Universiteit Utrecht

University of Utrecht

# Simulation of Water Droplets using Metaballs on mobile phones

Master Thesis

Thesis number ICA- 3722252

*Author:*
Cindy Müllenmeister

*Supervisor:*
Drs. Arno Kamphuis

August 2012

**Abstract**

In the game Hydrotilt, the player controls a water droplet that consists of metaballs. This droplet does not have all the physical abilities of a water droplet in the game. For a newer version of Hydrotilt meant to run on mobile phones, Codeglue seeks for a realistic water droplet simulation using metaballs. Earlier researches of water droplet simulation used high computational resources device. Mobile phones on the other hand are low resources devices.
Therefore, in the scope of this thesis we implemented in an experiment set-up a simulation of water droplets using metaballs for mobile phones. In this implementation, you have two kinds of water droplets. The first one is a static water droplet that waits until it is close enough to another droplet for merging. The other kind of droplet is a moving water droplet. This droplet moves according to the tilt of the mobile phone just like in the game Hydrotilt.
Furthermore, metaballs need another algorithm to be able to generate the surface. In our implementation, two algorithms were used for that: octree and marching cube algorithm. Octree is slower than the marching cube algorithm, but needs less memory which is one of the low resources on mobile phones.
In the simulation of the water droplet, the moving water droplet will change its shape while moving from a sphere-like shape to a water droplet like shape. Next to this, there are two techniques implemented to calculate the movement of the droplet. The first one is a following one that behaves like a spring and the second one is an interpolation of the movement.
Testing of the experiment set-up showed that the simulation runs successfully on mobile phones. Furthermore, the testing also showed that there is a tradeoff between visual graphics of the water droplets and the performance. Better visual graphics (for example through high resolution) results in lower performance and for high performance the visually of the droplet needs to be reduced.

# Contents


**1 Introduction** **1**
- 1.1 Previous Work . . . 2
  - 1.1.1 Water Droplets . . . 2
  - 1.1.2 Metaball . . . 4
- 1.2 Outline . . . 6

**2 Techniques and Algorithms** **7**
- 2.1 Surface . . . 7
  - 2.1.1 Marching cube algorithm . . . 8
  - 2.1.2 Octree . . . 9
- 2.2 Movement . . . 9
  - 2.2.1 Following . . . 13
  - 2.2.2 Interpolation . . . 14
- 2.3 Contact angle . . . 17

**3 Experiment set-up** **19**

**4 Octree vs. Marching cube algorithm** **23**
- 4.1 Result . . . 24

**5 Interpolation vs. Following** **27**
- 5.1 Visual observation . . . 27
  - 5.1.1 Result . . . 28
- 5.2 Speed of the calculation . . . 28
  - 5.2.1 Result . . . 29
- 5.3 Conclusion . . . 29

**6 Static Water droplets** **31**
- 6.1 Number of static water droplets . . . 31
  - 6.1.1 Results . . . 32
- 6.2 Sink in . . . 32
  - 6.2.1 Results . . . 32
- 6.3 Merging . . . 34
  - 6.3.1 Results . . . 35

**7 Resolution** **37**
- 7.1 Octree . . . 37
  - 7.1.1 Results . . . 38
- 7.2 Marching cube algorithm . . . 38
  - 7.2.1 Results . . . 38

**8 Performance** **41**

**9 Discussion** **45**

**10 Conclusion** **47**

**A Class Diagram and implementation** **51**
A.1 Surface . . . . . 55
A.2 Movement . . . . . 56
A.3 Sink in . . . . . 57
A.4 Water Droplet . . . . . 58

**B Test Environment** **61**

# List of Figures


1.1 Screenshots of the game Hydrotilt [3] . . . . . 1
1.2 Plots of the different density functions . . . . . 5
1.3 Examples of surface of two metaballs . . . . . 6

2.1 The cube representation and the tree presentation of an octree. . . 9
2.2 Shape of a moving water droplet . . . . . 10
2.3 A water droplet on a inclined plane with indications of the gravity . 10
2.4 Water Droplet with a slope degree of 45 . . . . . 12
2.5 Generated shape of a water droplet with a slope degree of 0 and 1 . 12
2.6 The moving direction of the water droplet changes with Interpolation. 15
2.7 The rotation possibilities of the droplet with marked area where the interpolation will not be used. . . . . 16
2.8 A water droplet on a solid surface . . . . . 17

3.1 Moving the water droplet by tilting the mobile and changing the moving direction. . . . . 20

4.1 This plot shows the measured calculation time for every the marching cube algorithm in ten runs on five mobile phones . . . . . 26

6.1 This plot shows the time needed until the application runs normal with the static water droplets. . . . . 34

7.1 This plot shows the calculation time of the octree for different depths. 39
7.2 This plot shows the measured calculation time for every cube size and the standard deviation.All the times are reported in milliseconds. 40

8.1 The plot shows the measures calculation time for every cube size using the following technique and a grey area marks the area where a good performance feeling is reached. . . . . 42
8.2 The plot shows the measures calculation time for every cube size using the interpolation technique and a grey area marks the area where a good performance feeling is reached. . . . . 43

A.1 The class diagram of the whole implementation. . . . . 52
A.2 The first half of the class diagram of Figure A.1. . . . . 53
A.3 The second half of the class diagram of Figure A.1. . . . . 54
A.4 Main structure . . . . . 55
A.5 Implementation structure of the Surface . . . . . 55
A.6 Classes structure of the complete implemented Surface . . . . . 56
A.7 Class structure of the movemenet implementation . . . . . 57

A.8 Changing the tilt of the mobile to change the moving direction of the water droplet in the opposite direction. . . . . 58
A.9 Class structure for the class water droplet . . . . . 59

B.1 The test environment that was used for the implementation and the experiments. . . . . 61

# 1 Introduction

In 2008, Codeglue released a game called Hydrotilt for the iPhone and iPod Touch [3]. Hydrotilt is a 3D puzzle where the user needs to guide a water droplet through an environment. Figure 1.1 shows screenshots of this game. The water droplet

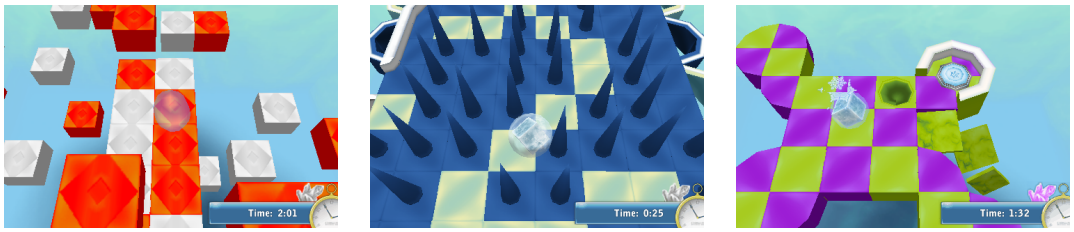



Figure 1.1: Screenshots of the game Hydrotilt [3]

is moved and controlled with the tilt function of the iPhone/iPod Touch. For example lifting the left side of the iPhone will let the water droplet move to the right.
Furthermore, the water droplet is able to change its liquid state to a solid state (ice) or a gaseous state (water vapor), enabling the user to overcome obstacles which are impassable for liquid.
The water droplet of Hydrotilt was implemented using two metaballs, according to Codeglues programmer Harald Maassen. The metaballs were connected to each other with a spring, so that the water droplet would be able to change shape when the movement speed increases. The physics used in Hydrotilt were standard game physics with sphere colliders for the metaballs. The sphere colliders were placed a little higher than the metaballs to fake the impression of water on solid ground.
In this old version of Hydrotilt, merging with static water droplets was not possible. Also, the physics used in Hydrotilt is unable to mimic the full physical behavior of real water droplets. Because of this, the water droplet is not able to react realistically in all situations.
Adding more physically behavior to the water droplet like merging would make it possible to widen the game and offer new possibilities for levels.

Codeglue plans to develop a new version of Hydrotilt with more realistic and physical behavior of water droplets. This planned new version of Hydrotilt should run on mobile phones that have low resources.
The research on the simulation of water droplets focuses on the shape and physical behavior of water droplets. For example the physical-based method in the research of Wang et. al. [17]. Some of these methods, for example the research of Yu et. al. [21], also use metaballs. All these techniques were developed for devices with high computational resources. Mobile phones on the other hand have low computational resources.
This all leads us to our main research question:

> How to implement water droplets based on metaballs with the right behavior on a device with low resources?

## 1.1 Previous Work

The simulation of water droplets and metaballs are not new research fields in computer graphics. There is much research on water droplets and also on the use of metaballs. In the next Sections we will outline the most important methods and techniques.

### 1.1.1 Water Droplets

For at least two decades, the simulation of water has been an active field in computer graphics and many different methods have been developed since then. Earlier methods, developed in the 1980s, concentrated on the simulation of large bodies such as oceans and seas. Iglesias et. al. [7] gives an overview of the earliest methods that were developed in the 1980s and the 1990s. In these simulations, particle systems were often used to simulate the foam and spray that is generated when water breaks on objects.
Only later, in the early nineties, researchers started to develop methods that concentrated on the simulation of smaller bodies of water such as droplets. The first methods were modifications of previous particle systems to simulate the behavior of water droplets, like the method of Kaneda et. al. [9]. This method is used to simulate water droplets on a surface based on a particle system. Kaneda et. al. further focused on the behavior of water droplets and developed methods using not only particle systems but also subdivision of meshes to simulate the water droplets [8].The concentration of their work was on the animation itself and not on performance. Therefore, the performance of the method is not mentioned in their research. However, subdivision of the mesh would require much calculation

power due to its complexity.
Fournier et. al. [6] mainly used a mass-spring system to model the motion and shape of a droplet. In this method, they also considered the surface tension and volume conservation of water droplets. However, the merging of two or more water droplets, also referred to as the merging process, did not work successfully in the simulation.
Yu et. al. [21] concentrated on the shape of the water droplets, but instead of using a mass-spring system, they used metaballs. The resulting shapes of the water droplets look quite realistic. Furthermore, the merging process was successful, unlike the merging process of Fournier. With the focus on the shape and behavior of the water droplet, the performance of the method was neglected. The computers used by Yu had fewer resources than the computers nowadays and are therefore slower. It might be possible that the technique from Yu et. al. is able to run on low resources devices such as mobile phones.
Tong et. al. [14] extends the method of Yu et. al. to enable volume-preservation of water droplets. Just like Yu, Tong did not take a closer look at the calculation time and the performance.
The goal of most of these methods was to simulate the behavior and the looks of water droplets as plausibly and realistically as possible, ignoring the physics at least partially. In 2005, Wang et. al. [17] developed the first physics-based method for simulation of water droplets. The physics of the method are mainly based on the physical description of water droplets by Gennes [4]. The resulting water droplets are realistic but require a high calculation time according to Chen et. al. [2].
Chen et. al. [2] developed a method to make the merging process smoother. The method is based on the method of Yu et. al. and uses a modified version of metaballs. The physics used for the motion of the water droplets is a highly simplified version of the physical-based method of Wang et. al. because that method was too computationally expensive for real-time rendering.
The latest method for the simulation of water droplets is by Zhang et. al.[22]. It is a model that is capable of running in real-time using implicit surface generation. The physics of water droplets are also considered in this method. The behavior of the water droplet is based on the physics of droplets. The implicit surface is generated using mesh splitting, merging and optimization instead of metaballs.

Of all these techniques, we will take a closer look at only two of them. First the method of Yu et. al. [21] because of the simplicity of their calculation and the use of metaballs. Metaballs are also used in the research of Tong et. al. [14], but the calculations introduced there seem to complex to be of use in the simulation introduced in this thesis. Second is the technique by Zhang et. al. [22]. This

method focuses on physics of the water droplet and not only on the shape like the method of Yu. Nevertheless, this technique does not make use of metaballs. Therefore, only part of this method is relevant for the simulation introduced in this thesis.

### 1.1.2 Metaball

Not only the previous work of water droplets are of interest for the simulation of the water droplet introduced in this thesis, but also the previous work and the introduction of the metaballs.
Metaball are used to model implicit surfaces. Instead of modelling a surface by a mesh a mathematical description of the shape is used [2]. Blinn was the first to introduce metaballs [1]. He did not refer to them as metaballs, but as "*blobs*". Nishimura et. al.[13] , Murakami et. al. [11] and Wyvill et. al. [19] improved the technique from Blinn and they coined the terms *metaball* and *soft objects*. Since then, metaballs were often used to model soft objects like clouds and water droplets.
For the modeling of water droplets the surface of metaballs is defined by the points that satisfy the following equation:

$$f(x,y,z) = \sum_{i=0}^{n} q_i f_i - T_0 = 0 \tag{1.1}$$

where $n$ is the number of metaballs, $T_0$ is a threshold, $q_i$ is the maximum density factor of metaball $i$ and $f_i$ is the density function of metaball $i$.
There exist many different forms of density functions that can be used in the equation for generating the shape of a water droplet. The density function from Blinn is the following:

$$f_i(r) = e^{(-ar^2)} \tag{1.2}$$

where $r$ is the distance from point (x, y, z) to the center of the metaball.

The density functions from Murakami (equation 1.3), Nishimura (equation 1.4) and Wyvill (equation 1.5) are the most commonly used density functions for generating the shape of water droplets.

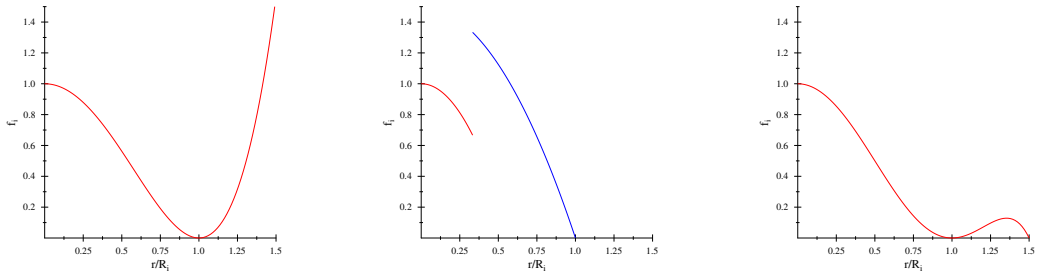
$$f_i(r) = \left(1 - \left(\frac{r}{R_i}\right)^2\right)^2 \tag{1.3}$$

$$f_i(r) = \begin{cases} 1 - 3(\frac{r}{R_i})^2 & \text{if } \frac{R_i}{3} \geq r \geq 0 \\ \frac{3}{2}\left(1 - (\frac{r}{R_i})^2\right) & \text{if } R_i \geq r \geq \frac{R_i}{3} \end{cases} \tag{1.4}$$

$$f_i(r) = -\frac{4}{9}\left(\frac{r}{R_i}\right)^6 + \frac{17}{9}\left(\frac{r}{R_i}\right)^4 - \frac{22}{9}\left(\frac{r}{R_i}\right)^2 + 1 \tag{1.5}$$

where $r$ is the distance between the center of the metaball and point (x, y, z) and $R_i$ is the influence radius of metaball $i$.

The three density functions seem to be very different from each other. To see



(a) Plot of Murakamis density function (1.3)

(b) Plot of Nishimuras density function (1.4)

(c) Plot of Wyvills density function (1.5)

Figure 1.2: Plots of the different density functions

why those different functions could be used as density functions to generate surface of water dropelts, we take a closer look at the plots of those functions. The plot of Murakamis function (see Figure 1.2a) and the plot of Wyvills function (see Figure 1.2c) seem to be exactly the same at the beginning until the function reaches the point where $y = 0$. From this point on, the functions behave differently. Nishimuras function is divided into two function bound by a condition. This explains the jump in the plot as seen in Figure 1.2b. Beside this jump in the function, it is similar to Wyvills function and Murakamis function. Therefore, we can say that all three density function seem to describe the same pattern (at least at the beginning of the graph) and therefore should describe the same shape when used to calculate the surface of metaballs.
The shape of one metaball is marble-like. When modeling more than one metaballs, the shape of it depends on the distance between both metaballs. For example, modeling two metaballs would result in two round marble-like surfaces when the distance between them is high enough. If the metaballs are closer to each other, they seem to merge like water do as you can see in Figure 1.3.

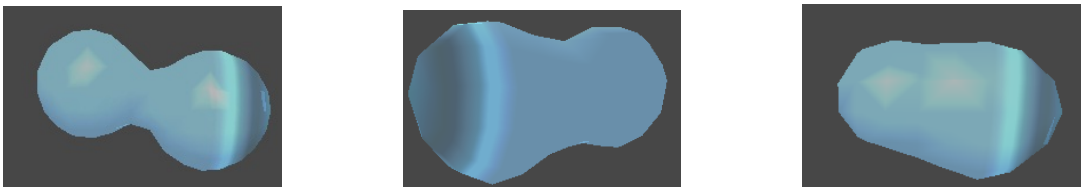


Figure 1.3: Examples of surface of two metaballs

## 1.2 Outline

This thesis is outlined as follows: First, Chapter 2 shows all the techniques and algorithms with made changes that will be used for the simulation of water droplets introduced in this thesis.
To see how this simulation of water droplets works on mobile devices, experiments need to be conducted. The setup for the experiments is shown in Chapter 3. The experiments were conducted in Chapter 4 to Chapter 8. We then discuss the results of these experiments in Chapter 9. At last, the research questions are answered and a conclusion is drawn in Chapter 10.

# 2 Techniques and Algorithms

The water droplet of Hydrotilt is already able to run on an iPhone which is a device with lower resources than a computer. Nevertheless, the water droplet in Hydrotilt still does not behave as natural as some of the water droplets of other researches and miss some of the physical abilities such as merge with another water droplet. The method for simulating water droplets introduced in this thesis aims at generating behavior for water droplets that resemble real-life droplets as close as possible. For this, we will take a closer look at some of the algorithm and techniques we want to use in this thesis. These techniques were all designed and implemented on computers which offer more resources than mobile phones. Therefore, some of those techniques and algorithm we want to use need to be changed.
The algorithm and techniques were all designed with different goals in mind. Yu et. al. [21] for example concentrates more on the form of the water droplet while the research of Tong et. al. [14] concentrates more on the volume preservation of the water droplets. Therefore, we will discuss these techniques and algorithm and their changes in the following Sections: surface, movement and contact angle.

## 2.1 Surface

The surface of metaballs consists of all the points that satisfy the equation 1.1 as described in Section 1.1.2. In researches the surface of the metaballs is also referred to as isosurface. An isosurface is defined as a surface that satisfies the following equation

$$F(P) = \alpha \tag{2.1}$$

where $\alpha$ is an isovalue. Therefore, the isovalue of the metaballs would be 0 according to the equation 1.1. The equation can be rewritten to the following equation:

$$\sum_{i=0}^{n} q_i f_i = T_0 \tag{2.2}$$

where $n$ is the number of metaballs, $T_0$ is a threshold, $q_i$ is the maximum density factor of metaball $i$ and $f_i$ is the density function of metaball $i$.

With this, the isovalue is now the value of the threshold instead of 0.
For the calculation of this surface, different algorithms were developed. In the scope of this thesis, two of these algorithms were considered: the marching cube algorithm and the octree.

### 2.1.1 Marching cube algorithm

According to Newman et. al. [12] the marching cube algorithm is a popular but not the oldest algorithm to extract the isosurface.
The basic marching cube algorithm consists of a "*cuberille grid*"[10], a grid made out of cubes. For every corner of every cube in the grid, the algorithm calculates the isovalue and uses a lookup table to estimate the surface belonging to that cube [10]. There exist 256 different surfaces that a cube can contain [12]. Therefore, the lookup table has 256 entries. Fifteen of these surfaces are the so called basic topologies because the others are reflective and/or rotational symmetry to those [12].
Going through the whole grid to look at every cube in the grid is an expensive operation. Triquet et. al. [15] developed an optimization of the marching cube algorithm which foregoes checking every cube in the grid. In their method, they first search for one cube that is filled with a surface. From this cube, they take a look at all the neighbors. The neighbors that are filled with the surface are then visited and the procedure is repeated until the whole surface is found. A so-called timestamp is tagged to every visited cube, so that a cube is not calculated twice. This optimization helps to make the marching cube algorithm faster, but the storage of the grid itself could cost much memory. Mobile phones have less memory available than personal computers. Therefore, we have to be careful that the marching cube algorithm does not use too much memory.
The data structure from Wyvill et. al. [19] is similar to the marching cube algorithm. Because of the similarity and because the optimization of the marching cube is far more recent than the structure of Wyvill, we did not consider the implementation of the data structure of Wyvill.
Most research papers about water droplets and metaballs do not mention which algorithm they use for surface calculation. We assume that some of the researches use the marching cube algorithm or the data structure from Wyvill because Wyvills density function is used in them such as in the research of Yu et. al. [21].

### 2.1.2 Octree

The octree describes (as the name already indicates) the object in a tree structure. The starting node of the tree, the root, contains the whole object in a cube. This cube is split into eight smaller cubes, which are represented by the nodes in the next level [5].
The cubes that contain part of the surface will be further divided into 8 smaller cubes. Cubes that are completely inside/outside of the object (and thus contain no surface) will not be divided any further [5]. Figure 2.1 shows such an octree structure. Furthermore, the division of the cubes also stops if the self defined maximum depth of the tree is reached.
The depth of the tree defines the resolution of the object. The more levels (depth) the tree has, the higher the resolution of the object will get.

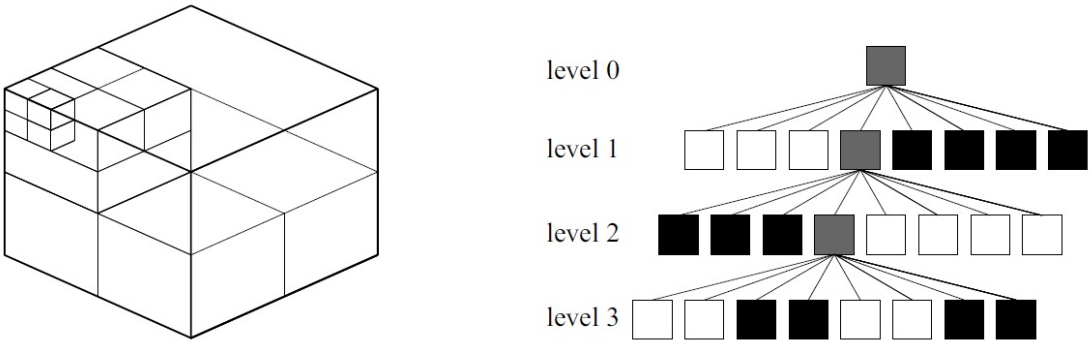
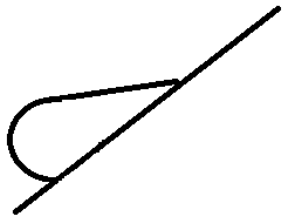



Figure 2.1: The cube representation of an object (left) with the tree presentation of the octree (right [5, fig. 8.5]. The black cubes of the tree are completely inside of the cube, the white cubes are completely outside of the object and the grey once contain surface.

## 2.2 Movement

We want the simulation of the water droplet introduced in this thesis to also be able to move around. A real water droplet is also able to move due to the gravity. When you observe a water droplet moving down a window, the first thing you notice is that it does not have a sphere-like shape but a so called water droplet shape as seen in Figure 2.2. Yu et. al. [21] developed a way to deform the metaball so that it would change the shape to the water droplet shape according to the slope degree of a plane it is on.

(a) Photo of a moving water droplet



(b) Drawing of Water droplet form when it moves

Figure 2.2: Shape of a moving water droplet

In the research, Yu et. al. deform the points of the metaball before passing it to the equation 1.1. For this deformation, they split the point itself into the vectors $\vec{u}$, $\vec{v}$ and $\vec{w}$

$$\vec{u} = (x, 0, 0),\ \vec{v} = (0, y, 0),\ \vec{w} = (0, 0, z) \tag{2.3}$$

where x, y and z are the coordinates of the original point.

These vectors are then scaled to change the shape of the water droplet. For the scaling Yu et. al. takes the gravity and the friction forces into consideration as you can see in Figure 2.3. To calculate the scaling factor for the vectors $\vec{u}$, $\vec{v}$

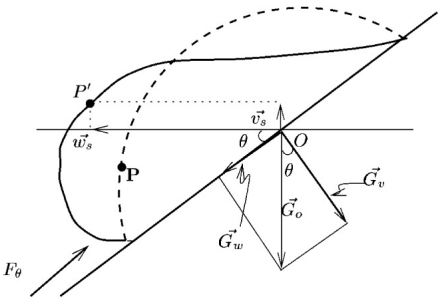



Figure 2.3: The water droplet on a plane with slope degree $\theta$ according to Yu [21, fig. 17]. The dotted line indicates the form of the original metaball. $F_\theta$ is the friction force, $G_o$ is the gravity, $G_v$ is the gravity component perpendicular to the plane and $G_w$ is the gravity component parallel to the plane.

and $\vec{w}$, the friction force has to be calculated first. This is done with the following

equation:

$$F_\theta = sign(w_z)||G_w|| - \frac{sin\theta||\vec{v}||}{R} \tag{2.4}$$

where $R$ is the radius of the metaball, $\theta$ the slope degree of the plane and *sign($w_z$)* the sign of the z value of the vector $\vec{w}$.

$$h(r_w) = \frac{7}{8}r_w^3 + \frac{9}{4}r_w^2 + \frac{15}{8}r_w \tag{2.5}$$

In equation 2.5 $r_w$ is the ratio of the magnitude of the vector $\vec{w}$ and the radius of the metaball and is calculated as follows:

$$r_w = \frac{-(w_z)}{R} \tag{2.6}$$

where $R$ is the radius of the metaball and *sign($w_z$)* the sign of the z value of vector $\vec{w}$.

These two equations are then used to calculate the scaled vectors $\vec{u_s}$, $\vec{v_s}$ and $\vec{w_s}$ from the vectors $\vec{u}$, $\vec{v}$ and $\vec{w}$ according to:

$$\vec{u_s} = \vec{u}(1 - \frac{||G_v||}{3}) \tag{2.7}$$

$$\vec{v_s} = \vec{v}(||G_v|| + h(r_w)sin\theta) \tag{2.8}$$

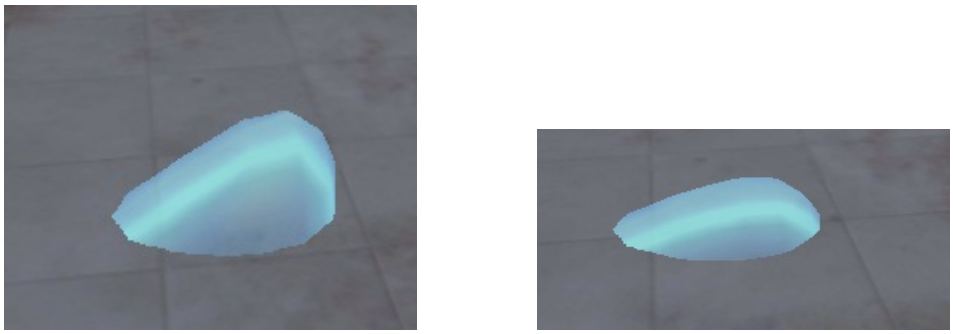
$$\vec{w_s} = \vec{w}(1 - \frac{||G_v|| + ||G_w|| - F_\theta}{3}) \tag{2.9}$$

The new coordinates of the surface point can now be determined by adding the calculated vectors $\vec{u_s}$, $\vec{v_s}$ and $\vec{w_s}$ together.
With this scaling of the surface points, the shape of the water droplet changes to the water droplet form according to the degree of the slope of the plane that lies between 0 and 90 degree.
In an earlier paper of Yu et. al. [20] the calculation of $\vec{v_s}$ is slightly different:

$$\vec{v_s} = \vec{v}(1.3 + ||G_v|| + h(r_w)sin\theta) \tag{2.10}$$

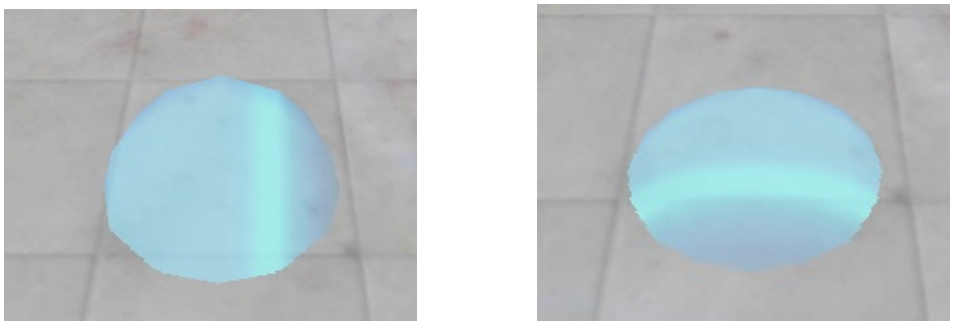
Here an extra value of 1.3 is added to the scaling value before scaling $\vec{v}$ with it. So, we need to estimate whether or not we need to use this extra value in our simulation of water droplets.
To see which one of these two equations to use (2.10 or 2.8), the resulting shape those two generate need to be compared to each other. First, the equations are tested with a slope degree of 45 as seen in Figure 2.4. This shows clearly that a shape generated with equation 2.8 generates a peak which seems unnatural for

a water droplet. Using equation 2.10 results in a less high peak which is a more natural shape of the water droplet.
Testing the equation 2.10 further shows that changing the slope degree from 0 to 1 result in instantly flattening of the sphere-like form. This change in shape is visually not realistic.
Because of these two observation, we decided to use an extra value as in equation 2.10. This extra value we will use will not be not static, but ranges from $-\frac{1}{3}$ to 1.3 depending on the slope degree. This way, the change of form from one degree to another is less obvious and also the shape looks natural when a slope degree of 45 is used.
Although the droplet is now able to change its form, it will only face one direction.



(a) using equation 2.8 (b) using equation 2.10

Figure 2.4: Water Droplet with a slope degree of 45



(a) Shape of a water droplet with a slope degree 1 2.8 (b) Shape of a water droplet with a slope degree 1 2.10

Figure 2.5: The shape of the Water droplet while changing the slope degree from 0 to slope degree 1 using the equation 2.10

To be able to move it around, the water droplet needs to be able to face other

directions as well. Furthermore, the water droplet needs to be able to change from one plane with a slope degree to a plane with another slope degree because of for example a ramp. This means, the water droplet needs to be able to face up or down according to the visual plane.
Therefore, the solution for this is to align the coordinate frame with the orientation of the water droplet before calculating the new water droplet. First, the coordinate frame will be rotated around the y-axis to face the desired direction:

$$x_{new} = sin(\alpha)z + cos(\alpha)x \tag{2.11}$$

$$z_{new} = cos(\alpha)z - sin(\alpha)x \tag{2.12}$$

$$y_{new} = y \tag{2.13}$$

where $\alpha$ is the angle between the facing direction of the water droplet and the wanted facing direction and $z_{new}$, $x_{new}$ and $y_{new}$ are the rotation of the coordinate frame.
In addition to this rotation, the coordinate frame also needs to be rotated around the x axis, so that the water droplet will face up or down and lies this way on the plane. The rotation is done by recalculating the y and z coordinates of the coordinate frame with equations 2.14 and 2.15.

$$y_{new} = cos(\beta)y - sin(\beta)z \tag{2.14}$$

$$z_{new} = sin(\beta)y + cos(\beta)z \tag{2.15}$$

where $\beta$ is the angle between the surface of the horizontal floor and the incline surface (for example a ramp or a slide) on which the water droplet is moving. Furthermore, this angle also needs to consider the moving direction of the water droplet (upwards or downwards).

Now, with these calculations, the water droplet is able to change the shape to the water droplet shape and even face the desired direction.
Besides changing the shape, the water droplets need to move like a real water droplet. The previous researches that uses metaballs (see Section 1.1) , do not state how to move the water droplet. Yu et. al. [21] only considered static water droplets without any movement.
Furthermore, just moving the water droplet and turning it straight with the new direction does not look natural. Therefore, we came up with two different methods to move the water droplet: Interpolation and Following.

### 2.2.1 Following

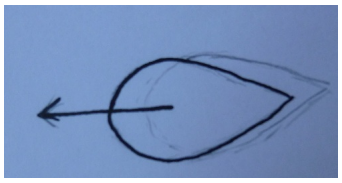
When you put a water droplet on a plane and let it move in one direction and then change the moving direction of the droplet, you can observe that one part of

the water droplet will follow the direction changing directly, while the other part of the water droplet will after some time. With the Following technique, we try to simulate this behavior.
For this Following technique, the water droplet itself consists of two metaballs. The two metaballs will react to each other as if a spring is between them. This means that the second metaball will follow the movement of the first one step for step after a little delay. The faster the first one moves, the more distance will be between the two metaballs, until a maximum distance is reached. The second metaball will, when the maximum distance is reached, follow faster so that the distance will not be increased any further.

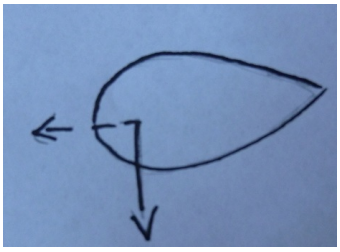
### 2.2.2 Interpolation

As the following technique needs two metaballs to work, the interpolation technique only needs one metaball for the water droplet. With one metaball it is not possible to simulate the partly directional changing move of a water droplet. Instead of that, the change of direction will be interpolated. This means that if the turning of the water droplet is greater than the small angle $\alpha$ then the water droplet will turn in steps of size $\alpha$ and move every step a little forward. Figure 2.6 shows the changing of the moving direction of a water droplet with interpolation steps.

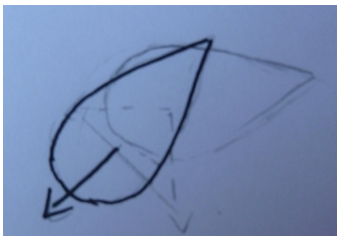
If the moving direction of the water droplet changes into the opposite direction, changing the direction step for step and letting the droplet move in a circle is not natural. Instead of that, the water droplet should change back its form and then start moving in the desired new direction. The same goes for the direction that is almost in the opposite to the current moving direction. Therefore, we define a range in the opposite moving direction (see Figure 2.7). When the desired new direction is in this range, the water droplet will first stop moving, letting this way the form of the droplet go back to the sphere-like form. As soon as the form is sphere-like again, the water droplet will immediately start to move in the new direction and change the shape again according to the new moving direction. If the new desired moving direction is outside of the grey area in Figure 2.7, the direction is changed (as mentioned before) step for step.



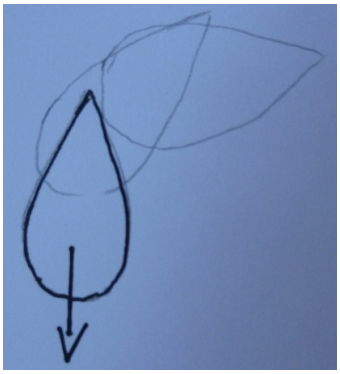
(a) The water droplet moves into the direction of the arrow.



(b) Change of moving direction. The broken arrow indicates the previous moving direction and the other arrow indicaties the new desired moving direction.



(c) The moving direction of the water droplet is changed first to an interpolated moving direction.



(d) The moving direction of the water droplet is changed again and is now moving in the desired direction.

Figure 2.6: The moving direction of the water droplet changes. Interpolation steps are used to change the direction.

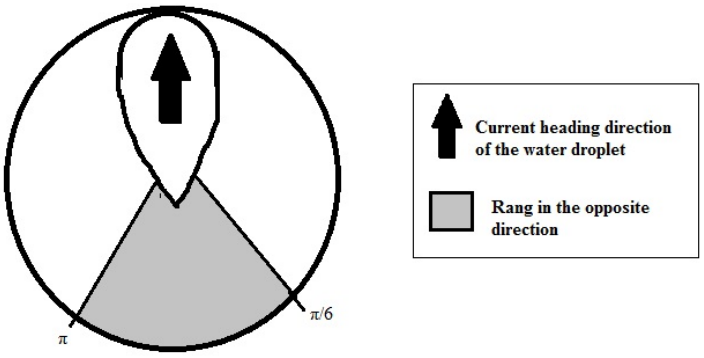



Figure 2.7: Rotation possibility of a moving water droplet. When the new direction falls into the rang (grey area), then no interpolation takes place.

## 2.3 Contact angle

Next to the moving of the water droplet, the contact angle of a water droplet needs also to be considered in the simulation introduced in this thesis.
If you put a water droplet on a solid surface, for example a glass plate, you can observe that the form of the water droplet will not remain a complete sphere, such as a marble. Instead, the water droplet will spread a little and appear to spread and sink into the surface. How far the water droplet seems to sink in depends on the contact angle. When a droplet comes in contact with a solid surface, the angle between the surface of the liquid and the solid surface is a characteristic of the liquid itself. This angle is referred to as contact angle [17] (see Figure 2.8).
According to Gennes et. al. [4] the water droplet reaches equilibrium with the

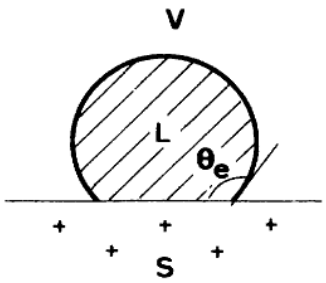



Figure 2.8: A water droplet on a solid surface [4, Figure 1a)], where $\theta_e$ is the contact angle, V is the air, S is the solid surface and L the liquid.

solid surface if the contact angle and the surface tensions satisfy the following equation:

$$\gamma_{SA} - \gamma_{SL} - \gamma cos\theta = 0 \tag{2.16}$$

where $\gamma_{SA}$ is the solid surface air tension, $\gamma_{SL}$ is the solid surface liquid tension and $\gamma$ is the surface tension.

The research of Wang et. al. [17] is about a physic-based model of water droplets. Although the physics in this model makes the simulation of water droplets behave natural, the calculation time is quite high. Furthermore, the device used for the simulation has high computational resources. Therefore, running this simulation on a low resources device such as the mobile phone would result in higher calculation time then on the high computational resources device.
To simplify this physical behavior for the use on low resources devices, we intend to give the surfaces their contact angle instead of calculating it with the surface

tensions. Just like Zhang et. al. [22] we will use the normal of the solid surface and the normals of the points of the water droplet surface to search for the points that satisfy the contact angle between the solid surface and the droplet.
Just sinking into the ground makes the water droplet appear as if part of it will disappear instead of spreading over the ground as real water droplets seem to do. Therefore, the radius of the water droplet needs to be recalculated in such a way that it visually seems as if the water droplet spreads out. What will be left of the water droplet after sinking into the ground is not the total sphere but a so-called spherical cap. This spherical cap should have the same volume as the total sphere beforehand because the water droplet should visually not lose any volume, only spread itself on the surface.
The volume of a spherical cap is calculated with the following equation [18]:

$$V = \frac{1}{3}\pi h^2(3r - h) \tag{2.17}$$

where h is the heigth of the cap, r the radius of the sphere and V the volume.
Rewriting this so that the radius can be calculated gives:

$$r = \frac{V}{\pi h^2} + h \tag{2.18}$$

where h is the height of the cap, r the radius of the sphere and V the volume.

# 3
# Experiment set-up

The techniques and algorithm from the previous Chapter were used to implement an experiment set-up.
In this experiment set-up, the main platforms for the water droplets are not computers or laptops, as was the case with previous researches, but mobile phones. Such a mobile phone can either run iOS or Android or others such as windows.
For the newer version of Hydrotilt, Codeglue prefers to implement this game in Unity with C# for the scripts. Therefore, we will use Unity and C# for the implementation of the simulation of water droplets.
The Unity development tool [16] enables publishing on both platforms: iOS and Android. Furthermore, it offers the possibility to use and add scripts to 3D content. These scripts can either be written in C# or in JavaScript. Unity also provides out-of-the-box functionalities. The scripting part and the 3D enabled world of Unity will enable the water droplet to be used in games later on. Also, it offers the possibility to add other parts.
Before implementing the techniques and algorithms of Chapter 2, a test environment has been set-up in Unity. This test environment consist of four floors, one slide and several ramps, as seen in Appendix B.
In the game Hydrotilt, the water droplet was moved through the tilt of the mobile phone. In the experiment set-up, we decided to implement this type of control as well. Therefore, in the implementation, the water droplet moves according to the tilt of the mobile phone (see Figure 3.1). The speed of the water droplet depends on how far the mobile is tilt.
As explained in Section 2.2, moving the droplet will also change the form of the droplet according to a slope degree. When the mobile is tilt, it also looks like the floor is tilt. We want to use this tilting of the floor and therefore decided to use the tilt of the mobile for the slope degree. With this, the water droplet looks in the experiment set-up as if the floor is tilt and the water droplet moves and changes form according to gravity.
Another implementation choice for the experiment set-up we made was that there are two kinds of water droplets. The first one is the moving water droplet. This

(a) Basic holding position of mobile where tilt is (almost) 0 and therefore the slope degree is also 0.

(b) Changing the moving direction into the opposite movement by changing the tilt of the mobile.

Figure 3.1: To change the moving direction into the opposite direction, the tilt of the mobile has to change. In this movement, the mobile device is held shortly in a basic. This causes the water droplet to change its form back before changing movement into the opposite direction

water droplet is able to move around according to the tilt of the phone.
The other water droplet is a static water droplet. This one is not able to move around. Instead, it waits until the moving water droplet is close enough to merge. Another difference between these two kinds of water droplets is that the static water droplet uses the contact angle to sink in. During the implementation of the experiment set-up, we noticed that the calculation of the contact angle points take too much calculation time as that this techniques could be used in the simulation for the moving water droplet. Therefore, the moving water droplet is already sunken in to half of the droplet and does not calculate the contact angle like the static water droplet does.
A more detailed description of the implementation with class diagrams can be found in Appendix A.

This experiment set-up was used to perform experiments. There exist many different mobile phones which run Android. They can have different hardware and therefore different amounts of calculation resources. Because of this, we will perform the experiments on five different mobile phones: Samsung Galaxy S2, Nexus

S, Samsung Galaxy S, HTC Desire and HTC Desire HD.

# 4

# Octree vs. Marching cube algorithm

The first two experiments are about the octree and the marching cube algorithm. The implemented marching cube algorithm and the octree can be used to calculate the surface of the water droplet. The calculation in these two algorithms differs from each other and can therefore have different effects on the water droplet on mobile phones. For the use on mobile phones it is important to know which of those two algorithms is more suitable and what influence it has on the behavior of the water droplet. To test this, we have set up two experiments.
One of the lower resources in a mobile phone is the memory. The memory in the mobile phones is more limited than that of a computer. Therefore, it is important that the algorithm doesnt use too much memory, otherwise the mobile phone could refuse to start up the simulation. This leads us to the first experiment: placing of static water droplets and look at the memory usage needed for the calculation of the surface.
Furthermore, we want to see which of these two algorithms is faster. Therefore, in the next experiment we measure the time needed to calculate the surface of the water droplet on the mobile phone.
To actually measure the calculation time, a stopwatch was implemented to measure the time between the start and the end of the surface calculation in millisecond. Because the calculation time of the merging and moving and such can differ, the total time of one experiment will be the average time of all the measured times.
As mentioned in Chapter 3, the experiments are conducted on five different mobile phones with different amounts of computational resources.
In order to compare the results of the second experiment on the different mobile phones, the water droplet will be moved along the same route. First of all, the water droplet will be moved towards one static water droplet until it is merged with that one. Then, it will be moved up a ramp from which we will let it fall of at the end. As soon as the water droplet reaches the ground, the route is finished and the test stopped.
The water droplet is moved with tilting of the mobile phone. This can lead to small deviation in the route that the water droplet will take. To compensate for

| Experiment run number | Samsung Galaxy S 2 |
|---|---|
| 1 | 39,8 |
| 2 | 38,6 |
| 3 | 37,4 |
| 4 | 38,9 |
| 5 | 36,8 |
| 6 | 39 |
| 7 | 36,1 |
| 8 | 39 |
| 9 | 39,3 |
| 10 | 35,2 |
| average | 38,01 |
| stddv | 1,5401 |

Table 4.1: The table shows the measured time for the octree algorithm, including the average of these ten runs and the standard deviation. All the times are reported in milliseconds.

this deviation, the experiment is run 10 times on every mobile phone.

## 4.1 Result

The first experiment was only conducted on the Nexus S because it already showed the difference in memory usage of the two algorithms.
Using the marching cube algorithm, it was not possible to run the simulation with more than four static water droplets. The memory usage was too high for the mobile phone resulting in termination of the simulation.
Using the octree instead of the marching cube algorithm, the Nexus S showed no problem with even ten static water droplets. This shows clearly, that the octree uses less memory than the marching cube algorithm to calculate the surface.

In the second experiment, the calculation time needed for the octree algorithm was too high making it impossible to conduct the experiment on any other mobile then the Samsung Galaxy S 2. Even on the Samsung Galaxy S 2 it was almost impossible to keep it on the route we described and conduct the experiment (see Table 4.1 for the calculation times).
The marching cube algorithm on the other hand needed less calculation time as seen in Table 4.2 and Figure 4.1.
The marching cube algorithm was fast enough to be tested on all five mobile

| Experiment run number | Samsung Galaxy S2 | Nexus S | HTC Desire | Samsung Galaxy S | HTC Desire HD |
|---|---|---|---|---|---|
| 1 | 16,2 | 48,7 | 29,6 | 38,5 | 32 |
| 2 | 15,4 | 41,6 | 29,8 | 39,1 | 26,7 |
| 3 | 16,5 | 38,9 | 28,7 | 40,6 | 27,4 |
| 4 | 16,3 | 41,1 | 28,8 | 37,4 | 26,7 |
| 5 | 15,6 | 42 | 29,3 | 39,6 | 26,3 |
| 6 | 17,5 | 38,2 | 27,6 | 38,8 | 26,4 |
| 7 | 16,7 | 45 | 28,1 | 37,6 | 25,9 |
| 8 | 16,5 | 39,7 | 29,4 | 37,5 | 26,4 |
| 9 | 16,2 | 43,4 | 29,1 | 38,1 | 25,9 |
| 10 | 15,6 | 43 | 30,2 | 36,9 | 26,6 |
| average | 16,25 | 42,16 | 29,06 | 38,41 | 27,03 |
| stddev | 0,6204 | 3,1074 | 0,7862 | 1,1415 | 1,7987 |

Table 4.2: This table shows the measured calculation time for the marching cube algorithm in milliseconds, including the average and the standard deviation.

phones.
Furthermore, the result also shows the already mentioned difference between the mobile phones and their resources because the time needed to calculate the surface is different for each of these mobile phones.

The result of these two experiment show that the octree needs less memory than the marching cube algorithm. On the other hand the marching cube algorithm is faster than the octree. Therefore, we recommend to use the octree for the surface calculation of the static water droplet as this needs to be calculated only once at the beginning. For the moving water droplet, it is better to use the marching cube algorithm instead, so that the water droplet will be able to move faster.

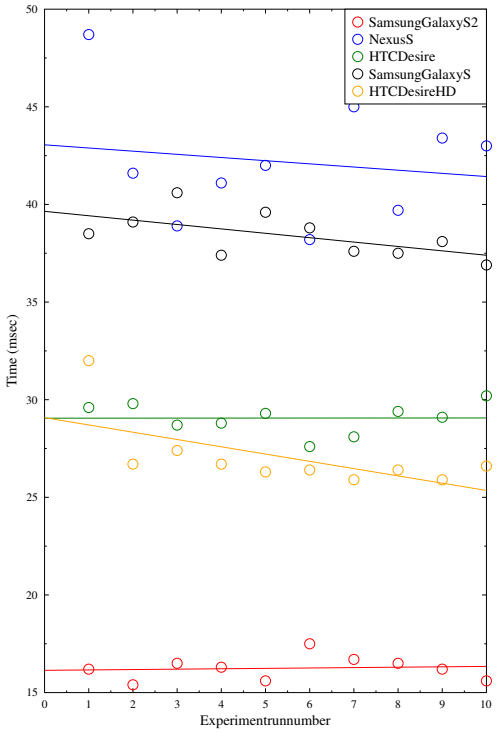



Figure 4.1: This plot shows the measured calculation time for every the marching cube algorithm in ten runs on five mobile phones

# 5 Interpolation vs. Following

Just like marching cube algorithm and octree, the interpolation technique and the following technique are two different techniques to calculate the move of the water droplet. Both techniques have advantages and disadvantages over each other. To show these, we have to conduct two experiments. The first experiment is a visual observation of the moving behavior. The second experiment is checking the speed to calculate the move of the water droplet.

## 5.1 Visual observation

The experiment "visual observation"is done to verify and observe the physical behavior of the movements of the water droplet and which one of the above mentioned techniques makes it look more realistic. For this, we move the water droplet on the mobile phone Samsung Galaxy S2. Because this experiment is only about the visual behavior and not the calculation times for the performance of the water droplet, it is not necessary to test it on more than one mobile phone.
In the previous experiment, the water droplet was moved along one specified route. In this experiment, the water droplet will be moved along almost the same route. The difference to the earlier route is that the experiment will not be stopped directly after the fall of the water droplet. Instead, the water droplet is moved to the slide (see for the position of the slide Appendix B) and slide down to the next floor. Only then the observation is stopped. Because only the moving behavior of the water droplet is within the scope of this experiment, the water droplet will not merge with a static droplet as in the previous experiment. With this, the route of the water droplet is as follows: Move to the ramp, move up the ramp, the water droplet falls down, move towards the slide and letting the water droplet slide down.
To see that the visual physic behavior is not a onetime occurrence, the experiment is repeated five times for both movement techniques.

### 5.1.1 Result

The first difference between both techniques is already visible as soon as the water droplet starts to move up the ramp. With the following technique, parts of the water droplet stays on the ground while the first part of the water droplet starts to move the ramp up, just like in real life. On the other hand, using the interpolation technique made the whole water droplet move upwards on the ramp. This makes it seems as if the water droplet flicks.
Another visual difference occurs when the water droplet falls. The water droplet changed back to the sphere-like form as soon as it starts falling using the interpolation technique. This is different when the following technique is used instead. The water droplet seems to stretch in the air, as if part of the water droplet falls slower. This is more realistic because in real life the water droplet does not change back to a sphere-like form when falling.

With these two observations, we can conclude that the following technique simulates the movement of the water droplet more realistic.

## 5.2 Speed of the calculation

The calculations of both techniques differ from each other. Therefore, the time needed to calculate the movement can also be different. This experiment is conducted to find out what the time difference is between the two movement algorithms.
First of all, a stopwatch is needed for this just like in the experiment for the octree and marching cube algorithm. The stopwatch is implemented into the simulation so that it will measure the time needed to calculate the movement in milliseconds. Furthermore, during the simulation, the movement of the water droplet is calculated multiple times. The measured calculation time can vary during the simulation. To compensate for this variation in time, the average of these calculation times is calculated and given as the result.
For this experiment, the experiment set-ups of Chapter 3 is used, including testing on five different mobiles.
The water droplet movement is simulated the same on these five devices. First, we move the water droplet towards the ramps. Then just before reaching the ramp, we let the water droplet move to the right. Just before it reaches the border of the test environment, we will move it back to the ramp again and stop the simulation just before reaching the ramp.
Because the movement is not done automatically, small differences in the movement can appear. To compensate for this, the experiment is repeated ten times

| | Samsung Galaxy S 2 | Nexus S | HTC Desire | Samsung Galaxy S | HTC Desire HD |
|---|---|---|---|---|---|
| 1 | 0,0838 | 0,0598 | 0,1225 | 0,0714 | 0,0944 |
| 2 | 0,0722 | 0,0766 | 0,1155 | 0,2622 | 0,1330 |
| 3 | 0,0524 | 0,0677 | 0,1325 | 0,0718 | 0,1012 |
| 4 | 0,0836 | 0,0716 | 0,1507 | 0,0909 | 0,1084 |
| 5 | 0,0733 | 0,0579 | 0,1190 | 0,0694 | 0,1052 |
| 6 | 0,0631 | 0,0564 | 0,1626 | 0,0745 | 0,0902 |
| 7 | 0,0666 | 0,1174 | 0,1282 | 0,0716 | 0,0830 |
| 8 | 0,0967 | 0,0741 | 0,1367 | 0,0848 | 0,0853 |
| 9 | 0,0772 | 0,1012 | 0,1696 | 0,1007 | 0,1021 |
| 10 | 0,0837 | 0,0990 | 0,1651 | 0,0905 | 0,0960 |
| average | 0,07526 | 0,07817 | 0,14024 | 0,09878 | 0,09988 |
| stddev | 0,012630668 | 0,020809562 | 0,020233976 | 0,058409394 | 0,0143015 |

Table 5.1: The table shows the measured time of the following technique for ten runs, including the average of these ten runs and the standard deviation. All the times in this table are in milliseconds.

on every mobile phone.

### 5.2.1 Result

As you can see in Table 5.1, the average calculation times for the following technique for all five mobile phones lies between 0,07ms and 0,14ms. On the other hand, the interpolation technique has an average calculation time between 0,01ms and 0,04ms as you can see in table 5.2. This clearly shows that the following technique needs more time to calculate the movement than the interpolation technique.

## 5.3 Conclusion

As we found out in these two experiments, the follow technique simulates a more realistic moving behavior than the interpolation technique. On the other hand, the interpolation technique is faster with the calculation. Therefore, the choice on which of these two techniques to use depends on the goal one want to reach: more realistic or faster.

| | Samsung Galaxy S 2 | Nexus S | HTC Desire | Samsung Galaxy S | HTC Desire HD |
|---|---|---|---|---|---|
| 1 | 0,0098 | 0,01597 | 0,0606 | 0,0255 | 0,0229 |
| 2 | 0,0233 | 0,01897 | 0,0423 | 0,0324 | 0,0280 |
| 3 | 0,0104 | 0,02247 | 0,0401 | 0,0208 | 0,0300 |
| 4 | 0,0107 | 0,02693 | 0,0313 | 0,0217 | 0,0316 |
| 5 | 0,0111 | 0,03040 | 0,0408 | 0,0249 | 0,0353 |
| 6 | 0,0251 | 0,02464 | 0,0424 | 0,0179 | 0,0344 |
| 7 | 0,0155 | 0,02643 | 0,0409 | 0,0225 | 0,0217 |
| 8 | 0,0196 | 0,02735 | 0,0387 | 0,0262 | 0,0269 |
| 9 | 0,0118 | 0,01829 | 0,0523 | 0,0227 | 0,0315 |
| 10 | 0,0212 | 0,02561 | 0,0451 | 0,0204 | 0,0423 |
| average | 0,01585 | 0,023706 | 0,04345 | 0,0235 | 0,03046 |
| stddev | 0,005922321 | 0,004637246 | 0,007975832 | 0,004022161 | 0,006095025 |

Table 5.2: The table shows the measured time of the interpolation technique for ten runs, including the average of these ten runs and the standard deviation. All the times are reported in milliseconds.

# 6 Static Water droplets

Next to the choice of implementing two different algorithms for the surface calculation and the movement, the water droplet was split in the implementation into static water droplets and moving water droplets as described in Chapter 3.
The static water droplets are implemented slightly different than the moving water droplets. This leaves us wondering what the impact of the static water droplets are on the whole simulation and the limitations. To investigate this, further experiments were necessary. To be more precise, three experiments are necessary.
The first experiment is to test the limitation of the number of static water droplets in the simulation. Second is an experiment about the sink in implementation of the static water droplet and the last experiment is about the merging with the moving water droplet. In all three experiments, the experiment set-up of Chapter 3 is used, including testing on five different mobile phones.

## 6.1 Number of static water droplets

In the simulation, it is possible to place more than one static water droplet. Because these water droplets need to be calculated at the start of the simulation, they can be a burden to the memory of the mobile phone. Furthermore, the calculations of the static water droplets need process time. Therefore, every added static water droplets slows down the movement of the moving water droplet. Because the calculations of the static water droplets take place at the beginning of the simulation, the simulation is only slows down there and speeds up again after some time.
In this experiment, one till twenty static water droplets are placed in the simulation and run on each of the five mobile phones.
To see how long it takes until the moving water droplet is able to move normal again, a stopwatch is used. The stopwatch is started as soon as the simulation starts up and stopped as soon as the moving water droplet seems to move as smoothly as without any static water droplets.
Furthermore, the marching cube algorithm needs more memory to calculate the

surface than the octree as we saw in Chapter 4. This is why the static water droplets in this experiment use the octree for the surface calculation. As the static water droplets do not use the movement, it is not necessary to decide between the movements for them.

### 6.1.1 Results

On all the mobile phones it was possible to start and run the simulation with twenty static water droplets. It was even possible to start the simulation with 40 static water droplets on the five mobile phones. However, the time the simulation needed until the moving water droplet was able to move normal again increases with every added static water droplet. Table 6.1 and Figure 6.1 show this time in seconds for one to twenty static water droplets.
Furthermore, the measured time is different on each of these five mobile phones as seen in Table 6.1. This shows that the measured time until the moving water droplets move faster again depends on the resources of the mobile phones.

## 6.2 Sink in

In the experiment set-up, the static water droplet sinks in according to method explained in Section 2.3. The moving water droplet is not implemented with this algorithm. The sink in needs too much time for the calculation as that it would be usable for the moving water droplet.
To show how much time the sink in actually needed to be calculated, this experiment is done with one static water droplet. The static water droplet is placed near the first ramp in the simulation. A stopwatch is used in the implementation of the simulation to measure the time needed to calculate the sink in of the droplet.
Processes running in the background could influence the measured calculation time. Because of this, the experiment will be run ten times on all five mobile phones.

### 6.2.1 Results

As seen in Table 6.2, the sink in process needs at least 13 milliseconds to be calculated. The mobile phone with the slowest calculation abilities needs even 21 milliseconds in average to calculate the sink in.
This shows clearly that the sink in process demands a lot of calculation time.

| Number of Static Water droplets | Samsung Galaxy S2 | Nexus S | HTC Desire | Samsung Galaxy S | HTC Desire HD |
|---|---|---|---|---|---|
| 1 | <1 | 1 | 1 | <1 | <1 |
| 2 | <2 | 2 | 3 | 2 | 2 |
| 3 | 2 | 3 | 4 | 2 | 2 |
| 4 | 3 | 4 | 4 | 3 | 4 |
| 5 | 3 | 5 | 5 | 4 | 4 |
| 6 | 4 | 5 | 5 | 4 | 5 |
| 7 | 5 | 6 | 6 | 4 | 6 |
| 8 | 5 | 7 | 7 | 5 | 7 |
| 9 | 6 | 9 | 9 | 6 | 8 |
| 10 | 7 | 9 | 10 | 7 | 8 |
| 11 | 7 | 9 | 11 | 7 | 8 |
| 12 | 7 | 11 | 12 | 7 | 9 |
| 13 | 7 | 11 | 13 | 8 | 11 |
| 14 | 8 | 12 | 13 | 9 | 12 |
| 15 | 9 | 13 | 15 | 10 | 13 |
| 16 | 9 | 13 | 16 | 11 | 14 |
| 17 | 10 | 14 | 16 | 11 | 14 |
| 18 | 10 | 15 | 17 | 12 | 16 |
| 19 | 11 | 16 | 20 | 13 | 17 |
| 20 | 12 | 17 | 21 | 14 | 17 |

Table 6.1: The measured time needed until the moving water droplet was able to move faster again. All the times are reported in seconds.

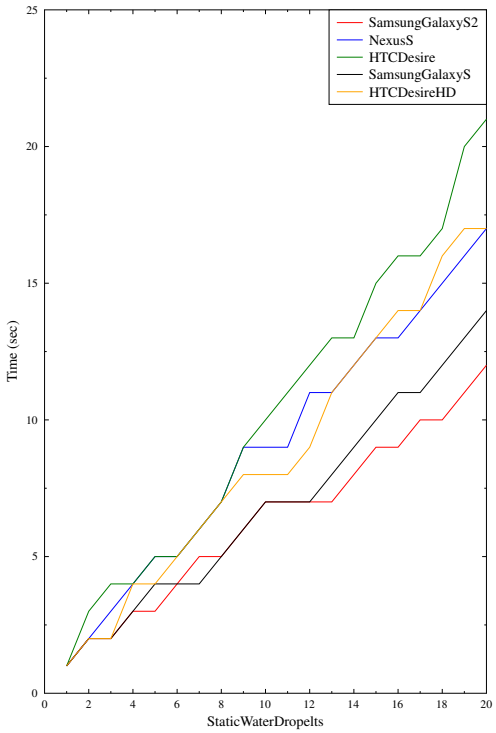



Figure 6.1: This plot shows the time needed until the application runs normal with the static water droplets.

## 6.3 Merging

Another implemented ability of the static water droplets is to merge with the moving water droplet as soon as the moving one is close enough. After merging, the moving water droplet has increased in size. The marching cube algorithm, however, only has a limited grid size. Therefore, the number of water droplet

| Experiment run Number | Samsung Galaxy S2 | Nexus S | HTC Desire | Samsung Galaxy S | HTC Desire HD |
|---|---|---|---|---|---|
| 1 | 11,05 | 12,7 | 21,7 | 12,25 | 16,5 |
| 2 | 14,1 | 12,55 | 26,15 | 12,65 | 14,65 |
| 3 | 13,45 | 12,25 | 21,65 | 12,7 | 14,9 |
| 4 | 11,6 | 13,4 | 18,45 | 13,3 | 15,2 |
| 5 | 14,65 | 13,05 | 20,95 | 12,35 | 14,85 |
| 6 | 14,85 | 12,75 | 19,75 | 12,55 | 15,05 |
| 7 | 13,05 | 12,45 | 21,25 | 12,4 | 16 |
| 8 | 16,6 | 13,55 | 19,75 | 12,7 | 15,05 |
| 9 | 14,4 | 12,7 | 19,55 | 12,85 | 15,25 |
| 10 | 14,2 | 12,7 | 19,8 | 12,7 | 15,1 |
| average | 13,795 | 12,81 | 20,9 | 12,645 | 15,255 |
| stddev | 1,611 | 0,408 | 2,123 | 0,296 | 0,564 |

Table 6.2: This table shows the measured calculatime time of the sink in for all ten runs in milliseconds, including average and standard deviation.

with which the moving water droplet can merge is limited. So in this experiment, it was tested with how many other water droplets the moving one can merge without having surface outside of the grid of the marching cube algorithm. For this experiment, the resolution of the water droplet was chosen to be low so that the water droplet is able to move as fast as possible. Furthermore, with every merg the water droplet grows. This means that the resolution would change. To prevent that, the cube size changes according to the number of calculated triangles of the surface.
With this, the water droplet is moved around the test environment and merged with the static water droplets in the environment.
The implementation is further changed, so that a log message is generated as soon as surface calculations outside of the grid take place.

### 6.3.1 Results

In this experiment, we were able to let the moving water droplet merge with twenty static water droplets. The simulation runs further without any noticeable changes in speed. Furthermore, the marching cube algorithm did not give any message that part of the surface was found outside of the grid.
At first, merging with more than six water droplets seemed to let part of the water droplet disappear when moving to the left. After changing the texture, this phenomenon disappeared. This shows that it was because of the texture and not

because of the grid size that it seemed to disappear.
This shows that the merging with twenty static water droplet is not a problem for the simulation on the mobile phones.

# 7 Resolution

Another experiment we need to conduct is the testing of the resolution.
The resolution of the water droplet is defined in its surface. With high resolution, the surface would be very smooth and with low resolution the surface would be bumpy and less smooth.
Calculating the surface can therefore be influenced by the defined resolution. Some resolution might even be too high as that some mobile phones will be able to handle with their low resources. Therefore, we need to test the influence of the resolution on the calculation time of the surface. As mentioned in Section 2.1, we implemented two different algorithms for the calculation of the surface: Octree and Marching cube algorithm.

## 7.1 Octree

The resolution of the octree is defined in the maximum depth till where the algorithm calculates the tree (see Section 2.1). The deeper the maximum depth is, the higher the resolution of water droplets.
In Chapter 4, we already saw that the octree is too slow for the moving water droplet on mobile phones. Therefore, the testing of the octree resolution will be conducted with one static water droplet. The static water droplet is placed at the beginning of the test environment. Again, a stopwatch is used to measure the time needed to calculate the surface of the water droplet. The difference here is that the surface is only calculated once at the beginning. Therefore, it is not necessary to calculate the average of all surface calculation during the simulation.
Furthermore, we used four mobile phones for the testing of the resolution instead of five as mentioned in the experiment set-up in Chapter 3: Samsung Galaxy S2, HTC Desire, HTC Desire HD and Samsung Galaxy S.
Other processes running on the mobile phones in the background could influence the needed calculation time which would result in different times for every run. Therefore, we decided to run the simulation ten times on every mobile phone. The maximum depths that were used in the experiments are two and three. We also

tried with a maximum depth of four, but the calculations were for most of the devices computational expensive and let them even crash.

### 7.1.1 Results

Running with a maximum depth of three makes the water droplet smoother than when it is run with a maximum depth of two. But a greater maximum depth also increases the calculation time as you can see in Figure 7.1. On most of the mobile phones is the average time for the calculation of the surface with a maximum depth of three even higher than 100 milliseconds and slows the performance down at the beginning.

## 7.2 Marching cube algorithm

The resolution of the marching cube algorithm is not defined with a maximum depth as the octree is, but with the size of the cubes in the grid (see Section 2.1). Smaller cubes in the grid allow the surface of the water droplets to have higher resolution.
In this experiment, the cube size of the lowest resolution is 0.2. Any lower resolution would result in having a not sphere like form anymore. Higher resolution than with a cube size of 0.075 leads to resource problems on mobile phones. Therefore, the cube size used in this experiment range from 0.075 to 0.2.
In Chapter 4, the calculation time with the marching cube algorithm was fast enough to test it with the moving water droplet. Also, what we want to measure is the same as in chapter 4, that is the time needed to calculate the surface, the set-up in this experiment will be the same as the set-up used there. This would be a stopwatch that measures the time needed for the calculation of the surface, testing it on five different mobile phones and using the same route.
Furthermore, the experiment is repeated ten times on every device and for every cube size.

### 7.2.1 Results

In Figure 7.2, you can see the average of the ten measured calculation times including the standard deviation. As seen there, the calculation time will lessen the higher the cube size (lower the resolution) gets. This shows clearly that the resolution has a great impact on the calculation time of the shape.
The side effect of the lower resolution is that the surface of the water droplet gets bumpier. With too low of a resolution, the water droplet even starts to have a

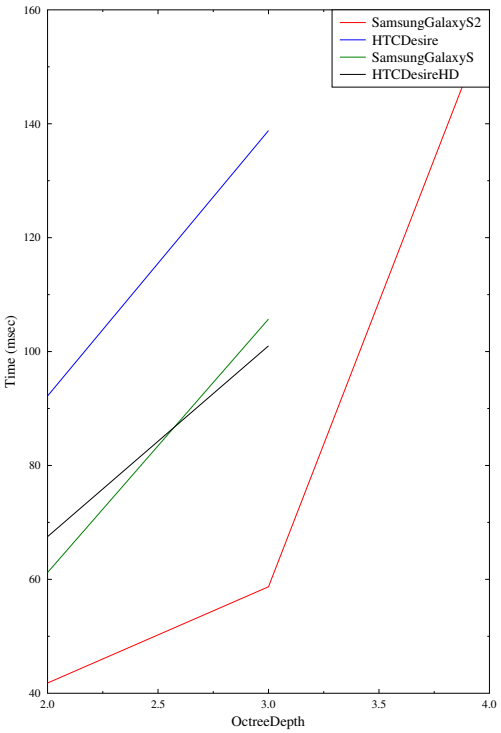



Figure 7.1: This plot shows the calculation time of the octree for different depths.

cube like form instead of a sphere like form. A cube size of 0.2 starts already with making the water droplet look like a cube.

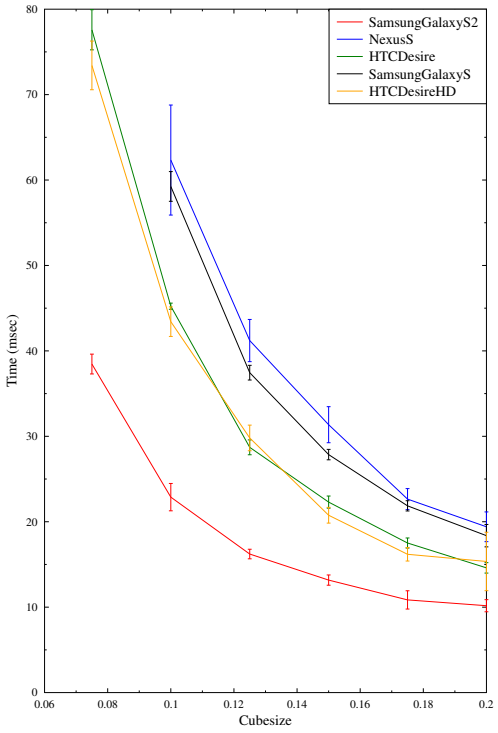



Figure 7.2: This plot shows the measured calculation time for every cube size and the standard deviation.All the times are reported in milliseconds.

# 8 Performance

To see how good an implementation actually is, one has to take a look at the performance of it and see if the performance is good. A good performance means that the movements in the simulation are fluid and not jerky. It is important to have a good performance especially in games. In games having jerky movement makes the player feels as if the object in the game is not completely under his/her control and therefore breaking the game flow.
Because mobile phones have low resources the performance depends on the calculation times. In the previous experiments, the needed calculation time was already measured, but the experiments only looked at part of the simulation and not at the whole simulation fo water droplets.
During the experiment in Section 7.2, we noticed that the water droplet moved smoothly when the calculation times was lower than or equal to 18 milliseconds. As you can see in Figure 8.1, most of the mobile phones have a calculation time below 18 msec when low resolution is used. The only exceptions are the Nexus S and the Samsung Galaxy S. Even with the lowest of the tested resolution, the calculation time of the marching cube algorithm is above 18 milliseconds on those two mobile phones. The controlling of the moving water droplet with those two phones did also not feel as smooth and good as with the other mobile phones during the resolution experiment because the movement was more jerky.
In the experiment in Section 7.2, the following technique was used to calculate the movement of the water droplet. As was seen in Chapter 5, the following technique needs more time to calculate the movement than the interpolation technique. Using the following technique instead of the interpolation technique should therefore lower the performance of the simulation.
Running the simulation again on the mobile phones with low resolution, but now with the interpolation technique, did indeed increase the feeling of the performance as seen in Figure 8.2. Even running on the Nexus S and the Samsung Galaxy S did not give any jerky feeling and the simulation ran smoothly
So, the performance is affected by the resolution and the choice of the movement. Lower resolution raises the performance. The performance is then again raised by

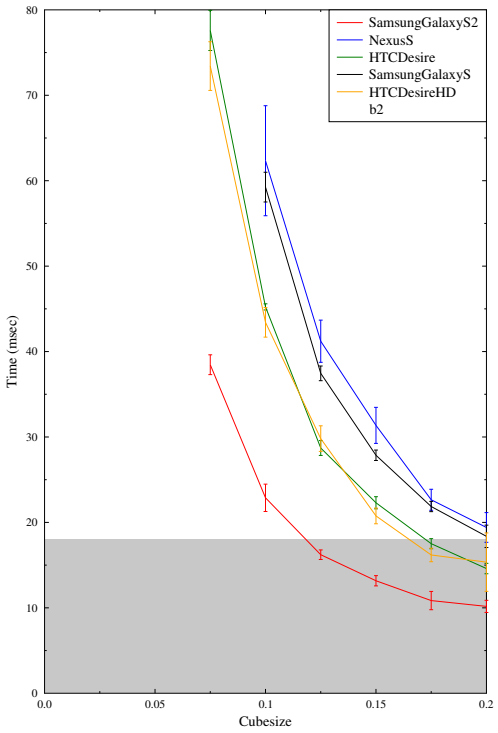



Figure 8.1: This plot shows the measured calculation time for every cube size and the standard deviation while using the following technique. The grey area marks where the performance starts to feel like a good performance. A good performance feeling is reached in the grey area.

using the interpolation technique instead of the following technique.
This shows that every calculation reduces the performance of the simulation.
The static water droplets also need to be calculated, but only at the beginning. That is why the performance dropped at the beginning and rises later, as was seen

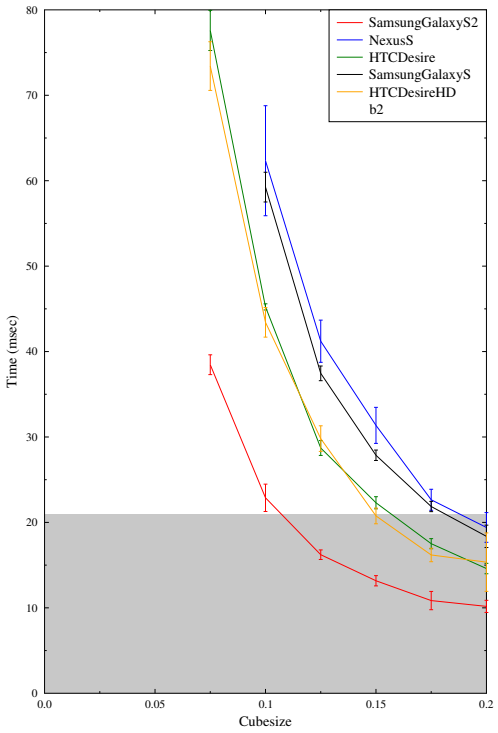



Figure 8.2: This plot shows the measured calculation time for every cube size and the standard deviation while using the interpolation technique. The grey area marks where the performance starts to feel like a good performance. A good performance feeling is reached in the grey area.

in Chapter 6.

# 9 Discussion

Previous researches simulate water droplets on high computational resource devices and not on low resources devices such as mobile phones. The previous experiments (Chapter 4 - 8) showed that the simulation of water droplets introduced in this thesis is able to run on different mobile phones.
Still, how it runs on the mobile phones depends on the mobile phone. All the experiments showed that the choice of the mobile phone has great impact on the performance. For example, the simulation runs faster and better on the Samsung Galaxy S 2 than on the Nexus S.
The low resources of the mobile phones made it necessary that more than one surface calculation method has been implemented. As seen in Chapter 4, the marching cube algorithm needs too much memory for the calculation of many static water droplets. The octree on the other hand needs less memory, but is slower. Therefore, both algorithms are needed in the simulation. Using the marching cube algorithm for the water droplet that moves is chosen because it is fast. We use the octree algorithm for the static water droplets because it uses less memory.
The behaviors of the two implementations for calculating movement are also different. The interpolation movement is faster, but visually less realistic than the following movement.
This shows that when using one of these implementations, there is a tradeoff between performance and visual graphics. Also, the choice of the resolution is another tradeoff between performance and visualization of the water droplet as seen in Chapter 7. A higher resolution makes the water droplet smoother and visually better, but also decreases the performance of the program. A good performance on different mobile phones is reached with a low resolution and interpolation movement whereas high resolution and using the following movement is visually better but has a bad performance (see Chapter 8).
In a game, the performance is important especially for the game flow. Visual quality is also important, but less than the performance. Therefore, it is advisable to reduce the visual quality for the performance in game wise.
Static water droplets also decreases the performance in the beginning of the simula-

tion for some time as seen in Chapter 6. The time before the performance increases again, depends on the number of static water droplets. Nevertheless, merging is one of the key abilities of water droplets and in the simulation of water droplet introduced in this thesis only possible with static water droplets. This ability can also be used for game elements in games such as Hydrotilt. The amount of static water droplets in such a game would therefore only be limited by the time the game is allowed to wait at the beginning, camouflaging this time with a loading screen.
The decrease in performance with the static water droplet is because of the calculation time needed for the sink in. The sink in allows a more realistic placing of the water droplet on a surface, but decreases the performance. Therefore, again a choice can here be made between visual correctness of the water droplet behavior (sink in) or performance. Not using the sink in would increase the performance, but it would then again be visually incorrect to let the static water droplet not sink into the ground.

# 10 Conclusion

In the scope of this thesis, we implemented in an experiment set-up moving and static water droplets on mobile phones using the metaballs model, Unity and C#. The physical behavior of water droplets (such as sinking in according to a contact angle, merging and changing shape from sphere to droplet form when moving) were considered and also implemented.
We tested the implemented set-up on five different mobile phones. On all these five mobile phones, the simulation was able to start up and we were able to let the moving water droplet move. This shows that the simulation of water droplets with metaballs is able to run on low resources devices such as mobile phones.
According to the physics of a water droplet, the water droplet will sink into a solid surface and form a contact angle with this surface. Another physical behavior is the merging with other water droplets. A water droplet merges as soon as it is close enough with another water droplet resulting in one bigger droplet. These two physical behaviors are implemented in the experiment set-up. The only limitation is that the moving water droplet does not sink in according to the contact angle because the sinking in process is to slow and that would result in performance issue on the low resources devices. Therefore, the moving water droplet is always sunk in to half of the sphere form.
With a very good performance, real time behavior of the implementation is possible. Nevertheless, the experiments conducted show that the graphics of the water droplets (visually) and the performance on the mobile phones conflict with each other. This is why, using a movement that looks more realistic and high resolution of the water droplet decreases the performance. On the other hand, using low resolution and a movement that does look less realistic improve the performance and even result in real time performance on the mobile phones. This shows that real time performance is possible with our implementation with less realistic look and behavior.
The main question of this thesis was:

> How to implement a water droplet based on metaballs with the right behavior on a device with low resources?

Our implementation makes this possible and answers therefore this.

## Future work

A game is implemented in such a way that it runs in real time. As was shown, using low resolution and a less realistic movement, the water droplet in our implementation is able to move in real time, without hesitations on low resources devices. This indicates that this simulation of water droplets is usable in a game for low resources devices such as mobile phones. It is possible that game elements would slow the simulation down and decrease the performance resulting in running not in real time. Therefore, testing the simulation of water droplets introduced in this thesis with game elements is one possible future work.
Furthermore, the sink in process implemented was to slow to be usable for the moving water droplet. Nevertheless the sink in is according to a contact angle with the solid surface a natural and more realistic behavior than always sunk in to the half of the whole droplet. Therefore, we want to look if there are other possibilities and implementations for the sink in process in future researches.

# Bibliography


[1] J. Blinn. A gerneralization of algebraic surface drawing. *ACM Transaction on Graphics*, 2(3):235 – 242, 1980.

[2] D. Chen and J. Zhang. Merging of water droplets base-on metaball. *International Conference on Digital Manufacturing and Automation*, pages 716 – 719, 2010.

[3] Codeglue. Description of hydrotilt from Codeglue on their homepage. `http://codeglue.com/game.php?id=11\&game=HydroTilt`, Mar. 2009.

[4] P. de Gennes. Wetting: Statics and dynamics. *Rev. Mod. Phys.*, 57:827 – 863, 1985.

[5] P. Eisert. Chapter 8. reconstruction of volumetric 3d models. In O. Schreer, P. Kauff, and T. Sikora, editors, *3D Videocommunication: Algorithms, Concepts and Real-Time Systems in Human Centred Communication*, pages 133 – 150. John Wileys & Sons, Ltd, 2006.

[6] P. Fournier, A. Habibi, and P. Poulin. Simulating the flow of liquid droplets. *Graphics Interface*, pages 133 – 142, 1998.

[7] A. Iglesias. Computer graphics for water modeling and rendering: a survey. *Future Generation Computer Systems*, 20:1355 – 1374, 2004.

[8] K. Kaneda, S. Ikeda, and H. Yamashita. Animation of water droplets moving down a surface. *Journal of Visualization and Computer Animation*, pages 15 – 26, 1999.

[9] K. Kaneda, T. Kagawa, and H. Yamashita. Animation of water droplets on a glass plate. *Proceedings of Computer Animation ́93*, pages 177 – 189, 1993.

[10] T. Lewiner, H. Lopes, A. W. Vieira, and G. Tavares. Efficient implementation of Marching Cubes cases with topological guarantees. *Journal of Graphics Tools*, 8(2):1 – 16, 2003.

[11] S. Murakami and H. Ichihara. On a 3d display method by metaball technique. *Journal of papers at the Electronics Communication*, J70-D(8):1607 – 1615, 1987.

[12] T. S. Newman and H. Yi. A survey of the marching cubes algorithm. *Computer & Graphics*, 30:854 – 879, 2006.

[13] H. Nishimura, M. Hirai, T. Kawai, T. Kawata, I. Shirakawa, and K. Omura. Object modeling by distribution function and a method of image generation. *Transactions of the Institute of Electronics and Communication Engineers of Japan*, J68-D(4):718 – 725, 1985.

[14] R. Tong, K. Kaneda, and H. Yamashita. A volume-preserving approach for modeling and animating water flows generated by metaballs. *The Visual Computer*, 18(8):469 – 480, 2002.

[15] F. Triquet, P. Meseure, and C. Chaillou. Fast polygonization of implicit surfaces. *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG'01)*, 2:283 – 290, Feb. 2001.

[16] Unity Technologies. What is Unity and what can I do with it? `http://unity3d.com/create-games/`, June 2012.

[17] H. Wang, P. J. Mucha, and G. Turk. Water drops on surfaces. *ACM Transactions on Graphics (TOG)*, 24:921 – 929, 2005.

[18] WolframMathWorld. the web's most extensive mathematics resource, Spherical Cap. `http://mathworld.wolfram.com/SphericalCap.html`, July 2012.

[19] G. Wyvill, C. McPheeters, and B. Wyvill. Data structure for soft objects. *The Visual Computer*, 2:227 – 234, 1986.

[20] Y.-J. Yu, H.-Y. Jung, and H.-G. Cho. A new rendering technique for water droplet using metaball in the gravitation force. *Proceedings of the 6th International Conference in Central Europe on Computer Graphics and Visualization, WSCG '98*, pages 432 – 439, 1998.

[21] Y.-J. Yu, H.-Y. Jung, and H.-G. Cho. A new water droplet model using meatball in the gravitational field. *Computers and Graphics*, 23:213 – 222, 1999.

[22] Y. Zhang, H. Wang, Y. Tong, S. Wang, and K. Zhou. A deformable surface model for real-time water drop animation. *IEEE Transactions on Visualization and Computer graphics*, 2011.

# A

# Class Diagram and implementation

Figure A.1 shows the diagram of the whole implementation. Because it was made smaller to fit on a page, the names of the classes, functions and variables are not readable. Therefore, We split the class diagram into two half that are shown in Figure A.2 and Figure A.3.

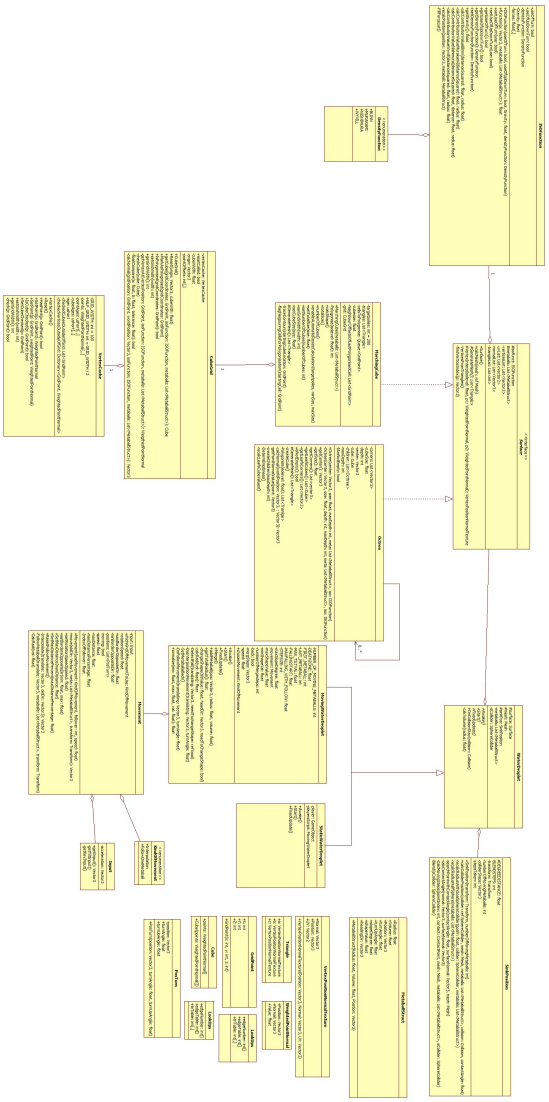


Figure A.1: The class diagram of the whole implementation.

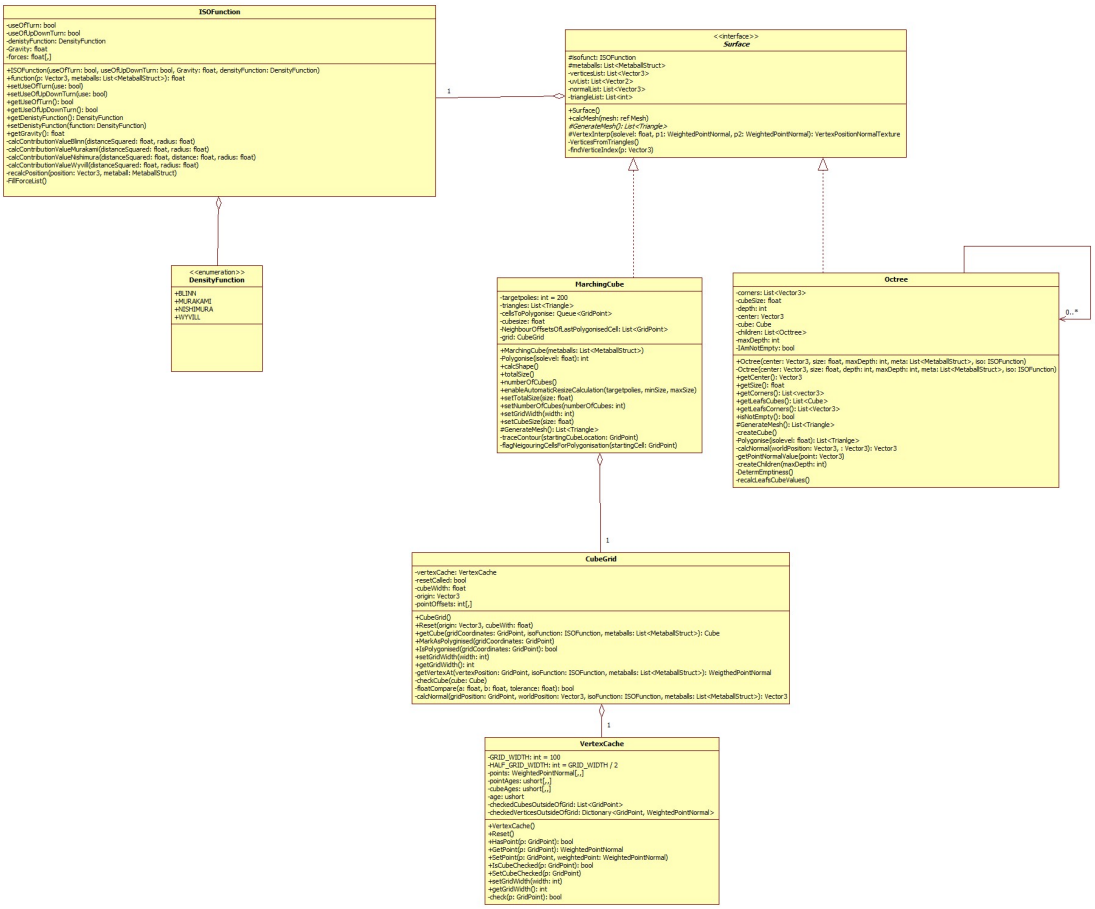



Figure A.2: The first half of the class diagram of Figure A.1.

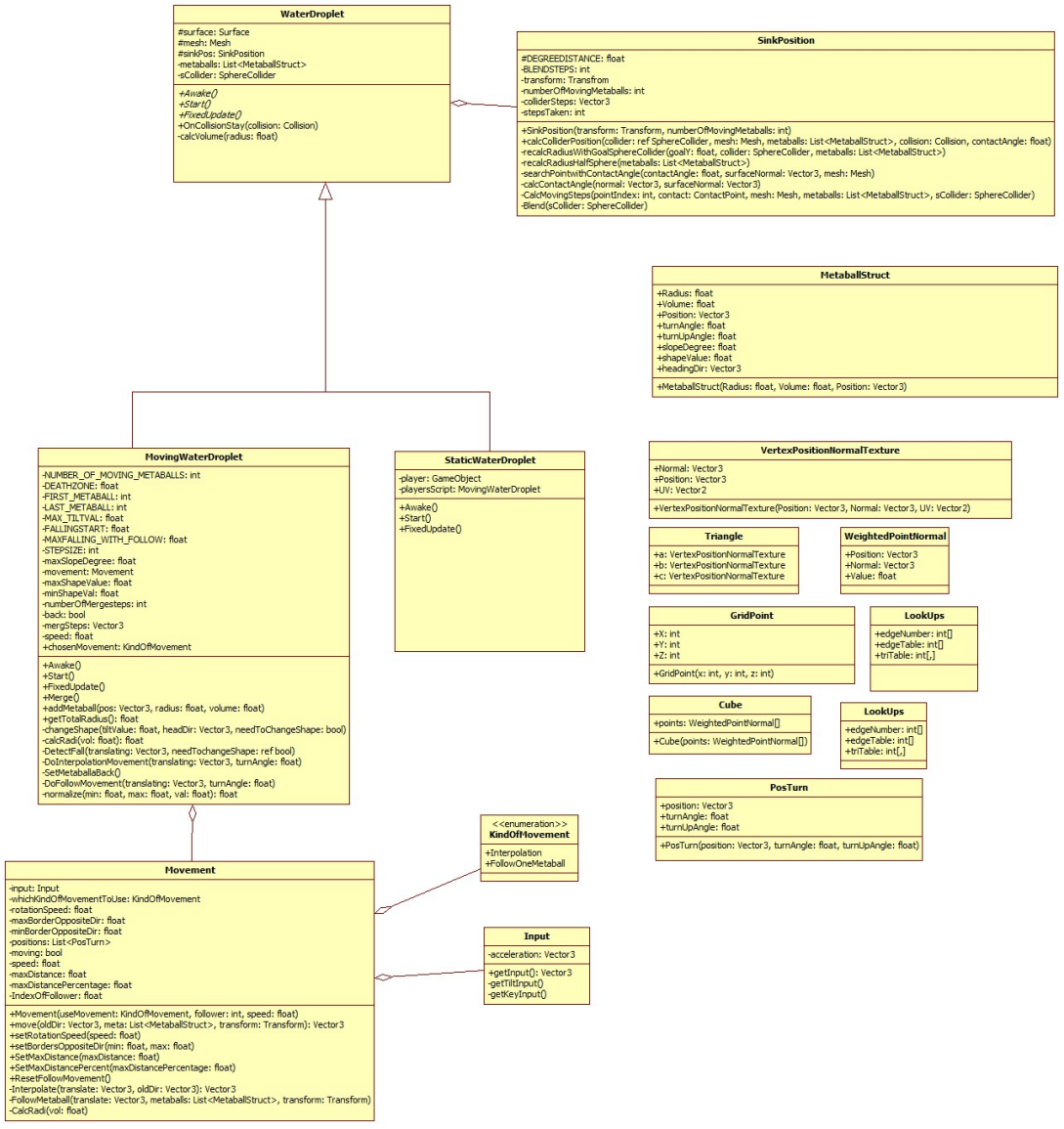



Figure A.3: The second half of the class diagram of Figure A.1.

As the class diagram shows in Figure A.1, the whole implementation of the water droplet is split into four classes/parts: Water Droplet, Surface, Movement and Sink in.
In Appendix A, you can find a class diagram of the complete implementation.

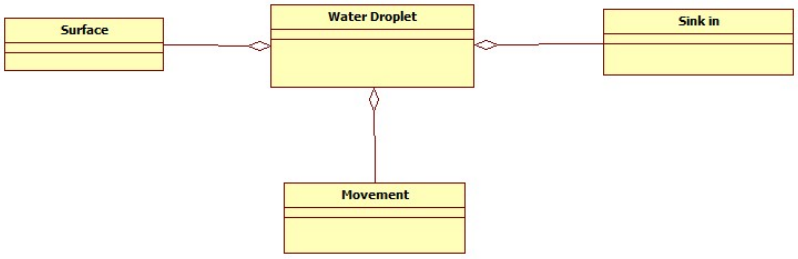



Figure A.4: Main structure

## A.1 Surface

As mentioned in Chapter 2.1, we want to implement two algorithms that are able to calculate the surface: the octree and the marching cube algorithm. The water droplet can have either one of those two algorithms to calculate the surface. Therefore, those two are subclasses of the class Surface as you can see in Figure A.5. To make the marching cube algorithm faster, as mentioned in Chapter 2.1.1,

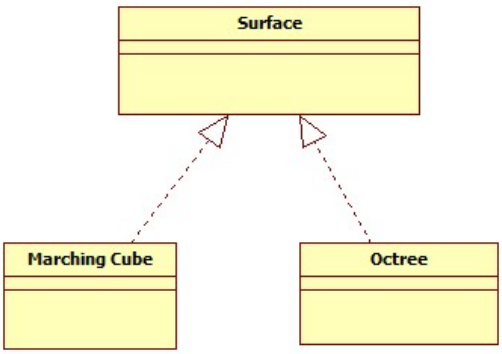



Figure A.5: Implementation structure of the Surface

we implemented the timestamps. The class vertexCache is where we implemented the timestamp. It keeps track of the current timestamp and the timestamp for

every cube in the grid of the marching cube algorithm.
The surface of the metaballs is calculated mathematically as mentioned in Chapter 1.1.2. Both algorithms need those mathematical functions to calculate the surface. Therefore, the class Surface should have those function implemented so that both algorithm can access them. But to maintain an object oriented implementation, we decided to place those function into an extra class and let the surface have an instance of this class.
In Figure A.6, you can see the complete implementation of the Surface.

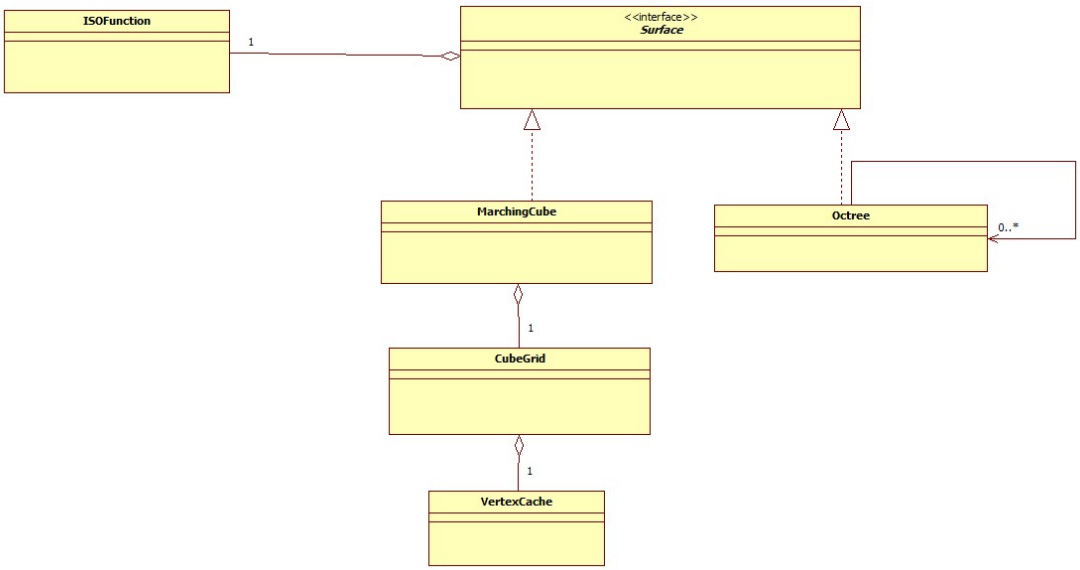



Figure A.6: Classes structure of the complete implemented Surface

## A.2 Movement

The movement of the water droplet can either be an interpolation movement or a follow movement. In the water droplet class, the kind of movement to be used has to be defined at the beginning. For the interpolation is the minimum value and the maximum value of the range defined by variables. Furthermore, so that the user can move the water droplet as he/she wants it to, the movement has an input. In the game Hydrotilt, the movement of the droplet depended on the tilt of the mobile device. We want to imitate this behavior. Therefore, we implemented the tilt and how much the mobile device is tilted as input. The speed of the movement depends then on how much the mobile device is tilted.
As mentioned in Chapter 2.2, the shape of the water droplet has to change if it

moves. This change is done with the help of a slope degree. When the water droplet does not move, the slope degree should be 0. Tilting the mobile phone changes that. To give the user a natural feeling of the change of the shape, we decided to use the angle of the tilt as the slope degree and change the shape according to that. This ensures also that the water droplet will change its form back before moving into the opposite direction as you can see in Figure A.8.
In Figure A.7, you can see the implementation structure of the movement.

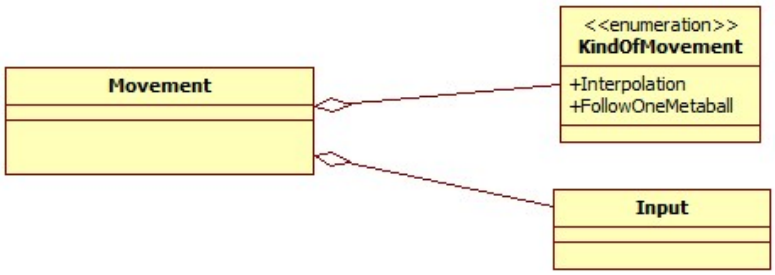



Figure A.7: Class structure of the movemenet implementation

## A.3 Sink in

Unity already provides some physics and physics mechanisms such as colliders. The colliders allow the detection of collisions between two objects. Because the shape of a droplet is sphere-like when it is not moving, we will use the spherecollider for the water droplets. That is a collider in the shape of a sphere.
With the spherecollider, unity detects every collision it will have with other rigid bodies such as for example the ground the water droplet will lay on. In our implementation, if the water droplet collides with another object, it should sink into that one according to the contact angle. Therefore, we first provide all the other objects, which the water droplet should sink into, in the environment of the water droplet with a script that contains only the contact angle belonging to that object.
Every time a collision is detected, our implementation will first check for that script. If in the collision such a script is detected, it will then go through all the points of the water droplet surface. Under all these points, the implementation searches for a point that has the same or almost the same contact angle as is stated in the found script.
To let the water droplet sink in, our implementation will move the spherecollider

(a) Basic holding position of mobile where tilt is (almost) 0 and therefore the slope degree is also 0.

(b) Changing the moving direction into the opposite movement by changing the tilt of the mobile.

Figure A.8: To change the moving direction into the opposite direction, the tilt of the mobile has to change. In this movement, the mobile device is held shortly in a basic. This causes the water droplet to change its form back before changing movement into the opposite direction.

step by step until the collision point between the water droplet and the surface is at the same height as the determined point. At every step, the radius of the water droplet is also recalculated depending on the volume of the water droplet (see Chapter 2.3) and adjusted so that it seems like the water droplet is spreading instead of sinking in.
These calculations make a moving water droplet too slow on mobile devices to move around. Therefore, we decided to only calculate the sink in for water droplet that will not move around. To prevent the moving water droplet from being placed on the surface like a marble, we will let it sink in till only half of it is seen, just like Yu et al. did in their research [21].

## A.4 Water Droplet

Water droplet is the main class where all the behavior of the water droplet is combined. Because we do not want only moving water droplets, but also droplets that are static, the water droplet can either be a Moving water droplet that uses movements or a static Water droplet as seen in Figure A.9.

The static water droplets will sink in the ground according to the contact angle

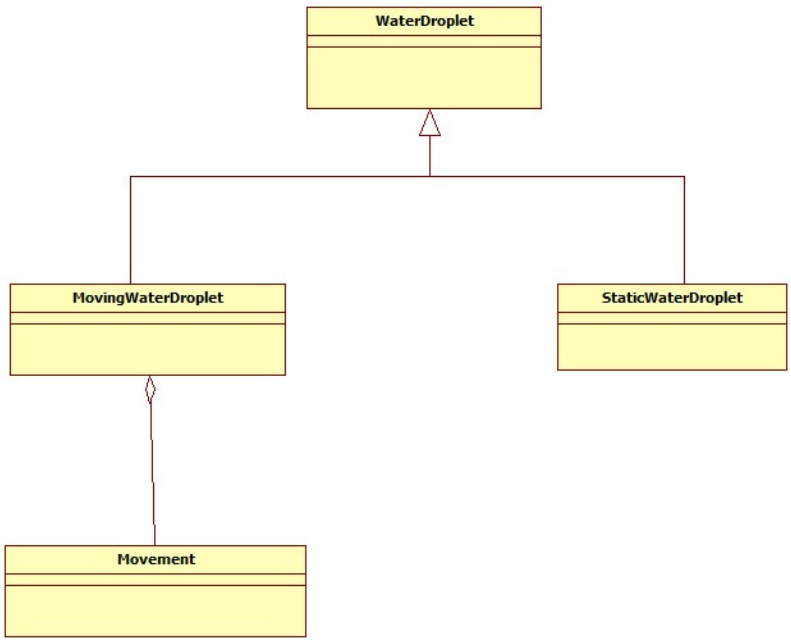



Figure A.9: Class structure for the class water droplet

and otherwise only wait until it is close enough to the moving one for merging. Because, in Unity, every water droplet will be created as a game object on its own, the static water droplet needs to add itself into the game object of the moving water droplet as soon as that one is close enough to be able to merge.
The moving water droplet will move according to the tilt of the mobile device. As soon as a metaball is added to the moving water droplet, it will stop moving to merge with the static one.
The merging process works as follows: the metaballs of the moving water droplet will move step by step towards the position of the static one. Because of the mathematical description of the metaballs, it will visually seem like two water droplets that are merging. As soon the positions of the metaballs are equal, the radius and volume of the moving water droplet are adjusted and the static water droplet deleted. The merging process is then complete and the water droplet will then again move according to the tilting of the mobile and the chosen movement.

# B
# Test Environment

To test the behaviour of the water droplet, a test environment was set-up in Unity. Figure B.1 shows how the test environment looks like. The moving watewr droplet starts always at the most upper floor.

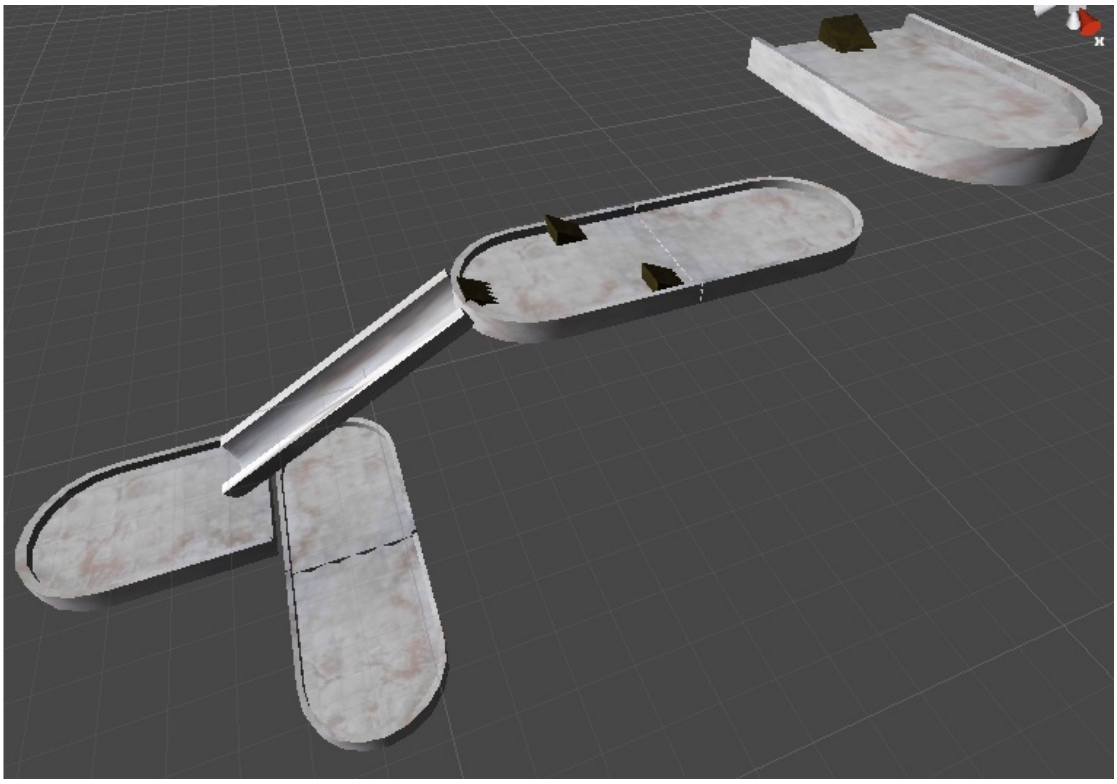


Figure B.1: The test environment that was used for the implementation and the experiments.