavior and may require a specific tactic in terms of emitter placement. For example, placing the particle emitters in a jagged line may indicate a bleeding cut or tear, while centralizing emitters around one point can be used to simulate a puncture in the virtual character which may have been inflicted by a foreign object such as a bullet. We focus on two types of injuries for our experiments: (deep) cuts and punctures (such as bullet wounds). The emitter placement tactics used for these specific injuries can be generalized towards visualizing other types of fluid effects, such as crying or sweating. As mentioned, in the current framework the emitter placement depends on user input. Emitters can be added throughout the simulation at any given location on the character mesh. This approach allows for direct control over the location of emitters and is most adequate and suitable for our specific experiments as opposed to automated emitter placement. However, to achieve various predefined fluid effects, developing an automated emitter placement routine might prove useful in the future.

Spawning particles from emitters that reside inside the surface mesh would cause initial collisions between the character mesh and the particles. This negatively affects the results due to the subsequent collision resolving that applies an initial velocity to particles. To prevent this from happening, the particle emitters are placed very close to the surface mesh. When the user picks a new emitter location, the closest surface triangle face is determined using ray-casting. Using the three vertices that define the picked triangle face, the normal vector perpendicular to the triangle face is calculated. For the next step, the emitter is placed along the determined

normal vector, setting the minimal distance between the surface and the emitter to the particle radius. In certain scenarios it may be desirable to supply an additional defined offset to increase the distance between the emitter position and the surface. This is especially useful when dealing with animated character meshes. The default collision detection algorithms of Bullet Physics is not very suitable for detecting collisions between moving triangle mesh collision shapes and other (primitive) rigid bodies such as the SPH-particles. The reason for this is that the collision detection is generally performed on triangle vertices or edges due to performance reasons. Since the particle size in our simulations is generally very small compared to the dimensions of the character mesh, particles often do not collide with the triangle vertices and edges, leading to undetected collisions. To remedy this, we can increase the particle size or reduce the scale of the character mesh. This is highly undesirable when producing realistic fluid effects that interact with virtual characters, in which case particles are significantly smaller than the mesh itself. By specifying an additional emitter offset, the particles spawn further away from the mesh which increases the efficiency of the triangle-particle collision detection. The increased distance between the spawning particles and the mesh surface is hardly noticeable during animations, making it a viable solution for our fluid effects that heavily rely on effective collision detection.

For storing the initial positions of particle emitters and tracking updated emitter positions we make use of the *barycentric coordinate system*. The exact position of an emitter on a mesh triangle can be written as the weighted sum of the three vertices that construct the mesh triangle (which was manually selected by the user). For each emitter we store its three vertex indices and the three weights corresponding to each vertex. At any given moment during the character animation we can calculate the current emitter position by using the current vertex locations and the specified weights. After determining the updated surface normal, all that remains is applying the predefined emitter offset.

3.2.5 Fluid Effect Strategies

Creating natural looking bleeding effects depends on the placement of the particle emitters as well as the time interval used to spawn particles. In our experiments we explore the possibilities to create two types of injuries (cuts and punctures) by varying the emitter placement and particle spawning tactics. For the spawning tactics we explore the option of using uniform and normal distributions to achieve gushing and continuous bleeding effects. By combining both emitter placement tactics and spawning tactics we can create a set of bleeding effects, such as gushing bullet wounds and steadily bleeding (deep) cuts that bleed heavily or slowly. For details on the proposed methods and exact values, please refer to chapter 4.

3.2.6 Fluid Rendering

The fluid rendering methods are not the focus of this experimentation project. That said, the results we want to achieve with the experiments are primarily judged by their visual appearance (e.g. the effects must look convincing) and having a form of fluid rendering is a welcome - perhaps even mandatory - addition. In the field of computer graphics there is still a lot of ongoing research regarding the area of realistic fluid rendering. There are various approaches that describe how a fluid can be visualized adequately. Some examples of popular fluid rendering techniques are *dynamic billboarding* and more recently *screen space rendering*.

In our experiments we make use of two rendering methods. The first method uses a primitive sphere mesh where each particle in the fluid is represented by one rendered sphere. This method is substantially slower than using sprites, however it is very suitable for our experiments to visualize and assess the particle behaviour. The loss of performance due to rendering the 3D objects can be discarded for most of our fluid effects due to the limited amount of used particles. An added advantage of rendering the particles using meshes is that we can use basic shading to represent the type of fluid, such as glistening blood, which adds to the viewer immersion. The second rendering method resembles that of the reference paper, where we employ billboarding of sprites. This method is especially useful when dealing with large amounts of particles. Due to the scope of our project we disregard the extensive work described in Xu et al. [1] that deals with rigorous optimization of fluid rendering via GPU buffers which aims at achieving the highest performance possible.

3.3 Simulation Setup

To experiment with the fluid and the subsequent bleeding effects, we have built a simulation framework that utilizes both Bullet Physics [6] and Bullet-FLUIDS [7]. For better understanding of our custom-built framework we give a synopsis of the pipeline and workflow used for our experiments, as well as a compact overview of the implemented SPH-method from Bullet-FLUIDS. Bullet-FLUIDS supplies an extensive set of variables that can be adjusted, however we discuss only the variables that are important to our experiments in section 3.3.2.

3.3.1 Pipeline and workflow

Initialization Upon initializing the framework, we do most of the setup regarding our fluid simulations. During this stage we initialize the physics world for the SPH fluid simulation and set all the fluid parameters, which cannot be changed during runtime. For rendering purposes we also load the Ogre rendering engine and setup the simulation scene consisting of a ground plane and directional lighting, with optionally a skybox. Additionaly we load the correct character mesh and set its appropriate scaling, rotation and translation as needed for the experiments. The last step involves building and setting the initial collision shape for our character mesh, which does not change during runtime if the character remains static.

Runtime Most calculations happen during runtime when the fluid is being simulated. If applicable the character animation and emitter positions are updated during each loop of the simulation. Each physics update there are three additional internal stages in which we can perform calculations and updates. These stages are referred to by Bullet-FLUIDS as *pretick*, *midtick* and *posttick* callback functions. As is already clear from the naming convention, the pretick happens before a physics update, the midtick during the physics update and the posttick happens after the physics update. In our simulation we only require the use of the midtick and posttick. During the midtick the contacts between the particles and the character mesh are updated. This is needed to have the most accurate contact points. During the posttick the SPH fluid has been completely updated for the current simulation loop, after which we calculate all the additional forces, such as adhesion, which will then be applied to the particles (as needed) in the next physics update. After each physics update we also update the billboard (or mesh) positions of our particles based on the current position of the SPH-fluid particles. Last but not least we update the current lifetime of all the particles that are present in the simulation and spawn particles at their emitter when their maximum lifetime is exceeded.

The internal update loop for the Bullet-FLUIDS SPH-method is as follows (as specified in the documentation):

1. Calculating and applying all applicable SPH forces based on the fluid composition;
2. Performing the collision detection and generating contacts;
3. Correcting velocity by applying collision response forces and other forces (such as gravity);
4. Updating the fluid particle positions.

User Input The placement of particle emitters depends on manual user input via mouse clicking. This is also done during runtime, allowing the user to directly interact with the simulation. When a particle emitter is added, we create additional particles in the simulation according to the number of particles per emitter. These particles are not yet active and remain invisible until the first time they spawn at their designated emitter. Emitters can be placed anywhere on the character mesh as the user sees fit. For performance reasons we only allow a set number of emitters which is typically set to 50. The user can also pause and unpause both the fluid simulation and the character animation seperately. This has no direct effect on the fluid simulation itself but can be very useful when the user needs to take a closer look at certain occurences during runtime.

3.3.2 Parameters and settings

Below is a general description of the simulation parameters that are important for our project. Some parameters have fixed values for all experiments, while others are adjusted in order to achieve better fluid effects. The exact values for the parameter used in this project can be found in chapter 4, as well as an explanation on how the values were acquired. We have seperated the fluid parameters into two sets: global parameters and local parameters. Global parameters deal directly with the fluid simulation settings, while the local parameters define the settings and behavior of the SPH-fluid itself. All parameter values are set when the simulaton is initialized and should not be adjusted during runtime as this may crash the simulation or cause undesired errors leading to eratic fluid behavior.

Global parameters

Simulation Scale This is the scale at which the SPH density and forces are calculated, contrary to the world scale at which the rigid body simulation, rendering and all other functions occur. By default, 1 meter in the world scale is equal to 0.004 meters at simulation scale. Viceversa the conversion works the same, 1 meter at simulation scale is equal to 250 meters in world scale. The simulation scale is very useful for the SPH simulation as setting SPH-parameter values does not require complex tuning to prevent the fluid from exploding. Increasing the simulation scale makes the particles smaller. By default the simulation scale is set to 0.004. For our simulations we increase the simulation scale to 0.16 to accomodate for our particle radius that is 40 times smaller than the default setting (1 meter). In this report we refer to 1 unit being 1 meter on the Bullet Physics scale. A particle with a radius of 0.025 units is equal to 0.025 meters in the physics world.

Timestepping To improve performance and stability the fluid simulation runs in a different timestep than the rest of the physics world, 3 milliseconds versus 16 milliseconds. We do not change this value as the default setting results in a stable fluid simulation and is well suited for the fluid behavior in our experiments.

Local parameters

Energy restitution Causes particles to bounce of rigid bodies as they collide. Typically this value is set to zero to prevent additional energy to be introduced into the simulation, which may cause the fluid to become unstable. Using low values for the energy restitution should not cause any problems in general.

Boundary Friction Removes a fraction of the tangential velocity of a particle when it collides with a rigid body. Setting this to a relatively high value can be used to mimic the effect of fluids - such as blood - sticking to the virtual character when the adhesion force from our methods is not active.

Particle Radius Sets the radius of the (rigid body) sphere collision shape used for detecting the collisions with other rigid bodies in Bullet Physics (such as the virtual character). The particle radius does not affect the SPH calculations (like density) directly.

Viscosity Defines the fluid's resistance to flow. Higher viscosity makes it harder for the fluid to flow. For example, mud and slime have a high viscosity, while water and blood have a relatively low viscosity. Viscosity also offers numerical stability to the simulation as it reduces particle velocities.

Particle Expansion This parameter expands the collision detection radius of particles to find objects that are near, but not colliding with a individual particle. We use this in our experiments to find the closest point on the character surface for particles that are subject to our adhesiuon force method.

Stiffness Determines how violently the fluid reacts to density variations, defining the magnitude of the SPH pressure force. For performance reasons, generally this value should be kept as high as possible to simulate incompressible fluids. Generally, particles in (incompressible) fluids with low stiffness have more neighbour particles which results in more calculations.

4 Experiments

The experiments described in this chapter are conducted in order to determine appropriate implementation methods and simulation parameters to achieve realistic fluid effects for predetermined scenarios. We will use the perceived naturalness of the fluid behaviour to evaluate the effectiveness of our methods. Perceived naturalness essentially describes how well the simulated fluid behaves in terms of realism compared to fluids that exist in real-life environments. Our experiments are mainly focused on the simulation of bleeding effects, both in terms of spawning techniques for the fluid – e.g. the fluid origin such as cuts and bullet wounds – as well as the actual fluid behaviour as it interacts with the (animated) virtual character. The bleeding effects generally require a limited amount of particle emitters to achieve the desired visual results, meaning that the emitters are located at specific surfaces of the character mesh such as the arms or torso.

For all our experiments, the particle radius is set to 0.025 units. While it is theoretically possible to reduce the radius even further, our tests show that performance greatly suffers when the radius is set to values below 0.025 units. Most likely the cause for this lies in the fact that the simulation scale is directly tied to our particle radius, which we explain in section 3.3.2. Even though our particles should ideally be smaller to visually match real-life fluids, we can upscale our character model to maintain the illusion of small particles compared to the virtual character, without affecting performance. Upscaling of the character mesh might not be ideal for various reasons, however it is suitable for our experiments to obtain reliable results. The fluid stiffness is set to 0.5, where the default value is 1.5. In practice, reducing the stiffness created less volatile fluid behavior when used in our methods. Since blood has a higher viscosity than water, we used a value of 0.895 compared to the default value of 0.5 to reduce the flowing capacity of our fluid. The friction is set to 0.25, which is the default value of Bullet-FLUIDS. This value is already quite high but also best approaches the fluid behavior we want to achieve using our adhesion force method, making for an overal consistent visual result. We keep the energy restitution at 0, so we do not introduce any energy into the fluid simulation. The particle mass is set 0.0386kg which is calculated from the sphere volume of the particles ($^3/_4 \times \pi \times particleRadius^3$) and the approximate density of water ($1000kg/m^3$).

4.1 Bleeding with Adhesion Force

Regarding simulation of basic bleeding effects, simply spawning particles at their designated emitter will overall lead to less than convincing visual results. Fluids such as blood have the tendency to stick to other surfaces, otherwise known as adhesion. While the character mesh is positioned below a fluid particle, the applied friction between the surface and the particle is sufficient to simulate the fluid adhesion. Exceptions occur when particles encounter sudden changes in the character mesh such as (sharp) edges. Depending on the current velocity of the particle, this may result in detachment of the particle from the character surface, whereas in real-life environments the fluid would continue to remain in contact with the character surface due to the adhesion force. An example of this scenario can be seen in Figure 1. To avoid this unrealistic fluid behaviour we introduce our method, referred to as *adhesion force*, that aims to remedy the lack of fluid adhesion. For the methods described in this section we make use of static meshes since the lack of motion makes it far simpler to assess the actual visual results. Please refer to section 4.2 for our experiments regarding animated meshes.

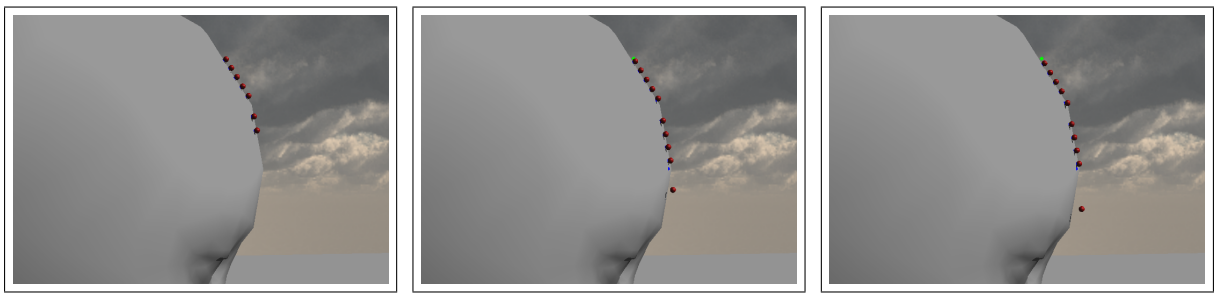

Figure 1: Lack of adhesion resulting in non-realistic fluid behavior.

The basic idea behind the adhesion force method is to calculate the distance that naturally occurs between particles and the mesh and pushing the particles back in the general direction of the mesh surface when the particles get separated from the mesh surface. More specifically, the adhesion force will be applied on individual particles in order to push them in the direction of the closest point on the character mesh. The magnitude of the adhesion force is determined by the actual Euclidean distance between the particle and the mesh. Naturally, this Euclidean distance can be expressed as a 3D distance vector that is calculated by using the 3D particle position vector and the 3D vector that defines the closest point on the mesh surface. The individual 3D distance vector components (X, Y and Z) are used to determine the magnitude of the adhesion force in each direction. For example, when a particle is positioned directly underneath the closest surface point, the adhesion force will only be applicable in the Y-direction as the distances in the X and Z-directions will be negligible. Since we have an active gravitational force in our simulation, we will not apply any adhesion force in case the Y-coordinate of the closest surface point is lower than the Y-coordinate of the particle position. In such cases, gravity will ensure that the contact between particle and mesh is maintained and the resulting friction will supply the adhesion force which will mimic the real-life behaviour of fluids that come in contact with other surfaces. Another condition for the application of the adhesion force is that at least one individual component of the 3D particle-mesh distance vector exceeds the distance that we have set as the threshold distance within the simulation. This threshold distance is referred to as the $minAdhesionParticleDistance$ and has a set value of 0.0001 units.

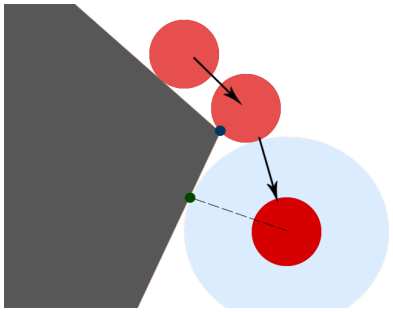

Figure 2: Expansion radius for particle (in red). Blue and green are respectively the last contact point and closest surface point on the mesh.

During each physics update of the simulation, the contacts between the SPH-particles and the virtual character mesh are calculated and updated based on the detected particle-triangle collisions. We manually calculate the distance between the particles and the mesh by subtracting the particle position vector from the mesh contact position vector. This does not account for the actual particle radius – the particle position is based on the centre of the particle – so we also deduct the value of the particle radius variable. The resulting distance that we store for our adhesion force calculations represents the absolute distance from the particle perimeter to the mesh surface (contact point). The acquired distance value is used to decide whether the adhesion force should be applied and based on the distance value we also determine the magnitude of the adhesion force. The closest point on the mesh surface is used as the direction for application of the adhesion force. However, depending on the velocity of a particle and the interval between physics updates, the last known contact point might not give an accurate representation of the actual closest point on the mesh. Instead of implementing our own algorithms to detect the closest surface point, we can define a value for the particle expansion radius variable. When set to a value larger than zero, the particle expansion allows for the detection of particle-mesh collisions within an enlarged radius around the particle. This means that, even as the particle is in fact no longer in direct contact with the mesh, we can iterate through all contacts that occur due to the expansion radius. We then take the closest surface point (in Euclidean distance) as the reference point for our adhesion force. For the graphical interpretation of the expansion radius and how it works, please refer to Figure 2.

4.1.1 Functions and Distributions

As mentioned earlier, the magnitude of the applied adhesion force is a direct function of the Euclidean distance between the closest surface point and a particle. For the experiments we have defined one linear function and several different (skewed) normal distributions that determine the exact adhesion force magnitude based on the input distance. The general idea for all our defined functions and distributions is that for increasing distances, the resulting magnitude will increase accordingly. The further a particle is removed from the mesh, the more force needs to be applied to push the particle back in contact with the mesh in order to maintain the adhesion effect. However, we also make the assumption that for every function and distribution there is a predefined critical distance that serves as a limitation to prevent unrealistic fluid behaviour. When the separation between particle and mesh becomes too great, the adhesion force is no longer able to successfully maintain the contact between particle and mesh and the magnitude of the adhesion force will (rapidly) diminish. The breaking point distance is a predefined value for our linear function and for the (skewed) normal distributions it is a combination of the mean and variance combined. Below we will elaborate on the used linear functions and

(skewed) normal distributions including their intended effect and actual outcome.

Standard Linear Function

We start off with a piecewise linear function to calculate the magnitude ($F_m(d)$ of the adhesion force based on the distance (d) between the particle and the closest point on the mesh. We define variable $breakingPointDistance$ (d_{bp}), which is the particle distance at which the adhesion force is at its strongest. After the particle distance exceeds the $breakingPointDistance$, the adhesion force (quickly) diminishes. We define the maximum strength of the adhesion force as the variable $maximumForce$ (F_{max}), which we can vary to control the overall strength of the adhesion force. In case of the diminishing force after reaching $breakingPointDistance$, we also define variable $maximumDistance$ (d_{max}) at which the adhesion force should be back at zero. By varying $maximumDistance$ we can control the rate in which the adhesion force diminishes. Formula 1 describes the piecewise linear function using a diminishing force after reaching $breakingPointDistance$. First, we experiment with the linear adhesion force calculated using Formula 1.

$$F_m(d) = \begin{cases} (F_{max}/d_{bp}) \times d_{bp} & \text{if } d \leq d_{bp} \\ (F_{max} \times (d - d_{max}))/(d_{bp} - d_{max}) & \text{if } d > d_{bp} \end{cases} \quad (1)$$

First we defined the value of $breakingPointDistance$, at which the adhesion force is at its strongest, equal to 100% of the particle radius (=0.025 units), and the $maximumDistance$ to 200% of the particle radius (=0.05 units). Simply put, this means that the magnitude of the adhesion force increases and diminishes at an equal rate. Initially, we kept the value for the particle expansion radius at 0. Running experiments with these values yielded results that were (visually) not very satisfactory where particles fail to stick to the mesh. This can be explained by the fact that we use the expansion radius to detect the closest point on the mesh surface for each particle. If we set the value for the expansion radius to zero, the last known closest point on the mesh surface is equal to the last contact point between the mesh and the particle. Depending on the actual particle velocity, the last known contact point may be very different from the actual closest point on the mesh surface, as shown in Figure 2. This means that, when we calculate the distance between the particle and the last known contact point, the particle potentially appears to be further away from the mesh surface than it actually is. Using this 'incorrect' distance - which is typically larger than the 'real' distance between particle and mesh - in the adhesion force magnitude ($F_m(d)$) calculation is not ideal and causes the adhesion force to perform inadequately. To remedy this situation we set the particle expansion radius equal to the $maximumDistance$ variable (for the piecewise linear function). This means that, until the distance between particle and mesh exceeds the value of $maximumDistance$, the expansion radius of the particle always returns the closest point on the surface. Subsequentially, we always calculate the correct adhesion force magnitude. Experimenting with the adjusted value for particle expansion radius yielded far better results. However, we discerned another visual flaw, where particles that are a reasonable distance away from the character mesh (and naturally no longer susceptible to the adheshion force) were still attracted towards the mesh.

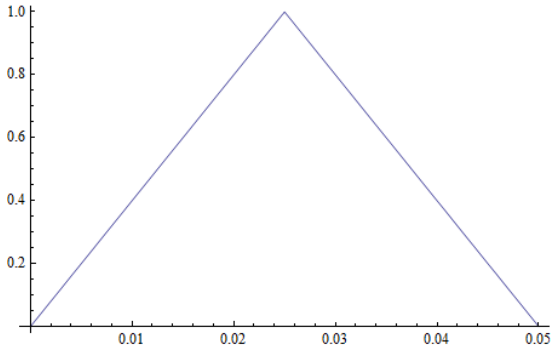

Figure 3: Linear force using Formula 1, $d_{bp} = 0.025$, $maximumDistance = 0.05$

There are two ways to deal with this unnatural attraction phenomenon. First, we can (drastically) reduce the $maximumDistance$ variable such that it nears the $breakingPointDistance$, using values of 150% or 125% of the particle radius. This elimates most of the problem but, as we noticed during further experiments, still causes some eratic fluid behavior from time to time. The visual results keep improving as we reduce the $maximumDistance$, as particles are no longer unnaturally attracted towards the mesh. Once the $maximumDistance$ is nearly identical to the $breakingPointDistance$ (≈101% of the particle radius) we almost cannot discern any more improvement. This leads us to believe that eliminating the adhesion force entirely after the particle distance exceeds the $breakingPointDistance$ may yield the same results. After all,

it makes intuitively sense that once the particle reaches a certain distance from the mesh, the adhesion force does not longer apply. To accomplish this, we reduce the adhesion force's magnitude to zero after the distance reaches $breakingPointDistance$, which is shown by the piecewise linear function in Formula 2.

$$F_m(d) = \begin{cases} (F_{max}/d_{bp}) * d_{bp} & \text{if } d \leq d_{bp} \\ 0 & \text{if } d > d_{bp} \end{cases} \tag{2}$$

After running a few simulations, we determined that using the piecewise linear function from Formula 2 yields better results compared to the one used in Formula 1. This supports our claim that removing the adhesion force after the particle distance reaches $breakingPointDistance$ improves the overall naturalness of the adhesion force. An additional advantage of using Formula 2 is that we do not need any additional calculations once the $breakingPointDistance$ is reached, which should increase performance. Still, with our test configuration the actual gain in performance was negligable with no discernable change in framerate.

Another problem we notice, especially on steep surfaces, is that particles still get detached even though we expect them to stick to the mesh. The most probable cause is that the strength of the adhesion force is not adequate, which we can solve by either scaling the $maximumForce$ variable or decreasing the $breakingPointDistance$. Setting a lower value for $breakingPointDistance$ 'pushes' the particles more towards the mesh for lower particle-mesh distances. This reduces the potential particle distance in the next physics timestep, requiring less force to maintain the adhesion effect. We have conducted some tests in which we gradually lowered the $breakingPointDistance$ from 100% particle radius (= 0.025 units) to 5% of the particle radius (= 0,00125 units). Setting the $breakingPointDistance$ to 25% of the particle radius (=0,00625 units) yielded the best result for our simulations. Likely related to the physics timestepping, setting the value lower than 25% starts to negatively affect the adhesion force, in which case the particles fail to stick to the mesh. Judging the naturalness of the adhesion effect using the determined values for both $breakingPointDistance$ and $maximumDistance$, we still notice some scenarios in which the particles do not stick to the mesh when we would naturally expect them to. By scaling the $maximumForce$ from 1 to 5 during various test, we find that a value of 3 leads to the most convincing adhesion effect. This way particles stick to the mesh as we expect them to, except at certain mesh features such as sharp edges or when the particle velocity becomes too high. Figure 4 shows the results from implementing the piecewise linear function from Formula 2, where the $breakingPointDistance$ is set to 25% of the particle radius

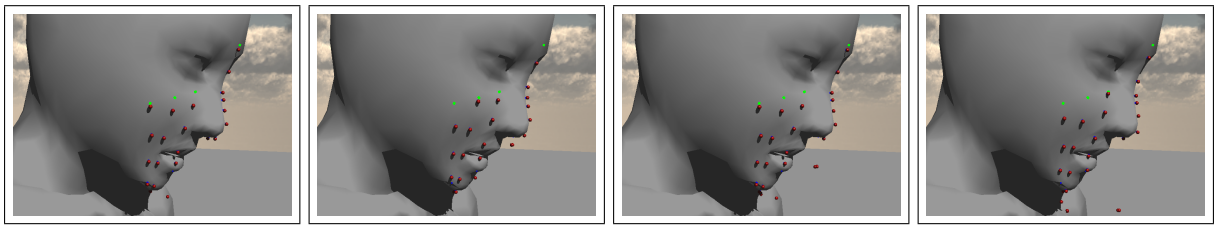

Figure 4: Bleeding effect where the $maximumDistance$ is equal to the $breakingPointDistance$.

(Skewed) Normal Distribution

The linear functions we use for the adhesion force already produce satisfying results, although sometimes the particles react quite eratic to the adhesion force. This may be solved by using a smoother function for the adhesion force magnitude, which we describe in this subsection. Especially smoothening out the magnitude near the $breakingPointDistance$ is likely to have a positive effect on the naturalness of the adhesion force.

Initially we determined a suitable normal distribution where the *mean* is equal to the ideal $breakingPointDistance$ value (e.g. strongest adhesion force) we determined for the linear force function, placing the mean at 25% of the particle radius. To determine the suitable value for the variance we start with 100% of the particle radius, resulting in the function graph seen in Figure 5. Unlike the linear function, this results in a relatively large adhesion force magnitude for particle distances starting at 0 units. Based on earlier assumptions that we only want the adhesion force to affect particles that are no longer in contact with the mesh, this does

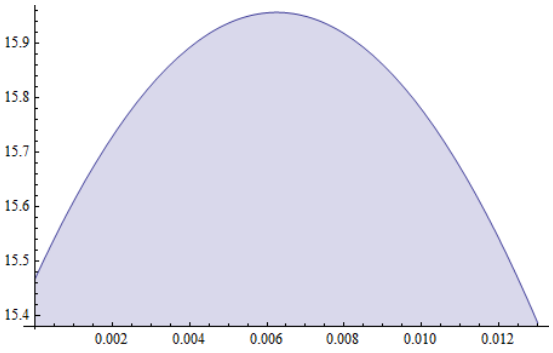

Figure 5: Normal distribution with μ=0.00625, σ=0.025

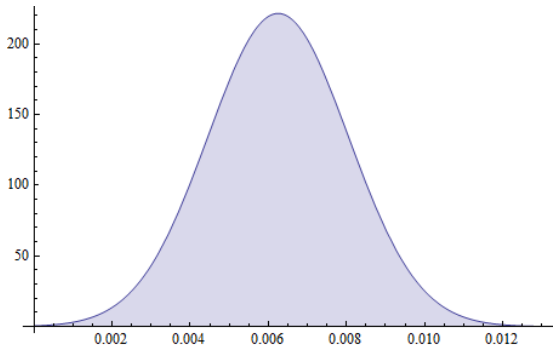

Figure 6: Normal distribution with μ=0.00625, σ=0.0018

not seem desirable. To test our assumption, we ran a limited number of simulations with the current normal distribution to determine the outcome. As expected, the results show that the adhesion force performs well with regard to making the particles stick, but the effect is far too strong and does not look natural in the slightest. Even when scaling down the adhesion force by 20% to match the $maximumForce$ (value of 3) from our linear function, the effect still looks excessively unnatural. Having ruled out that the current setup for our normal distribution is capable of producing a suitable adhesion effect, we re-introduce the dependency that the adhesion force magnitude is obsolete for particle distances ≤ 0. We maintain the mean at its ideal, original value, which automatically implies that, in order to meet the previously mentioned dependency, we are forced to lower the variance value. The dependency is met when the variance is set to a value of approximately 0.0018 which is roughly 19% of the $breakingPointDistance$ (see Figure 6). Simulations that we run with the revised normal distribution look considerably more natural even though the adhesion force magnitude, looking at the function graph, seems excessively large. Lowering the magnitude indeed has a large positive effect on the naturalness as we set the magnitude as low as 1% and 2% of its original strength for the X+Y and Z distance components respectively.

The normal distribution already shows great results in terms of how natural the bleeding effect looks. Yet, we encounter the same flaw as with the linear force where the particles are unnaturally attracted towards the mesh for larger distances. We account this to our normal distribution which is inherently symmetrical in nature. There are two scenarios we explored to find a possible solution to this, with both scenarios revolving around introducing *skewness* to our normal distribution. We can either introduce positive skew to make the adesion force more effective for lower distances, or introduce negative skew which causes a (more) rapid drop in the adhesion force magnitude after reaching the $breakingPointDistance$. We conducted further experimentation with equal values for the mean and variance as the regular normal distribution. What we see is that the adhesion force only has effect on larger distances, which would cause the adhesion force to fail as we have seen earlier. After running several experiments we come to the conclusion that the best results come from setting the skew (α) to a value of 5 and -5 for the positive skew and negative skew respectively and by shifting the mean towards the left. For the positive skew the new value for the mean is 20% of its original value, in our case 0,00125 instead of 0.00625. The new mean value for the negative skew is adjusted to 68% of its original value: 0.00425 versus 0.00625. In addition to shifting the mean for the negative skew, we have also found that slightly reducing the variance from 0.0018 to 0.001 (marginally) improves its results. The new graphs for the mean and variance of both skewed normal distributions are shown in Figure 7. Since the maximum possible magnitude of the negative skew function is approximately 1.5 times that of the positive skew, it also requires 1.5 times more scaling than the positive skew function. Overall, we found the best results where the scaling for the X, Y and Z components of the adhesion force lies between 100% and 67% for the negative skew and between 50% and 67% for the positive skew. Finding the absolute 'correct' scaling however is not a simple task since it heavily depends on the user preference with regard to the resulting fluid effect.

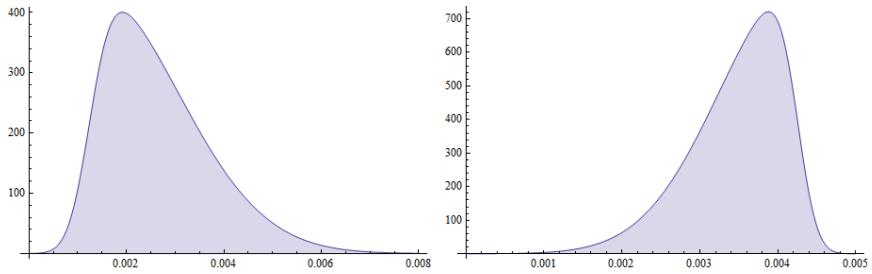

Figure 7: Unscaled positive and negative skew normal distributions with skewness of 5 and -5.

We have provided some images with results from both the negative and positive skew implementations (using the suggested scaling) in Figures 8 and 9. Both approaches work well and produce natural natural bleeding effects that, in practice, look almost identical. After running several additional simulations we find that increasing the skewness for both the positive and negative skewed normal distributions does not visually improve nor degrade the bleeding effects. As for picking the best suitable skewed normal distribution we find that the negative skew is preferred over the positive skew due to the more sudden drop in magnitude after reaching the $breakingPointDistance$. This property makes the negative skew normal distribution slightly more robust under various conditions, but again, the difference is marginal.

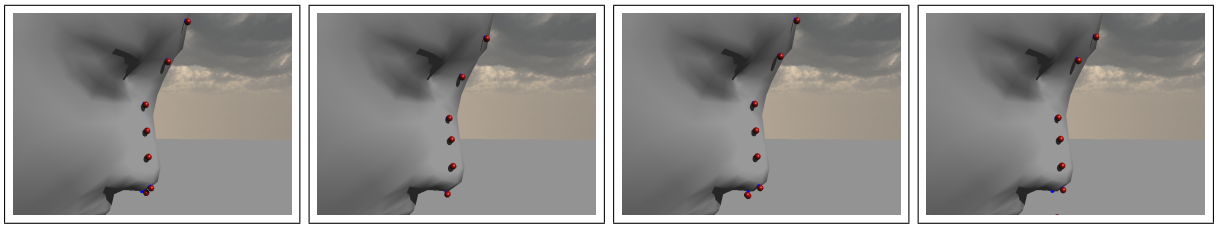

Figure 8: Bleeding effect using a positive skewed normal distribution ($\mu = 0.00125$, $\sigma = 0.0018$, $\alpha = 5$).

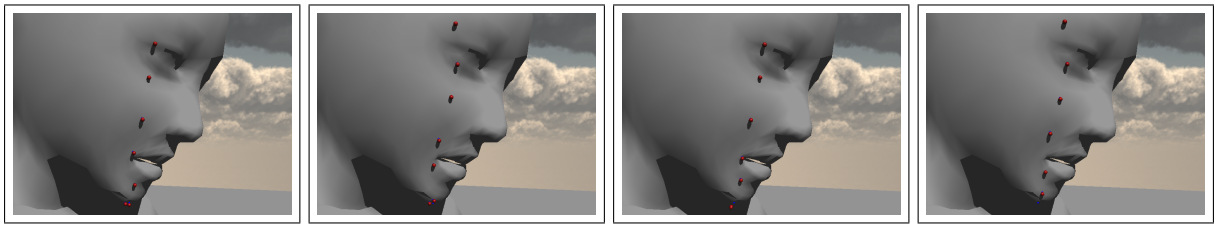

Figure 9: Bleeding effect using a negative skewed normal distribution ($\mu = 0.00425$, $\sigma = 0.001$, $\alpha = -5$).

4.1.2 Neighbour Scaling

Originally we designed the neighbour scaling method to influence the magnitude of the adhesion force in the Y-direction. The theory behind this is that we intended to mimic the increased gravitational pull on particles that accumulate into a 'droplet' of fluid due to cohesion. As the droplet hangs underneath from a feature of the character mesh, the combined mass of the particles would cause the adhesion effect to be less effective. The intended implemention of the neighbour scaling is a fairly straight-forward method that, for each individual particle that has adhesion force in the Y-direction, takes into account the presence of possible neighbouring particles. We introduce a scaling factor by which we divide the Y-component of the adhesion force per particle depending on the amount of neighbours. Intuitively we want the adhesion force to gradually diminish where each neighbour does not contribute equally to the scaling factor.

Instead of using a linear function, we have chosen to use a (simple) polynomial function instead: the scaling factor we will apply to the adhesion force Y-component (F_{aY}) is $\sqrt{n}$, where n is the amount of neighbours. Therefore we can say that $F_{aY} = F_{aY} \times (1/\sqrt{n})$. As the count of neighbours grows, the total effect on

the scaling factor is gradually reduced. The maximum amount of neighbours that can exist for each particle depends mainly on the fluid stiffness. As mentioned earlier, we have set the fluid stiffness to a value of 0.45, leading to a maximum of approximately 30 to 40 neighbour particles.

Combining the neighbour detection method with the definitive (negative) skewed normal distribution, we see no (substantial visual) improvements but the bleeding effects remained natural nonetheless (Figure 10). We account this to the fact that the theory behind this method is based on an older implementation of the adhesion force, where particles would excessively stick to the mesh. As we described in section 4.1.1 the adhesion force produces natural and convincing results. By lowering the stiffness value of the fluid, we already see that fluid particles cluster together more and break away from the mesh more easily. In conclusion, the neighbour detection method we describe here has not improved our experimentation results. If the fluid stiffness is increased up to a value of approximately 1.5, the particles have substantially more 'trouble' to clump together, in which case the method could add to the visual realism. For our project we did not test this, as our results would influenced dramatically by increasing the stiffness by 300%. In the next section (4.1.3) we describe our surface tension method, which is based on principles from our neighbour scaling method.

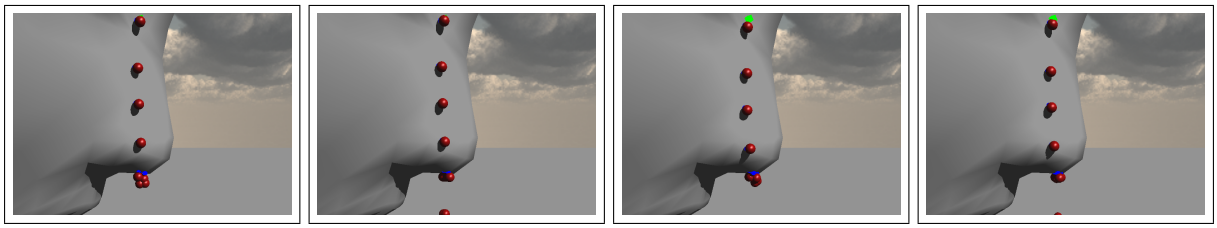

Figure 10: Reducing the adhesion force in the Y-direction using neighbour scaling.

4.1.3 Surface Tension

Cohesive forces between fluid particles are responsible for what is called surface tension, making individual fluid particles cluster together into droplets. Each fluid particle is pulled equally in the direction of its neighbour particles. In our experiments we have implemented a basic approach to mimic the effect of surface tension (see Formula 3). When determining the neighbours (N_p) of each particle p, as shown in section 4.1.2, we also calculate the attractive force towards each neighbour particle n in the same loop. The attractive force is calculated by taking the distance (d_{np}) between both particles and dividing it by its squared distance. This way we automatically compensate for the actual distance between particles since the actual attractive force diminishes as the distance between particles grows larger. For each particle the attractive forces towards its neighbours are summed up as the surface tension force Ft_p and stored seperately each loop. In order to decrease and increase the overall strength of the surface tension, we also introduce a scalar variable α that we apply to the total surface tension force of each particle.

$$Ft_p = \alpha \sum_{n \in Np} \frac{d_{np}}{||d_{np}||^2} \tag{3}$$

In Figure 11 some examples of the surface tension effect are shown with varying values of α. Overall, our basic implementation of surface tension does not improve the bleeding effects much. The surface tension effect is best visible in situations where many particles are gathered, for example in puddles. As for the bleeding effects such as cuts and punctures (section 4.3) the difference is only noticable when the number of neighbours of each particle. is high enough. Our method does not aim to minimize the fluid surface area, but only applies neighbour attraction forces each simulation loop. As a result the particles never reach a stable rest state, causing the fluid to continously move and deform (slightly) even when one would not expect it to do so. Hence we can say that our surface tension method does improve the natural feel of fluid, but it is also inherently flawed.

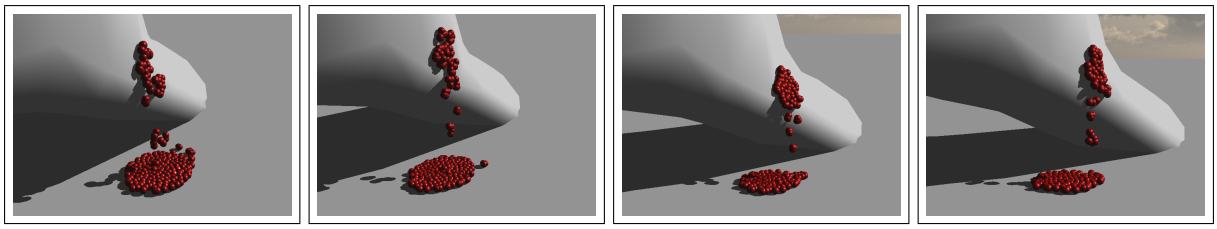

Figure 11: Examples of the fluid using surface tension, $\alpha = \{0, 0.334, 0.667, 1\}$.

In succession to implementing our own surface tension method, the author added a surface tension technique to the Bullet-FLUIDS library as well. This technique is based on work done byAkinci et al. [8]. In this paper the surface tension force consists of two force components: cohesion force and curvature force. The cohesion force forces particles to attract each other, similar to our own implementation. Additionaly, the curvature force acts to minimize the surface area of the fluid by defining a *scalar field*. The curvature force pushes the particles together to form the shape of a sphere (or ellipsoid), balancing out the curvature force. Figure 12 shows results from varying the Bullet-FLUIDS surface tension parameter as mentioned in section 3.3.2. For our specific bleeding effects, the best results were acquired using a value of 0.25 for the Bullet-FLUIDS surface tension. The fluid does not look considerably more natural when using the Bullet-FLUIDS implementation for surface tension, as seen in Figure 11. But, unlike our own implementation, the fluid does reach a stable rest state due to the surface area minimalization. The framerate suffered considerably on a few occassions, however the author indicated that the implementation is still in an experimental stage.

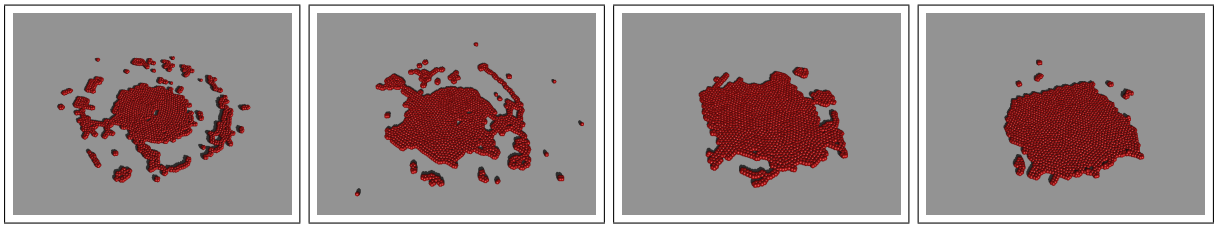

Figure 12: Examples of the fluid using Bullet-FLUIDS surface tension, $\alpha = \{0.25, 0.5, 0.75, 1\}$.

4.2 Animated meshes and bleeding

Another part of our experiments is simulating animated meshes in combination with our bleeding effects. Simply put, we can imagine a walking virtual character with a cut in its arm that is bleeding heavily. We require an updated collision shape for the virtual character after each change in the animation state. Should the character's arm move, we want the physics (i.e. fluid bleeding effects) to take that into account. Updating the collision shape is fairly straightforward, as we supply the vertices of the character mesh to Bullet Physics (each frame), resulting in an updated triangle mesh collision shape. The main problem here is that updating the collision shape for each frame has a significant impact on performance. We have noticed drops of over 200fps for meshes that consist of over 30.000 vertices compared to building the collision shape once at initialization. If the character mesh is kept fairly low-detail, by reducing the vertex count to approximately 1500 - 2000, the performance is not affected as much. This is certainly something to consider when using animated meshes, especially when working with real-time systems where every impact on the framerate counts.

During our experiments we found that there is another downside when working with animated meshes. As we mentioned before, the collision detection between the triangle mesh and particles has its limitations. This is not a problem for static meshes, as we can scale both particles and mesh such that the collisions are detected adequately by the physics engine. When we run simulations where the character is animated, we immediately observe that in many cases collision between the particles and mesh are missed, causing the particles to fall through the mesh (tunneling). This is a serious problem that makes the fluid effects look far from natural. After consulting the Bullet Physics forums and documentation, we must come to the conclusion that Bullet Physics has no adequate support (by default) for collision detection between primitive shapes and animated triangle meshes. The only partial solution we have found to this problem is to reduce the speed of the animation and/or increasing the offset for the emitter placement. When the mesh is animated more slowly or the initial

distance between spawning particles and the mesh is larger, the collision detection improves somewhat.

Based on the character motion, we also apply an initial velocity to particles as they spawn based on the emitter positions. For each emitter we track its position in the current and previous timestep. By considering the time that has elapsed since the last timestep and the relative distance between the current and historic emitter positions, we can calculate the velocity of the emitter motion. When a particle spawns, we add the (interpolated) emitter velocity to the particle's initial velocity. In most cases the particle's initial velocity is zero, but this may not be the case when dealing with effects like gushing (see section4.5.1). Assigning initial velocities to the particles (based on the emitter motion) improves the natural feel of our bleeding effects considerably, removing most of the 'static feel' that occurs when particles spawn with zero velocity.

4.3 Differentiated fluid sources

In case of simulating bleeding effects, our fluid source would typically be some sort of injury inflicted upon the virtual character. As we can generalize bleeding towards the more abstract concept of fluid emittance, we can in fact define fluid sources that can be applied to a multitude of virtual characters. While bleeding effects would typically apply to human (or animal) characters that were injured, such effects could also be used to simulate damage inflicted to inanimate objects. Examples range from an animated robot character with a tear in its oil hose to a static punctured plastic bottle filled with milk. As for the visualization of the fluid origin, it holds true that many materials are susceptible to puncturing, tearing or cuts in general, disregarding exact material properties that would normally apply in real-life scenarios. For our experiments we focus on defining two types of different forms of injury that act as the source of our simulated fluid: (deep) cuts and punctures. In the following subchapters we will describe the individual sources and report how specific emitter placement and timed particle spawning can convincingly contribute to the visualization of injuries that bleed.

4.3.1 Superficial and deep cuts

To achieve the visual effect of superficial and deep cuts, the emitters should be placed along a (curved) line that represents the actual injury, as can be seen in Figure 13. The main importance when defining the cut is that the distance between emitters is kept within boundaries. Placing the adjacent emitters too close will result in particles that spawn (virtually) on top of each other, which negates the individual SPH-forces. When the distance between emitters is too large, particles that spawn at the emitters will be too separated to apply the attracting SPH-forces and the visual appearance no longer resembles the bleeding effect from a cut. If we use the default spawning location of particles – i.e. at the exact position of the emitter – then the optimal distance between the emitters should be roughly twice the particle radius. That way the spawned particles fit next to each other which supports the idea of a cut that is bleeding.

Since particles have a limited maximum lifetime before they are relocated to their designated emitters, we can influence the appearance of the bleeding effects by assigning starting lifetimes to the particles as they are created at initialization. When simulating effects based on steadily bleeding cuts, we want the particles to spawn at regular intervals. The interval is calculated by dividing the maximum particle lifetime by the amount of particles per emitter. For example: if we have a maximum particle lifetime of 10 seconds and there are 100 particles per emitter, the spawning interval is (10/100 =) 0.1 seconds. If we combine this with our emitter placement strategy for cuts, we are able to simulate a visually convincing effect resembling a cut, that is steadily bleeding, on the virtual character. Depending on the severity of the injury (cut) we can increase the particle count per emitter. This will decrease the interval in between particle spawns; if we take the previous example and increase the particle count to 200 per emitter, then the spawning interval will be reduced to 10/200 = 0.05 seconds. There is no 'best value' for the spawning interval, due to many external influences. Spawning particles too fast in combination with a high fluid stiffness will cause the fluid to 'explode' and low friction in combination with long intervals will not produce a coherent effect. In Figure 13 we show a simulated cut that is steadibly bleeding.

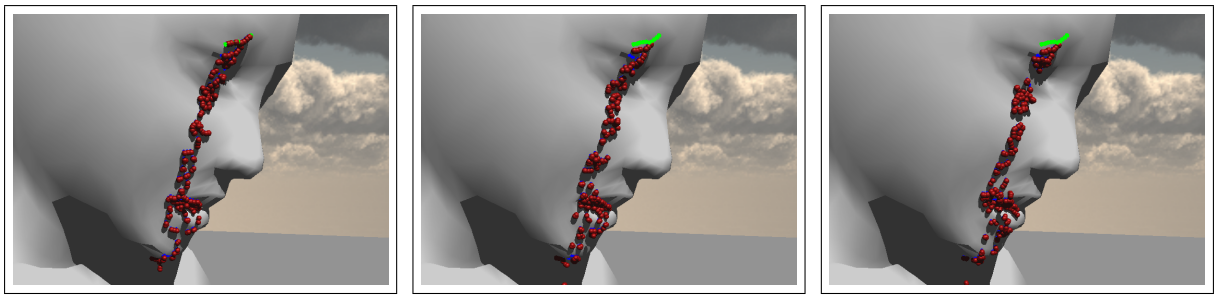

Figure 13: Example of a bleeding cut simulation using 10 emitters and 30 particles per emitter.

4.3.2 Puncturing

Our puncture simulations aim at recreating the scenario of a foreign object penetrating the rigid body of our virtual character, i.e. creating a puncture in its surface. There are numerous applications for simulating fluid pouring out of holes in characters and/or objects. For our experiments we focus on gunshot-like injuries that cause serious loss of blood and are inflicted upon our humanoid character. Unlike the injuries described in the previous sections, the source of the bleeding effect - associated with puncturing - is exclusively concentrated around the area where the surface has been penetrated. Initially our results show that the best method to simulate the gunshot-like injury is using a small amount of emitters around the puncture location and spreading them based on the actual diameter of the puncture. For now the exact placement of the emitters is user-defined. The experiments based on puncturing assume that the spawning behaviour of the particles is less continuous than with (deep) cuts. We aim to achieve results that approximate a form of gushing as opposed to steadily streaming bleeding effects.

The requirement of the gushing effect we strive to accomplish is that we change the regular spawning intervals to a form where the majority of particles is spawned in a short period of time. This leaves the remainder of the particles to spawn over the remaining time interval. Similar to our adhesion force method, we can employ the use of a (skewed) normal distribution that causes a spike in the particle spawning. By adjusting the variance, we can control the length and severity of the overall gushing. Having a fairly large variance will flatten the distribution where the particle spawning is more evenly spread. Low variance will cause the majority of particles to spawn semi-simultaneously and thus creating a more violent gushing effect. Figure 14 shows the results and visual differences using low- and high variance for two user-defined punctures. The maximum particle lifetime is set to 10 seconds.

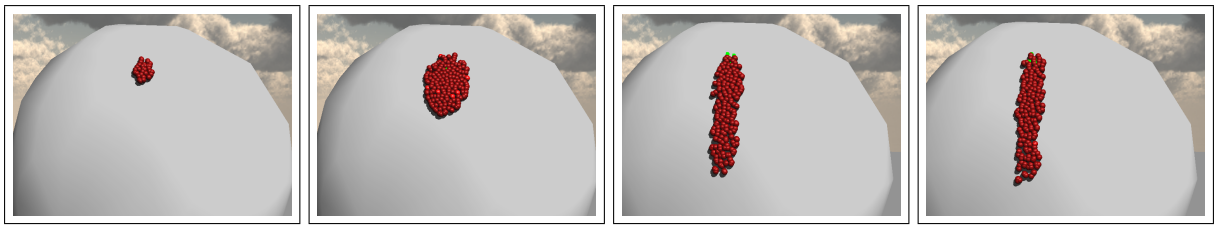

Figure 14: Puncture wound with low variance in the two left images and high variance in the right images.

4.4 Optimized Particle Spawning

When dealing with a (very) high amount of particles that spawn at one single position within a small timeframe, particles have a high tendency to overlap each other. This is especially the case in the gushing bullet wound scenario. When particles overlap in the simulation, there are two possible outcomes. Either the particles are not susceptible to each other's SPH-forces or, when they do exact SPH-forces, the particles will push each other away depending on the stiffness parameter of the fluid. The stiffness parameter of the fluid is set to a value of 1.5 by default, which often causes the fluid to behave unnaturally and even, in extreme cases, to explode violently if many particles (highly) overlap. In such cases, reducing the stiffness value helps in remedying most of the eratic fluid behavior. Reducing the fluid stiffness too much might not be desirable however, depending on the fluid effect and the previously acquired optimal combination of parameter values.

To prevent (most of) the overlapping of particles that spawn (almost) simultaneously, we apply an offset to the emitter position that we use as the new spawning position for particles. As mentioned before, the emitters are placed on the plane that is perpendicular to the mesh surface. The particle spawn offset is calculated by using *polar coordinates*. For each spawning particle a random value for angle φ is calculated that lies between $[0, 360]$ degrees, while omitting the value of θ. This means that we pick the offset position in a disk around the emitter. The radius r of the disk is also defined by a random value, however defining a suitable range for r is important for the resulting effect. As the allowed range for the random value of radius r is increased, more spread is introduced to the particle spawning. In case of bleeding effects that deal with cuts, the calculated particle offset should generally be relatively small to ensure that particles remain close to the emitter when spawning. On the other hand, increasing the spread of the particles works well for the puncture bleeding effect. By increasing both the range of radius r and the amount of particles per emitter, we can elimate the need for multiple emitters that are placed close to each other in order to mimic the appearance of a puncture as seen in Figure 15. The formula for determining the offset vector is described in Formula 4. The offset method can be applied to any of our bleeding effects but is most useful and effective for the puncture effect.

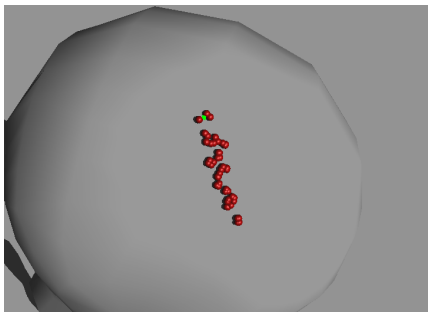

Figure 15: Bleeding effect using disk offset ($r = [0.025, 0.045]$) positions for spawning particles.

$$V_{offset} = (r \times sin(\theta) \times cos(\varphi), 0, r \times sin(\theta) \times sin(\varphi)) \tag{4}$$

4.5 Starting Velocities

By modifying the starting velocity of particles we control the behavior of the fluid, making it possible to enhance our bleeding effects. The enhancing fluid effect we describe in this section is *gushing*, which can be used for simulating heavy bleeding. The focus for the gushing effect lies on static meshes for our experiments but, in theory, the gushing effect can be applied to animated meshes as well. We propose combining several of our methods with our gushing effect to simulate realistic gushing bleeding effects.

4.5.1 Gushing Effect

In case of static meshes, we assign zero velocity as the initial particle velocity when particles spawn. In case of animated meshes we take the emitter displacement – compared to the previous frame – and the resulting velocity. The naturalness of this technique is quite high when dealing with animated meshes that show much movement. Due to the velocity of both the mesh and the spawning particles it is hard for the viewer to discern any particularly detailed effects, making it easier to approximate 'correct' fluid behaviour that looks natural. This is not the case in scenes with static or slowly animated meshes where lack of movement draws the attention explicitly to the visual aspect of the bleeding effect. Especially scenarios where many particles are used to simulate the bleeding effect, such as deep cuts and gunshot wounds, have a unnatural static feel to them. Most wounds, as found in real life, exhibit behavior where the fluid (blood) originates from the body itself, and due to pressure and the incompressibility the fluid gushes out of the wound. This behavior is what we base our *gushing effect* on (Figure 16). Contrary to wounds in general, our fluid does not originate from within the character surface since we place the emitters directly onto the character surface in order to avoid the initial collision resolution between particles and the collision shape. To mimic the gushing effect, we apply an additional starting velocity to spawning particles. The gushing effect acts outwards from the wound, i.e. along the surface normal that we update during each simulation loop. We define a variable $gushingMultiplier$ that is used to control the magnitude of the gushing effect by acting as a scalar for the additional gushing velocity. Getting the random direction offset for the velocity is done using polar coordinates, similar to the offset method in section 4.4. We take a random value in the range of $[0, 15]$ for angle θ to determine the offset direction vector that is added to the surface normal vector. Depending on the type of wound and the desired effect, the spread of the gushing effect can be limited by reducing the allowed range for angle θ. Intuitively, the gushing effect is naturally biased towards the surface normal due to the properties of actual wounds. As such, restricting the angle θ to $[0, 15]$ already results in a natural looking gushing effect for most cuts and gunshot wounds. To simulate the appearance of increased pressure in the wound, i.e. intensifying the gushing effect, the range for angle θ can be restricted further as well as increasing the value for $gushingMultiplier$. The formula for determining the gushing offset vector is described in Formula 5.

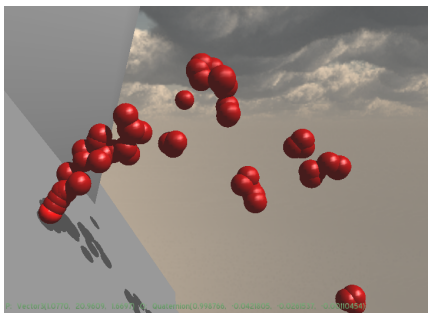

Figure 16: Closeup of the gushing effect.

$$V_{offset} = (r \times sin(\theta) \times cos(\varphi), cos(\theta), r \times sin(\theta) \times sin(\varphi)) \tag{5}$$

We can combine the gushing effect with our other methods to create gushing gunshot wounds and heavily bleeding gushing cuts. We can even combine the emitter offset method with the gushing method, making it easy to simulate gushing puncture wounds by placing one single emitter. In Figure 17 we show a gushing puncture wound with varying strengths, leading to different visual results.

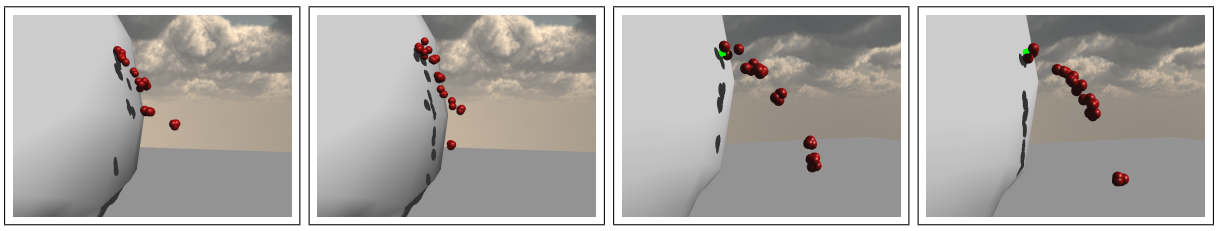

Figure 17: Gushing effect using various strengths (0.25 in the two left images, 0.75 in the two right images).

4.6 Performance Analysis

Besides the visual aspects of the implemented fluid effects, performance is likewise important due to the real-time focus of our experiments. In this section we review the impact of the discussed methods on performance by taking a look at the performance of each method individually and also in combination with other methods. The machine used for testing is equipped with a Intel® Core™ i7-950 with a Nvidia® GTX770 graphics card. The test are performed using billboard rendering for the particles to minimize the impact on performance. We only consider the surface tension, adhesion force and gushing methods, since these methods have the largest impact on performance. Our surface tension method relies on determining the neighbours of each particle, meaning we do not require an individual assessment for the neighbouring method from section 4.1.2. In Table 1 we show the performance results for our individual methods. We do not consider the optimized particle spawning method seperately, since determining the offset position for particles (section 4.4) implements the same polar coordinates technique as the gushing effect method, resulting in nearly identical performance for both methods.Table 2 displays the performance results of four (useful) pairs of combined methods, such as the adhesion force combined with surface tension. For each benchmark we let the simulation run for one minute after which we determine the average framerate from Ogre's internal framerate counter. This gives us a clear indication of the overall performance of our methods (combined and individually).

Table 1: Average fps performance results of individual simulation methods.

Method	Number of emitters	Particles per emitter	Triangle Count	Average FPS
Default setup (none)	10	100	21158	403
	25	100	23278	138
Adhesion (normal distribution)	10	100	21158	235
	25	100	23278	49
Surface tension	10	100	21158	372
	25	100	23278	185
Gushing	10	100	21158	401
	25	100	23278	230

Table 2: Average fps performance results of combined simulation methods.

Method Combination	Number of emitters	Particles per emitter	Triangle Count	Average FPS
Adhesion (nd) + surface tension	10	100	21158	217
	25	100	23278	47
Adhesion (nd) + gushing	10	100	21158	310
	25	100	23278	47
Adhesion (nd) + surface tension + gushing	10	100	21158	200
	25	100	23278	47
Surface tension + gushing	10	100	21158	364
	25	100	23278	118

The adhesion force is the method that has by far the most impact on the performance, however the method could be optimized more using GPU buffering techniques from Xu et al. [1]. We have also optimized our framework by implementing lookup tables for the (skewed) normal distributions. We sample the normal distributions at regular distance intervals and store the calculated magnitude. During runtime we consult the lookup tables for the actual mesh-particle distance using linear interpolation on the two magnitude values that lie closest to the relevant distance. Notice that sometimes the adhesion force performs better in combination with other methods. While this seems illogical, this is most likely caused by having less 'sticking' particles for the adhesion calculations. For example, when the gushing effect is active the particles are 'pushed' away from the mesh. In that case the distance between the particle and mesh becomes too large more often, resulting in less adhesion force that needs calculating. Obviously, this increases the simulation performance. The other methods that improve the bleeding effects, such as surface tension and gushing, have considerably less impact. However, performance in general decreases quickly as the amount of particles is doubled.

5 Conclusions

Based on the results from the experiments, we can conclude that smoothed particle hydrodynamics (SPH) are (very) suitable for simulating fluid combined with virtual characters. We were able to achieve convincing, natural looking bleeding effects but this requires implementing several methods that are complementary to the SPH-method. Most of the methods we introduced in this experimentation project substantially improve the visual quality while being reasonably simple to implement. The adhesion force method delivers the biggest improvement for simulating fluids that stick to other surfaces, such as blood. It also has the greatest impact on the simulation performance due to the amount and complexity of the calculations done during each simulation loop. Implementing optimizations such as the lookup tables are therefore mandatory to make the adhesion force method suitable for real-time applications. We were able to achieve realistic bleeding effects on static meshes by combining the adhesion force with both our emitter placement and starting velocity strategies, without greatly impacting the simulation performance. For animated meshes the results are less satisfying regarding both performance and immersion. The biggest visual boost for our bleeding effects in combination with animated meshes would come from improving the collision detection so that the particles are no longer 'tunneling' through the moving mesh. Rebuilding the collision shape every animation frame has the largest impact on the framerate. Introducing an optimized (or different) method that creates the collision shapes more efficiently would certainly benefit the overall performance.

In conclusion, our opinion is that smoothed particle hydrodynamics can be put to good use for simulating bleeding effects, elimiminating the need for manual animation or unrealistic traditional particle systems. This could save both time and effort for developers of games and virtual reality systems and could even benefit movie animators that work with computer generated graphics. While the focus for this project lies on bleeding effects, our methods can also be applied for various other applications such as crying or sweating. The main problem we would like to point out is the potential unstability of SPH-fluids. Creating stunning visual effects requires a lot of parameter tweaking to get the desired effects without volatile fluid behavior. Still, using a physics-based fluid for bleeding effects certainly has great potential in various applications. SPH has great potential - despite its flaws - and with recent technological hardware advances we expect that in the near future SPH will become a much more popular choice in real-time fluid simulation than it has been previously.

6 Future Work

The most interesting future work is to experiment with full body fluids. So far we have only experimented with fluid effects aimed at certain subparts of the character mesh. Using the right techniques, our methods could also be used to represent virtual characters that completely consist of a fluid such as mud or slime. Foremost this would require emitter placement that is (more or less) evenly distributed over the entire mesh, similar to the work done by Xu et al. [1]. The consequence is that there will be a (very) large number of particles present in the fluid. Because of the adhesion force performance, as it stands, it might not be feasible to have the full body fluid and adhesion effects at the same time. A possible solution could be to reduce the amount of emitters/particles and increasing the size of the billboards that represent the particles, even though it is likely that the visual quality would suffer. Other useful future work would involve optimization of the used methods such as the adhesion force. Building the collision shape is one of the biggest bottlenecks for using animated meshes, so either improving this process or finding a different technique could also prove very valuable.

References

[1] Tianchen Xu, Wen Wu, and Enhua Wu. Real-time generation of smoothed-particle hydrodynamics-based special effects in character animation. *Computer Animation and Virtual Worlds*, 2013. ISSN 1546-427X.

[2] R.A. Gingold and J.J. Monaghan. Smoothed particle hydrodynamics - theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181:375–389, November 1977.

[3] L. B. Lucy. A numerical approach to the testing of the fission hypothesis. *Astronomical Journal*, 82: 1013–1024, December 1977. doi: 10.1086/112164.

[4] T-C Xu, W Wu, and E-H Wu. Real-time character-driven motion effects with particle-based fluid simulation. In *Proceedings of the 26th International Conference on Computer Animation and Social Agents, CASA '13*, pages 7:1–7:10. Sabanci University Press, May 2013.

[5] Ogre - open source 3d graphics engine. URL `http://www.ogre3d.org`.

[6] Bullet physics library. URL `http://bulletphysics.org/`.

[7] Bullet-fluids for bullet physics 2.81. URL `https://github.com/rtrius/Bullet-FLUIDS`.

[8] Nadir Akinci, Gizem Akinci, and Matthias Teschner. Versatile surface tension and adhesion for sph fluids. *ACM Trans. Graph.*, 32(6):182:1–182:8, November 2013. ISSN 0730-0301.