FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Sketch-Based Steering in Visdom

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Computergraphik & Digitale Bildverarbeitung

eingereicht von

Roman Gurbat
Matrikelnummer 0525731

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Mitwirkung: Dipl.-Ing. Dr.techn. Jürgen Waser

Wien, 25.11.2011

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Sketch-Based Steering in Visdom

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Visual Computing

by

Roman Gurbat
Registration Number 0525731

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Assistance: Dipl.-Ing. Dr.techn. Jürgen Waser

Vienna, 25.11.2011

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Roman Gurbat
Ölweingasse 3-5/19, 1150 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Acknowledgements

I would like to express my gratitude to all those who gave me the possibility to complete this thesis. First of all, I would like to thank Eduard Gröller for his supervision and guidance. Also I would like to thank the VRVis for allowing me to do the practical part of my thesis there. I would like to thank Jürgen Waser, the founder of Visdom, and my supervisor at the VRVis. He helped out with implementation details and the development of this work. A special thank goes to Hrvoje Ribičić, for his valuable inputs and strong support during the implementation and the writing of this thesis.

My deepest gratitude goes to my parents for giving me as much support as they where able to ante up. Finally, I would like to express my sincere thanks to Nicole, for her patient and unconditional support.

This thesis was supported in part by a grant from the Austrian Science Fund (FWF):P 22542-N23 (Semantic Steering).

Abstract

Natural phenomena like earthquakes or floods cannot be avoided or controlled by humans but ignoring them can be fatal. In order to understand their characteristics and behavior computer simulations can be used. The gained knowledge is used to find strategies and countermeasures to minimize the damage caused by such disasters.

Visdom is a modular integrated visualization system which is developed to support the analysis of flood simulations. It provides the ability to test different decisions made by the user for example the establishment of barrier arrangements. These alternative scenarios can be visualized and analyzed in order to find the solution which minimizes damage in the vicinity. For modifying the flood, the simulation parameters have to be varied. In Visdom, this has to be done by inserting numeric values into textfields or using sliders. In most cases, the system is intended to be used in time-critical situations where fast interaction and testing is required. Therefore a sketch-based interface is introduced into Visdom which should simplify and accelerate the modification of simulation parameters. This kind of interaction increases the productivity and efficiency of the whole work-flow due to its intuitive and straightforward usage.

Since Visdom is based on a modular data-flow system, the new functionality is implemented within a new node. The so-called Spline Node provides methods to create and manipulate a spline. It requires the drawn user input points which are captured by a view node. Due to the fact that Visdom is based on the data-flow concept, a new mechanism called Modular Interactors has to be introduced to enable interaction communication between nodes especially upstream. In order to support the user during the sketching process the system provides visual feedback. The sketched stroke is represented using a tubular geometry which is directly visualized in the 3D view.

After the spline is constructed it has to be interpreted to trigger commands which influence the simulation. For this issue two additional nodes are introduced. The Spline To Barrier Node is responsible for translating the sketch to a sandbag barrier or a mobile protection wall. Manipulating the spline allows for changing the properties of the established barriers. The second interpretation possibility is implemented by the Spline To Force Node. It creates a force field according to the appearance of the spline which manipulates the fluid simulation directly. Also for this task visual feedback is essential which should provide information about the acting forces. Therefore a new render component is implemented which takes a vector field as input and generates an arrow plot. This component is also used to visualize the internal flow of the fluid.

Kurzfassung

Naturereignisse, wie Erdbeben oder Überflutungen, können von der Menschheit nicht verhindert werden. Sie zu ignorieren wäre jedoch ein fataler Fehler, der zu katastrophalen Folgen führen kann. Computersimulationen werden verwendet, um die Eigenschaften und das Verhalten der Naturgewalten zu analysieren. Das gewonnene Wissen wird verwendet, um Strategien und Gegenmaßnahmen zu entwickeln, mit deren Hilfe die Schäden vermindert werden sollen.

Visdom ist ein modulares integriertes Visualisierungssystem, das entwickelt wurde, um Überflutungssimulationen zu analysieren. Benutzer können Barrieren in verschiedenen Konstellationen errichten und deren Effizienz testen. Die dadurch entstehenden Alternativszenarien können visualisiert und analysiert werden, um jene Lösung zu finden, die den Schaden, der in der Umgebung angerichtet wird, minimiert.

Um die virtuelle Überflutung zu manipulieren, müssen die Simulationsparameter verändert werden. In Visdom werden diese über numerische Werte oder Schieberegler gesteuert. Die meisten Anwendungsfälle sind zeitkritisch und benötigen eine schnelle Interaktion zwischen dem Benutzer und dem System. Das Erweitern von Visdom durch ein sketch-basiertes Interface soll die Steuerung der Simulation vereinfachen und beschleunigen. Die Produktivität und Effizienz des gesamten Workflows soll durch die intuitive Handhabung gesteigert werden.

Visdom ist durch das auf Nodes basierte Design hoch modular. Die neue Funktionalität wird durch die Implementierung eines neuen Nodes, genannt Spline Node, ermöglicht. Dieser bietet die Möglichkeit, eine Spline zu erzeugen und zu manipulieren. Als Input werden jene Punkte verwendet, die der User durch das Zeichnen in dem 3D Fenster

definiert. Visdom basiert auf dem Data-flow Konzept, welches einen strikten Ablauf der Daten vorsieht. Um die Kommunikation zwischen den Nodes, vor allem entgegen der üblichen Flussrichtung, zu ermöglichen, wird das neue Konzept der Modular Interactors eingeführt. Visuelles Feedback wird verwendet, um den Benutzer während des gesamten Prozesses zu unterstützen. Die gezeichnete Spline wird durch eine schlauchförmige Geometrie repräsentiert, die direkt in dem 3D Fenster angezeigt wird.

Nach der Konstruktion muss die Zeichnung des Benutzers in einen Befehl umgewandelt werden, der die Simulation beeinflusst. Dies wird durch zwei weitere Nodes realisiert. Der Spline To Barrier Node wandelt die Spline in Barrieren aus Sandsäcken oder mobilen Schutzwänden um. Die Eigenschaften dieser Objekte können mit Hilfe der Spline und deren Manipulationsmöglichkeiten bestimmt werden. Als visuelles Feedback werden die Barrieren in einem Preview-Modus angezeigt, was Benutzern das frühzeitige Auffinden von potenziellen Fehlern ermöglicht.

Der Spline To Force Node ist verantwortlich für die Interpretation der Spline als Kraftfeld, welches die Partikel der Flüssigkeitssimulation direkt beeinflusst. Einflussbereich und Richtung des Feldes können wieder durch die Manipulation der Spline bestimmt werden. Um einen effizienten Ablauf zu ermöglichen, werden dem Benutzer Informationen über die wirkenden Kräfte durch ein Feld aus Richtungpfeilen bereitgestellt. Um dieses Feld aus einem Vektorfeld zu generieren, wurde eine neue Komponente entwickelt. Die Visualisierung durch Richtungspfeile wird auch verwendet, um Informationen über die interne Struktur der Flüssigkeit anzubieten.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Mankind always had to cope with the unpredictability of the forces of nature. Every year natural disasters like droughts, earthquakes, storms or floods affect thousands of people and wreak havoc all over the world. According to the Centre for Research on the Epidemiology of Disasters (CRED), floods were the most common natural disasters in 2010 [27].

Because of world-wide climatic change, the probability of climate-related hazards will even increase. For example, due to global warming the additional amount of water which flows into the rivers after the ice melts every year could lead to flash floods throughout Europe and the rest of the World. Also, heavy rainfalls can cause flooding, as happened in August 2010 in the border area between the Czech Republic, Poland and Germany. Many rivers and creeks burst their banks and overwhelmed near villages. Due to the flood, many households where cut from the power supply [30]. If a critical building, like a hospital or another important institution, would lose the connection to the electric supply network, the number of victims could increase dramatically. As a protective measure the European Union published a directive on the assessment and management of flood risks [24] which aims to reduce the damage caused by floods. Member states have to create flood hazard maps and flood risk maps which define po-

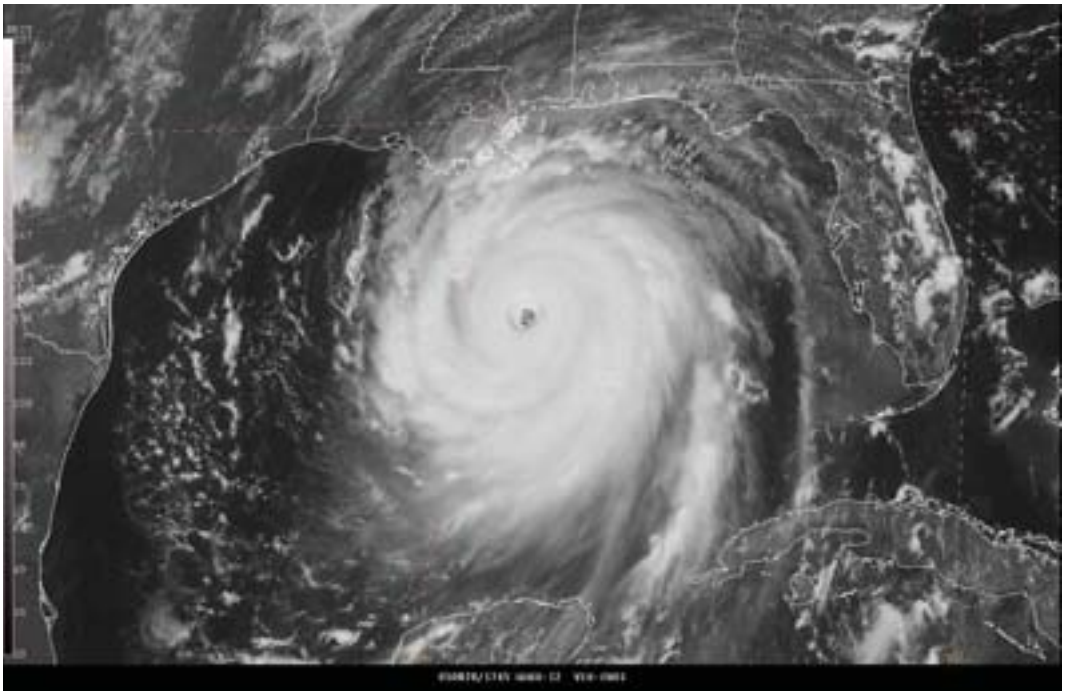

Figure 1.1: Satellite image of hurricane Katrina [48]

tential consequences of floods. Based on these maps, flood risk management plans have to be established in order to lay out a strategy of how to protect the vicinity and prevent the escalation of flood events. Every six years these plans have to be reviewed and updated if necessary.

Countries which are situated at the seaside could be even more affected by the perils induced by climate change. Coastal regions, especially those located in low-lying areas, have to face a higher risk of flooding caused by the sea-level rise and a growth of storminess [44]. One of the most disastrous storms in the last years was Hurricane Katrina (see Figure 1.1). According to a report of the National Hurricane Center [48], Katrina was the costliest and one of the five deadliest hurricanes ever to strike the United States. The storm first hit Florida as a Category-1 hurricane on the Saffir-Simpson Hurricane Scale before reaching the Gulf of Mexico, where it increased intensity and became a Category-5 hurricane. On 29 August 2005 Katrina struck the coastline along the northern Gulf of Mexico, the coast of Louisiana, Mississippi and Alabama. 1833 people died in this devastating natural disaster. Many of them perished directly in the actual hurri-

Figure 1.2: Breach at the Industrial Canal (top left and top right), the London Avenue Canal (bottom left) and the 17th Street Canal (bottom right) [3]

cane, but most of the fatalities were caused by the subsequent floods. With a property damage of over 81 billion dollars due to destroyed homes and businesses throughout entire neighborhoods in the New Orleans metropolitan area, Katrina caused twice as much damage as the previously most expensive storm Andrew in 1992.

In 2007, two years after the catastrophe, the American Society of Civil Engineers published a report where they analyzed the happening and tried to understand why things had gone wrong [3]. Like in many other cities, large areas of New Orleans are situated below sea level. To protect the inhabitants from the water of the Gulf of Mexico, Lake Borgne and Lake Pontchartrain, the United States Army Corps of Engineers (USACE) designed and built a flood protection system. The system consists of levees, floodwalls and one of the largest pumping systems in the world. But as Katrina hit New Orleans the system failed catastrophically. During and after the hurricane many of those protection structures could not stand the pressure. Multiple breaches occurred in more than 50 locations, allowing the water of the gulf and the lakes to rush into the city. Over 80

Figure 1.3: Helicopters dropping sandbags to close levee-breaches [65] [68] [76]

percent of the city was flooded and some neighborhoods were submerged more than 10 feet deep in water. The major breaches were at the Industrial Canal, the London Avenue Canal and the 17th Street Canal (see Figure 1.2). The levees and floodwalls breached because of a combination of unfortunate choices and decisions made at almost all levels of responsibility. The report by the American Society of Civil Engineers also states that if the protection system had not failed, two-third of the deaths would not have occurred. During every hurricane with the strength and intensity of Kathrina, some damage and major flooding are expected as well as some levee overtopping. But in the case of Katrina, the protection structures breached way before the maximum exposure was reached.

The breach of the levee situated at the 17th Street Canal was the most catastrophic and responsible for most of the flooding in the city. The USACE tried to close it for the restoration efforts to continue [15]. First they tried to close the breach by dropping 3000 lb sandbags deployed by U.S. Army Chinook and Black Hawk helicopters (see

Figure 1.3). This initial attempt failed because the sandbags were dropped too close to the breach and were too light, so they were washed away by the flood. The engineers noticed that the task of closing the breach was not straightforward and thus, many attempts were needed to finally make a stable barrier and seal the breach. Later they increased the size of the sandbags and used some with a weight of 6000 and 7000 lbs. The plans in the field for closing the breach were changed many times. Such trial-and-error procedures are common in case of emergencies, because there exists no standard procedure or systematic research in the field of routines during a flood event.

To gain more insight and to learn from the natural disaster in New Orleans Sattar et al. [79] carried out a case study of breach closure procedures. They constructed a 1:50-scale model based on the situation in New Orleans and simulated the 17th Street Canal levee breach and the subsequent flood in a hydraulic laboratory. Such a real-world simulation is very accurate and reliable, but also very expensive and time-consuming. Moreover there is no possibility to compare alternative scenarios in an efficient way. Therefore computer applications were developed to perform such case studies in a virtual environment. These so-called steering applications provide the ability to execute and control the simulation. The scenario can be set up and adjusted easily which allows the user to explore multiple alternatives with less effort.

1.2 Problem Statement

Within a steering application, the user is able to specify the terrain geometry and start a fluid simulation. As response a visualization of the impact of the flood is received. By changing the input parameters of the simulation, the influence of decisions made can be investigated. An example of an input parameter is the positioning of sandbag barriers. In emergency situations caused by natural catastrophies like floods, it is important that the on-site action force team leader takes appropriate actions quickly, such as the placement of these barriers. Fast decision-making is essential to minimize environmental damage and to protect people in the immediate vicinity. The team leader, as an expert, has the knowledge and experience needed to handle such situations.

However, traditional applications do not have an intuitive input interface, which would

enable a quick interaction. In these systems, the user has to insert inexpressive numbers into tables by hand if changes have to be done. This is tedious and slows down the workflow. Additionally the user needs to have special knowledge about how specific parameters effect the actual simulation.

In the majority of cases it is not possible to find the optimal solution on the first try. Therefore the user has to test different possibilities. Without an appropriate interaction interface, exploring alternatives might be cumbersome. A more convenient method than putting numbers into tables is a natural input interface like sketching.

1.3 Aim of the Work

To handle complex and crucial situations where time-critical decision-making is required, the process of examining alternative scenarios and finding the optimal solution should be as fast and as simple as possible. This is often essential for saving many lives and the property of people. The convenience should be achieved by using sketching as an input method of the steering system. Because of the similarity to writing and drawing, this is an intuitive way of communicating. Sketch-based interaction is natural and easy to understand and learn. The user should have the possibility to sketch directly into the visualization monitor of the simulation system. This allows for the modification of the environment to influence the fluid simulation indirectly or directly. To change the fluid indirectly, the user can set the boundary conditions of the simulation by positioning sandbags or other barriers into the scene. By setting pseudo-forces into the scene, the user has direct influence on the fluid. Due to the close interactive cycle of the system, the user should receive immediate feedback on how the sketches are going to affect the simulation. This enables quick responses to decisions made, which is essential for an efficient workflow. The important task is to answer the question of how to realize this feedback visualization to support the user as good as possible.

1.4 Methodological Approach

To achieve these goals a sketch-based interface is introduced in Visdom. Visdom is a steerable integrated visualization system for computational fluid dynamics, based on the data-flow concept. The system consists of modules called nodes which provide different functions. To enable sketching, interactors have to be introduced. Interactors allow nodes to interact with other nodes. One interactor should be established which receives user input data and constructs a spline. This spline then can be interpreted as boundary conditions or forces in the fluid simulation in Visdom. The user has the ability to interact with the simulation directly by sketching pseudo-forces into the fluid to realize effects like additional in-flow or wind. The user can also control the simulation indirectly by sketching barriers onto the terrain. According to a real-world procedure [79], sandbags are dropped from a helicopter to build the barriers. A real-time feedback visualization provides the user with information on how a sketch might influence the simulation. For the barriers, this can be a transparent preview of the sandbags, and for the pseudo-forces, glyphs, such as color coded arrows, can be used.

CHAPTER 2

Simulation Steering

Floods can cause tremendous property damage as well as threaten human lives when they occur in densely populated areas. Due to the unpredictability of inundations and other natural disasters they can appear suddenly and unexpectedly. This is also the reason why such incidences cannot be avoided. The only thing which can be done is to prepare as much as possible to minimize damage in case of an emergency. To be capable of doing so, experts of the responsible authorities have to understand the behavior of flooding. This knowledge supports the decision making in time-critical situations. To gain insight into the matter they have to explore the characteristics of floods which can be eased by using simulations in virtual environments. A sufficient framework should offer a suitable technique for simulating floods in real-time. Providing interactive steering should enable testing of decisions and analysis of their outcome. The user should be able to study the influences of individual input parameters like the velocity of the flood or some geometric boundaries. In an iterative exploration process the user should be able to test alternative solutions and receive immediate feedback about the impact. Intuitive and clear analyses have to be offered to give the expert a clue for new solution approaches which can be established in combination with previous knowledge.

2.1 Flood Simulation

For an efficient framework, which should help to investigate the characteristics of a flood, a suitable flood simulation tool is essential. The chosen method should allow for the replication of the conditions of the real-world scenario as accurately as possible. Furthermore, the needed simulation model should be flexible and should offer a good tradeoff between the speed of the computation and the accuracy of the results.

The standard approach for simulating floods and similar phenomena is the use of one- or two-dimensional hydraulic models. The Hydrological Engineering Center (HEC) of the United State Army Corps of Engineers (USACE) uses a one-dimensional river hydraulic model for an integrated package called River Analysis System [11]. The system provides hydraulic simulations and analysis of the flow in rivers and channels. Dhondia and Stelling [22] show an approach that combines a one-dimensional and a two-dimensional model for the simulation of flooding and overland flows. Using the combination they were able to bring the model's behavior closer to real physical behavior. Another existing model for flood modeling and flood forecasting was developed by Blöschl et al. [7], which is in operational use in northern Austria. Their model is distributed spatially and provides flash flood forecasting. Lamb et al. [51] implemented a fast 2D floodplain simulation model based on the JFLOW diffusion wave model using GPUs as high performance parallel processors.

The shallow water equations [77] are equations that describe the evolution of an incompressible fluid. They are derived from the well-known Navier-Stokes equations [8] and are a simple form of the motion equation that can be used to describe the horizontal structure of an atmosphere. Shallow water equations can be used to simulate urban inundations and dam-break scenarios with performance close to real-time. Mignot et al. [58] used the shallow water equations to model floods in an urban area. They implemented a solver for the 2D version of the equations to simulate floods in dense urban localities. Wenli et al. [97] solve the shallow water equations using MacCormack's predictor-corrector technique to simulate 2D flood waves caused by a dam-break. Another example of a dam-break flood simulation solver is provided by Song et al. [83] using unstructured meshes. Begnudelli et al. [6] and Liang et al. [53] use a Godunov-

based approach to solve the equations and simulate urban flooding and dam- and dyke-break scenarios.

Using the 2D shallow water equation has a crucial drawback. Since the model is just an approximation, it is not possible to apply it to all problems. The equation only describes one vertical level which enables the possibility to simulate large scale scenarios where the 3D fluid information is not important. To gain an exact solution and all the information about the evolution of the fluid, the full Navier-Stokes equations for incompressible fluids have to be solved.

The Navier-Stokes equations [8] are actually a set of partial differential equations which are valid throughout the whole fluid. The set consists of two equations, where the first one is called the momentum equation

$$advection = -pressure + viscosity + external forces \tag{2.1}$$

and the second one the incompressibility condition

$$divergence = 0 \tag{2.2}$$

The momentum equation again consists of three differential equations and can be derived from Newton's second law. This part of the Navier-Stokes equations tells us how the fluid accelerates due to the forces acting on it. How the quantities of the fluid or the fluid itself move with the velocity field, is called advection:

$$\rho \left(\frac{\partial \vec{v}}{\partial t} + \vec{v} \cdot \nabla \vec{v} \right) \tag{2.3}$$

where $\vec{v}$ is the velocity, ρ is the density and t is the time.

The three components describe the forces acting on the fluid. The first component represents the pressure existing inside the fluid:

$$\nabla p \tag{2.4}$$

where p is the pressure. Since ∇ is the gradient operator, this expression represents the change of pressure.

The second component is the viscosity of the fluid:

$$\mu \nabla^2 \vec{v} \tag{2.5}$$

where $\vec{v}$ is the velocity and μ stands for the viscosity.

Finally the third component represents the external forces acting on the fluid, like gravity:

$$\rho \vec{g} \tag{2.6}$$

where $\vec{g}$ represents the body forces and ρ is the density.

The second part of the Navier-Stokes equation, the incompressibility condition, indicates, that the velocity field of the fluid has to be divergence-free. That means that the volume of the fluid should be constant over time, which is a property of incompressible fluids. Divergence can be calculated with the equation:

$$\nabla \cdot \vec{v} \tag{2.7}$$

where $\vec{v}$ is the velocity.

One of the most sophisticated tasks of fluid simulation is to consider and to handle the boundary conditions [8] correctly. In comparison to the forces mentioned above, which determine the interior state of the fluid, the boundary conditions dictate how the fluid should behave on its boundaries. A boundary for a fluid could be a solid wall or a free surface, for example the boundary between the fluid and the not simulated air, or even the surface between two fluids. At a solid wall or at another solid obstacle, the fluid should not be allowed to flow into or out of the object. To achieve this, the component of the velocity which is normal to the boundary has to be zero.

$$\vec{v} \cdot \hat{n} = 0 \tag{2.8}$$

where $\vec{v}$ is the velocity and $\hat{n}$ is the normal vector at the boundary area.

This condition is called the “no-stick” condition. If the fluid should be viscid, this

condition is not enough, because the fluid should also be not allowed to move along the boundary. This leads to the condition that the velocity on the boundary should be zero

$$\vec{v} = 0 \tag{2.9}$$

which is called the "no-slip" condition. At a free surface boundary, the velocity should not be manipulated. Instead the pressure of the surrounding atmosphere has to be set to a constant value, for example zero. Boundary conditions are a physical phenomenon which are imposed by nature. When simulating fluids numerically an additional concern appears, namely the proper numerical implementation of the boundary conditions. The subject of proper and accurate boundary conditions is very important in Computational Fluid Dynamics [96].

The area of Computational Fluid Dynamics (CFD) uses numerical methods to solve the Navier-Stokes equations and to get a final numerical description of the complete flow field of interest [96]. Some methods simplify the equations by dropping the viscosity term to yield the Euler equations. The viscosity forces are important for the simulation of viscid liquids like honey or tiny water droplets, but for the majority of tasks, viscosity plays a minor role [8]. The traditional techniques from the field of Computational Fluid Dynamics provide good results when simulating flooding [87]. Gouda et al. [29] use a package based on the finite volume method to implement a dam-flooding simulation. Another traditional technique, the finite difference method [86] [96], achieves good results for flood simulation as well, but the calculation is too slow to provide an interactive user experience, especially for time-critical applications.

Every traditional Computational Fluid Dynamics approach is based on numerical methods. A numerical solution calculates the result of the equation only for discrete points in the domain, called grid points [96]. The domain of interest is subdivided into smaller cells which form a grid. In grid-based approaches the interaction between the cells can be described using the Euler or the Navier-Stokes equations. One example of the grid-based approach is the Lattice Boltzmann method (LBM) [84], which is a fast technique for viscous fluid dynamics. With this method, complex boundaries can be implemented

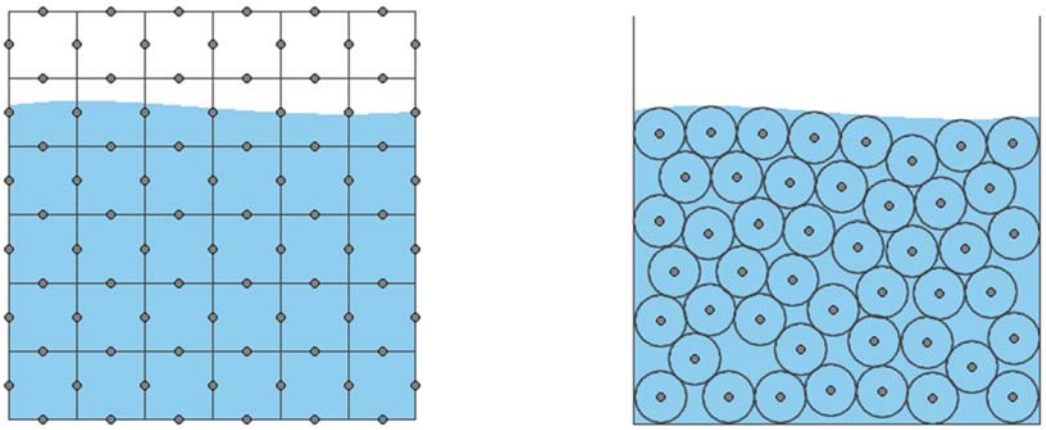

Figure 2.1: Comparison of the Eulerian (left) and the Lagrangian Viewpoint (right)

relatively easy. Judice et al. [45] compared the Lattice Boltzmann method with another lattice based approach, the Lattice Gas Cellular Automata technique, for the modeling and animation of fluids in computer games.

There are two different ways to simulate or even to look at a fluid (see Figure 2.1) [8]. The first way is called the Eulerian viewpoint. It is the viewpoint used in grid-based approaches, where the fluid motion is observed at specific locations in space through which the fluid flows as time passes. At this specific points, the fluid quantities, such as density, velocity or temperature, can be measured. The second approach involves examining the fluid quantities by following an individual part of the fluid as it moves through space in time. This is the particle-based Lagrangian approach. Müller et al. [63] present a less accurate particle-based method which allows for stable and fast computation of fluid effects. Particle-based approaches are preferred to Eulerian ones when simulation at interactive rates is desired. They are a good fit for the task of simulating flooding or similar phenomena for interactive visualization and analysis. The advantage of particle-based methods is that no time-consuming grid modeling is necessary, which also costs memory and computation time during the simulation. Furthermore, the geometric boundaries do not have to be voxelized, like when using grid-based approaches. A particle-based approach interprets the fluid as a set of particles. To realize the advection part of the Navier-Stokes equation in such a system, it is enough to use a so-called

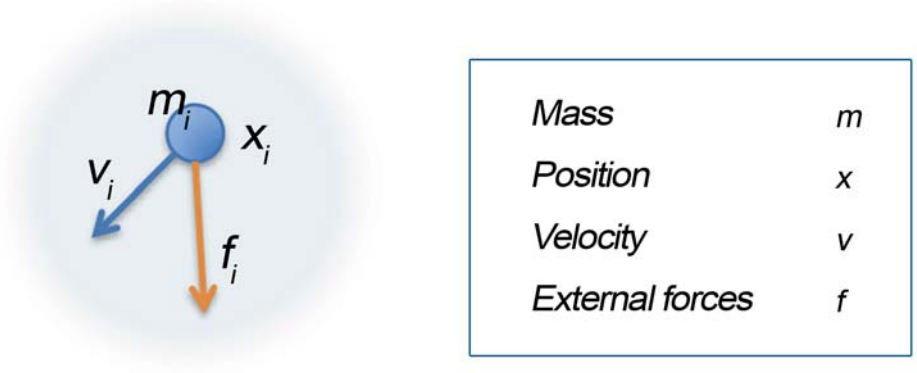

Figure 2.2: A particle object and the values stored

simple particle system [8]. This kind of system calculates the movement of the particles without any particle-particle interaction. Only a set of particles is needed where every particle stores its mass, the current position, the current velocity and the external forces acting on it (see Figure 2.2). Such a system can be implemented very efficiently, especially with the help of parallel computation on the GPU, and a large number of particles can be simulated. The only equations which have to be calculated for every particle are:

$$\dot{x}_i = v_i \tag{2.10}$$

and

$$\dot{v}_i = \frac{f_i}{m_i} \tag{2.11}$$

The first equation describes, that the change of the position of an arbitrary particle i ($\dot{x}_i$) is determined by its velocity (v_i). The second equation determines, that the change of the velocity of a particle i ($\dot{v}_i$) is the force f_i acting on it divided by its mass m_i. These two simple equations are enough for the implementation of the advection part.

If a flooding simulation would only be implemented with a simple particle system, the result would be very unrealistic and insufficient. It is essential that the particles influence each other and collide. There is a need to add additional forces to implement particle-particle interaction [8]. This pairwise interaction issue, also known as the N-body problem, is well known and can often be found when dealing with physical simulation. The brute-force approach of solving the problem is to calculate all the

interactions between two distinct particles. Nyland et al. [70] implemented this technique and even reached real-time performance. They achieved high efficiency by using the power of the graphics processing unit (GPU) via CUDA [17] and some other optimizations. Apart from the brute-force approach there exist many other techniques used to implement the N-body problem. For example the Fast Multipole method [5] can be used to avoid the calculation of forces between particles which are far away from each other. The magnitude of the interaction force between two particles only depends on the distance between these two. An approximation of this magnitude can be determined by the Lennard-Jones potential [69].

Smoothed Particle Hydrodynamics

As mentioned before, a realistic fluid simulation is based on the Navier-Stokes equations. Müller et al. [63] show, that it is possible to apply the equations on particles. The method they use is called Smoothed Particle Hydrodynamics (SPH). This method was originally developed in the field of astrophysics by Monaghan et al. [61]. Later, the method was adapted for use in the simulation of incompressible fluids. With SPH, it is possible to simulate flooding with real-time performance and to visualize it in a realistic way, as shown by Kipfer and Westermann [47]. They present an interactive technique to simulate and render rivers using the SPH approach. SPH can also be used for simulating fluid on a large terrain, which might be important for the implementation of flooding scenarios. Chatelain et al. [14] demonstrate this by periodically remeshing the particles using high-order interpolation kernels. Ghazali and Kamsin [28] used SPH to reconstruct the flash flood incident that struck Kuala Lumpur in 2007. The objective was to show that this approach is able to produce realistic simulations and renderings, and thus enable more precautions and countermeasures to prevent such natural disasters.

To apply the Navier-Stokes equations with the help of particles, the two equations mentioned above have to be applied to them. The first equation is the momentum equation, which forces the conservation of the momentum (see Equation 2.1). The second is the incompressibility condition, which forces the conservation of mass (see Equation 2.2). The equation of momentum can again be split in its three components. Due to the movement of the particles, the advection component corresponds to the change of the

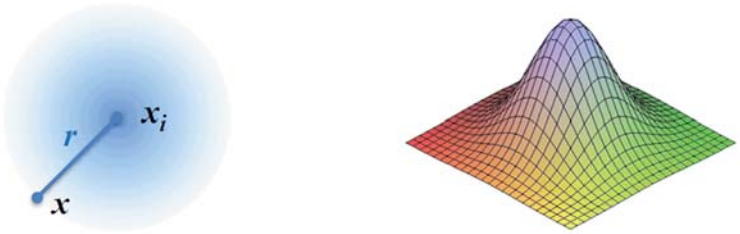

Figure 2.3: 2D (left) and 3D (right) visualization of the kernel function

velocity, thus the acceleration a_i of the particles [8]:

$$\left(\frac{\partial \vec{v}}{\partial t} + \vec{v} \cdot \nabla \vec{v}\right) \rightarrow \frac{d\vec{v}_i}{dt} = a_i = \frac{f_i}{\rho_i} \tag{2.12}$$

The incompressibility condition can also be simplified when using particles. Due to the fact that the number of particles in a system is constant and the fact that they always have the same mass during the simulation, the mass conservation is guaranteed. This holds if some rules are considered when setting up the SPH equations, which will be explained later in this chapter. To apply the resulting equation the SPH approach can be used. The basic idea of SPH is to build the necessary continuous global field over the whole fluid domain by summing up smooth local fields. Every particle induces such a local field. The most important component for the SPH methods is the so-called smoothing kernel $W_i(\vec{x})$, also known as the kernel function (see Figure 2.3) [8]:

$$W_i(\vec{x}) = W(|\vec{x} - \vec{x}_i|) \tag{2.13}$$

where $\vec{x}$ is an arbitrary point and $\vec{x}_i$ is the position of particle i.

The kernel defines a scalar weighting function around each particle. It is symmetric and its value at a certain position $\vec{x}$ depends only on the distance to the particle. Since the stability, accuracy and speed of the whole method highly depends on the smoothing kernel, choosing the right one for a specific purpose is essential. It is also possible to use different kernels for advection, pressure and the viscosity calculation [63]. A popular choice for a kernel function is the poly6 kernel:

$$W_{poly6}(r) = \frac{315}{64\pi d^9} \begin{cases} (d^2 - r^2)^3 & 0 \leq r \leq d \\ 0 & \text{otherwise} \end{cases} \tag{2.14}$$

where r is the distance between the particles with position $\vec{x_i}$ and an arbitrary position $\vec{x}$ (see left side of figure 2.3). The letter d stands for the core radius of the kernel. This value determines the region of influence of the particles.

With the help of the kernel function, a global density field $\rho(\vec{x})$ can be computed from the individual positions $\vec{x}_j$ and masses m_j of the particles using the following equation:

$$\rho(\vec{x}) = \sum_j m_j W(|\vec{x} - \vec{x}_j|) \tag{2.15}$$

The density for an arbitrary $\vec{x}$ is achieved by calculating the sum of the masses of all particles weighted by the kernel function. According to this, the density of a specific particle is the density calculated for the position of the particle. The density of a particle is important for calculating the quantity fields. First we come back to the conservation of mass issue. As mentioned before, one important property has to be considered. To guarantee the conservation of mass, the kernel function has to be normalized. This causes the integral of the density field to be the sum of all particles masses.

The next step is to calculate smoothed fields of arbitrary attributes $A_s(\vec{x})$, with the help of the density values ρ_j for the individual particles. This is realized by summing up the desired quantities A_j of all particles weighted by the particle densities ρ_j, the particle masses m_j and the kernel function $W(|\vec{x} - \vec{x}_j|)$:

$$A_s(\vec{x}) = \sum_j m_j \frac{A_j}{\rho_j} W(|\vec{x} - \vec{x}_j|) \tag{2.16}$$

The smoothed fields have a very important characteristic. Using this formulation, it is easy to calculate the gradient of smoothed fields by just applying the gradient operator ∇ only to the kernel function. This property is very useful when constructing the equations for the other forces to fulfill the Navier-Stokes equations. As mentioned above, the advection component corresponds to the acceleration of the particles. Since the acceleration is calculated by means of the forces acting on the particles, the three remaining components, pressure, viscosity and the external forces, can be interpreted as additional forces. Equations 2.17 and 2.18 show the transformation of the pressure and the viscosity components of the Navier-Stokes equations to the corresponding smoothed fields.

$$f_i^{pressure} = -\nabla p(x_i) = -\sum_j m_j \frac{p_j}{\rho_j} \nabla W(|x_i - x_j|) \tag{2.17}$$

$$f_i^{viscosity} = \mu \nabla^2 v(x_i) = \mu \sum_j m_j \frac{v_j}{\rho_j} \nabla^2 W(|x_i - x_j|) \tag{2.18}$$

Because these two forces are not symmetric and would contradict Newton's law "actio est reactio", the equations have to be adapted to achieve symmetry.

$$f_i^{pressure} = -\sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla W(|x_i - x_j|) \tag{2.19}$$

$$f_i^{viscosity} = \mu \sum_j m_j \frac{v_j - v_i}{\rho_j} \nabla^2 W(|x_i - x_j|) \tag{2.20}$$

The external forces, such as gravity or collision forces, can be applied directly to the particles, considering the density.

$$f_i^{external} = \rho_i g \tag{2.21}$$

When implementing a simulation using the SPH approach, a spatial data structure, such as a KD-tree, could be used to avoid comparisons of all particle pairs. The resulting algorithm only consists of three steps, which have to be done for every particle. Step one is the computation of the density. Step two is the computation of the force acting on the particle and the update of the particle velocity. Step three is the check for collisions and the update of the position.

One of the biggest advantages of SPH is that it does not need a grid. The constraints of a finite grid do not have to be considered. Unnecessary computations, for example in the case of empty grid cells, are avoided and the tracking of free surfaces is easy. Furthermore, the gridlessness has the advantage that the fluid can change shape easily. This feature is essential for simulating flooding scenarios, where levees can breach or barriers can be added to the scene, changing the area of the fluid. Due to the structure of the algorithm for SPH, the implementation is easy to parallelize, which enables the use of the power of the GPU [2] [32].

Some open source GPU-accelerated implementations of the SPH approach are available [35] [34]. The GPU-SPHysics code was used by Dalrymple and Herault to simulate a flooding scenario with various types of levee failures [19]. NVIDIA also provides a free and efficient GPU implementation of the SPH algorithm in their PhysX package [18].

NVIDIA PhysX

The NVIDIA PhysX package was initially developed to integrate physical animation and simulation into games. To guarantee high performance the package uses the computational power of the GPU. With PhysX, many different physical models can be simulated, for example rigid bodies, soft bodies, characters, vehicles, cloths and even fluids. The fluids component allows for the simulation of liquids and gases using a particle system. The use of the SPH method can be enabled or disabled. If it is disabled, a simple particle system is used. Another very important feature is collision detection. PhysX supports one-way and two-way interaction between fluids and other physical objects, like rigid bodies or cloth. These additional forces are very important for floods, namely to simulate the reaction of the fluid when it collides with geometry and the behavior of the geometry itself after the contact. To influence the simulation, NVIDIA PhysX provides the concept of force fields. The use of force fields will be explained in a later chapter.

Although the main focus of the engine is not accuracy, it allows the system to reproduce the case study of the laboratory with high performance. Additionally, NVIDIA PhysX implements all the functionality needed to simulate flooding scenarios with dynamic barriers and forces. Therefore this package is a good choice to be integrated as the simulation component of a simulation-steering application.

2.2 Steering Applications

After simulating the fluid with an appropriate method, the results can be rendered to visualize the flooding scenario. But the visualization alone is not enough to support decision making. To gain more knowledge about hazards and the behavior of the water,

the researcher should be able to modify the simulation by changing the environment or the fluid directly. This ability is essential for the analysis of how decisions made influence the flood. The traditional workflow of handling simulations is to prepare the input parameters and to execute the simulation. The visualization of the results is often implemented as a post-processing step. An application which runs these steps concurrently is more efficient, productive and provides more insight into the behavior of the flood. The user has to be able to change the input parameters interactively and receive visual feedback of the decisions made immediately. This is the basic idea of simulation steering, also known as computational steering [90].

Simulation steering is a powerful concept because the user, who should be an expert for floods, can guide the simulation into a desired region of interest. This could dramatically speed up the process of decision making in time-critical situations. Liere et al. [90] describe the requirements and the basic architecture of computational steering applications and introduce an implementation of the concept called Computational Steering Environment (CSE). Mulder et al. [62] provide a survey of Computational Steering Environments for scientific and engineering simulations, where they identify the scope and describe the architecture and the user interface of each environment.

Modular Visualization Environments (MVE) are a class of applications which use the concept of modules to simplify the access to complex visualization data [13]. A module is a building block of code which encapsulates its functionality in an object-oriented manner. Many modular visualization environments, for example AVS [40] or IRIS Explorer [64], are based on the visualization pipeline model [31].

The visualization pipeline consists of four main steps (see Figure 2.4). The first step is data acquisition. Data can be provided by measurements, for example from MRI scanners or computer tomographs, by modeling or by simulations, like flow simulations. After this step, the raw data can be passed on to the second step, the data enhancement. The raw data is modified to get the information that is necessary for subsequent steps. The data might be filtered to suppress noise, be interpolated to get additional values or be resampled to enable its use on a different-resolution grid. After the data is processed, the main step of the visualization pipeline occurs. The so-called visualization mapping

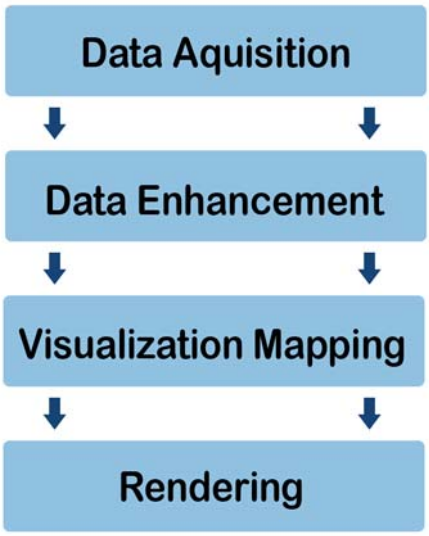

Figure 2.4: Visualization Pipeline

transforms the data into a renderable representation. The representation might be geometry, for example glyphs or icons, an iso-surface or even voxel attributes like color and transparency. The last step is the rendering of the mapped data to generate an image.

A Modular Visualization Environment provides modules to accomplish every step of the visualization pipeline. The biggest advantages of a modular approach are extensibility and flexibility. Functionality can easily be exchanged by using another module, which implements the desired functionality. The modules can be represented as nodes, with input and output connectors. The visualization pipeline can be established by connecting nodes in the correct order. The node for step one of the pipeline would then be connected with the node implementing the second step. This leads to a data-flow architecture where the data flows between connected nodes which all run as separate entities. The data-flow concept is commonly used for visualization systems [99]. An example of a steering application which is implemented using these concepts is SCIRun [73]. Miller et al. [59] implemented a distributed infrastructure for SCIRun to enable large-scale scientific computations. Although SCIRun provides some monitoring of running applications and allows for small modifications to be made to the data-flow network, it does not provide sufficient functionality for simulation steering [62]. Most data-flow

systems lack the ability to preserve states. Saving and restoring states is essential for iterative problem solving and also essential for supporting decision making. Without state handling, the user cannot compare alternative decisions made at a specific time and can not find out which one would be the better choice. GRASPARC [9] is a problem solving environment which implements a history tree to record simulation information during the computation to allow for backtracking. The HyperScribe mechanism is based on the GRASPARC idea and integrates the history tree as a data-flow module into IRIS Explorer.

Vetter and Schwan [91] implemented an application for steering physical simulations called Magellan. The main focus of this application is the high performance of the steering process itself and the ability to steer multiple applications simultaneously. Magellan uses a server-client architecture to separate the computation from the user interface. The user interface is text oriented, and steering commands have to be specified with a language construct called ACSL. Another server-client based steering application for physical systems is RealityGrid [10]. Using a server-client architecture can have many advantages. First of all, the server and the client can be implemented in different programming languages. They have different tasks and so also the requirements for the languages are different. Furthermore, this architecture enables a distributed implementation of the system, where heavy tasks can be split and solved by many computers in parallel.

To ensure sufficient support for interactive flooding scenarios, the application should be able to cope with computational fluid dynamics simulations. Kreylos et al. [50] describe a system which allows the user to explore computed solutions of a fluid simulation and find regions of interest. Afterwards, the grid can be refined interactively in these regions to get more details where needed. Matkovic et al. [56] use a system of multiple views, to enable computational steering for the development of automotive system components. The linked views are 2D scatter plots, 3D scatter plots, histograms and parallel coordinates. The views provide a comprehensive insight into the results of the simulation process. The linked views and the 1D computational fluid dynamics simulation are connected, which enables the user to see first results early. With this information and the help of brushing, new simulation runs with particular parameters

can be triggered for parameter ranges of interest.

The multiple views paradigm is another useful concept for steering applications while providing more insight for the user and helping to keep track of the simulation results. A distributed visualization and simulation environment for the analysis and forcasting of lake Erie is described by Marshall et al. [55]. POSSE [60] is another computational steering system for large-scale computational fluid dynamics simulations, the results of which can even be visualized using virtual reality facilities like CAVEs and RAVEs. SimVis [23] is a framework for interactive visual analysis of large, multidimensional and time-dependent data sets from computational fluid dynamics simulations using multiple heterogeneous views. Providing information and tools to support decision makers during the process of city-flood emergency-management is the target of the Flood Simulation and Decision Support System (FSDSS) described by Zhao et al. [101].

All the traditional systems lack intuitive and productive input interfaces. For example, the Computational Steering Environment uses text fields, sliders and buttons to receive user input [90]. The workflow of today's computational simulations is much more complex, with many different input parameters and large amounts of heterogeneous data results. In time-critical situations it is important to be able to interact quickly with the system. Furthermore, it would increase the productivity of an environment if the user can start more simulation runs, each with distinct parameters. This feature enables the user to compare and analyze many alternative scenarios, which help to make the right decisions.

2.3 World Lines

Due to their complexity, natural phenomena are difficult to predict. This is the reason why it is difficult to handle flooding scenarios even within steering applications. Also for domain experts with many years of experience it is very hard to assess the situation and to identify the hazards in advance. Even if a correct prediction would be possible, it would be hard to find the optimal countermeasures against floods. One possibility is to search for the best solution using trial-and-error. Within the application, the expert should be able to explore multiple scenarios simultaneously. Comparing a set of

related alternative solutions should provide more information about the whole scenario and should support the decision-making process.

World Lines [93] is an interactive visualization concept which allows the user to manage multiple simulation runs. The result of each run is determined by a specific decision of the user. The World Lines are intended as a control component in a system of linked views. The user can monitor various runs and even modify their settings to control the execution interactively. World Lines can also be used to visualize analysis results directly which supports the user's ability to understand complexity.

Every individual simulation run is represented as a track, which is visualized as a horizontal bar inside the World Lines view panel (see Figure 2.5). The length of the bar indicates the duration of the simulation run and the horizontal position determines when the run starts and when it ends. The horizontal dimension of the World Lines view panel can be interpreted as a timeline. Every track is distinguished by its own set of input parameters called settings. A track consists of many consecutive frames, where each frame represents the state of the simulation at a specific time value, considering the settings of the track. The so-called active frame is the frame of the selected track at the selected point in time and defines the data and settings used to update the linked views in order to visualize the simulation state. By defining the active frame, via selecting or moving the provided cursor onto it, the user can specify up to which point the simulation should be executed. The execution is performed when the record button is clicked. After the data is simulated the user has the ability to go back to any point in time to review the state of the simulation there. If the user encounters a point of interest, the system's settings can be modified in order to explore the scenario with different parameters. A user interaction, as well as an external data update, is registered by the system as an event. An event causes an action called branching to be triggered, which creates a new track. The new track starts at the point of time belonging to the actual frame and has the same settings as the parent track apart from the parameter or parameters which caused the branching. Afterwards, the simulation of both tracks is possible, which leads to the ability to navigate between these tracks to compare two or more alternative scenarios. The alternative tracks may overlap temporally implicating the existence of more than one frame for a given point in time. These frames are called parallel worlds for a specific time value.

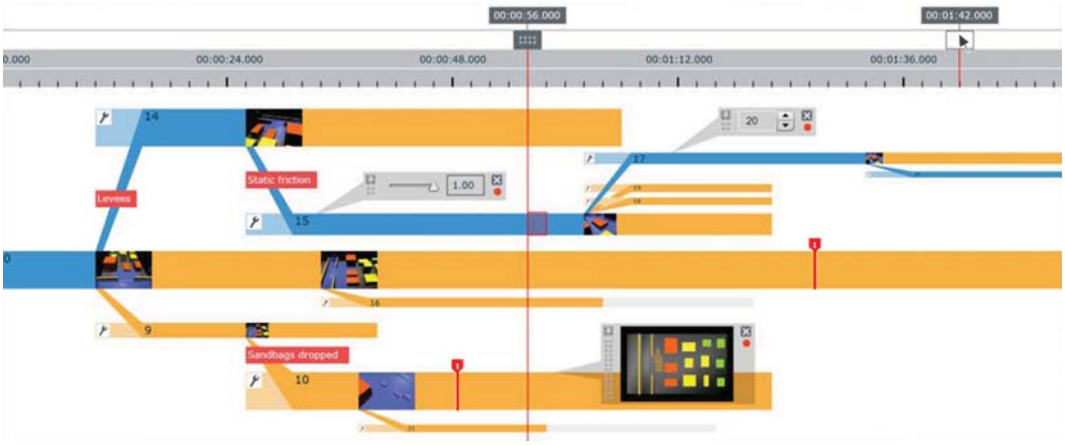

Figure 2.5: World Lines View [93]

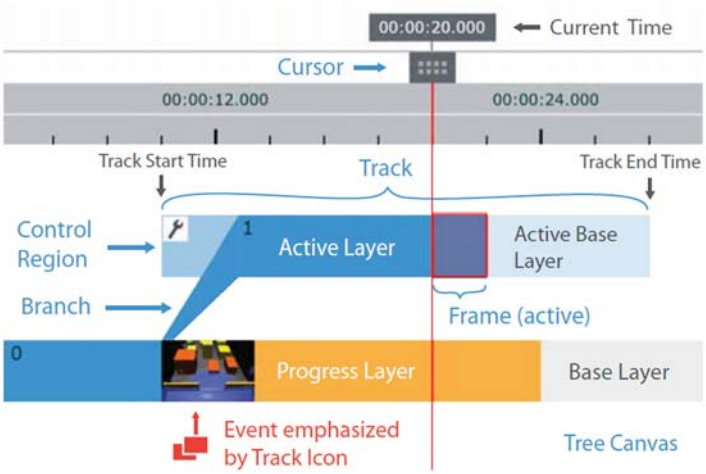

Figure 2.6: World Lines Layers [93]

A set of consecutive tracks, created by a sequence of events, is called a World Line and indicates a possible outcome (see blue tracks in Figure 2.5). The problem of finding the optimal solution is now mapped on the problem of finding the World Line with the desired outcome.

The World Lines are organized as a hierarchical tree where the tracks are the nodes and the branches are the connections between these nodes (see Figure 2.5). Each track can be uniquely identified by a number. The relationship between a parent track and

its branch is visualized by a connection called the incoming branch. The tracks and branches are color coded to indicate the progress of their simulation. The different colors originate from the different visual layers from which the tracks are composed (see Figure 2.6). The base layer represents the whole track, independent of its simulation progress. On top of it is the progress layer, which indicates which frames have already been simulated. The third layer is the active layer and highlights the selected World Line, which is currently in focus.

World Lines are regarded as a control component for a steering environment with multiple views which are synchronized with the active frame. The active frame is defined by the active track and the current time. To be able to navigate through time and parallel worlds, the active frame has to be accessible in a convenient way. The user has the opportunity to select the active frame directly by clicking on a desired frame anywhere in the World Lines tree. Additionally, there is the possibility to use the World Lines Cursor. This cursor is a vertical line designed to indicate the current time and to mark the active frame (see Figure 2.6). The cursor has a box in order to enable dragging it to a desired position. When the system is recording or playing back the simulation, the cursor always surrounds the active frame along the active World Line. The user can choose to follow the active frame to avoid the cursor leaving the view. To concentrate on some specific regions of interest the user can zoom into the World Lines view horizontally and vertically using scrollbars. If the user creates many tracks while experimenting, World Lines provide a mechanism which resizes and rearranges the tracks automatically. If the user is unsatisfied with the result, the layout can be changed manually. The thickness and vertical position of the tracks can also be manipulated. Uninteresting tracks can be collapsed into their parent track to keep the view clean and clear.

The World Lines view provides two modi, one for managing the simulation runs, called steering mode, and one for comparing them, called the visualization mode. In the steering mode, the user can generate, create, and manipulate multiple simulation runs. The advantage of the multi-view framework can be exploited by steering in one of the linked views. An example is a 2D view where the user can place geometric objects into the scene. It is also possible to change the settings of a track without creating a branch. This feature is important for situations where a change has to be applied to the base

Figure 2.7: World Lines visualization mode to display analysis results, for example the number of flooded buildings, per frame. Green frames indicate time-steps where few buildings are flooded, whereas red frames represent a high number of flooded buildings.

track and the whole subtree. Since all tracks, apart from the root track, are based on a parent track, a change in the settings of the parent track invalidates all of its child tracks. This leads to the invalidation of all subsequent tracks because the calculated simulation data in these tracks is no longer useful. The data has to be deleted and simulated again with the new settings.

The second mode is the visualization mode in which the user gets an overview showing the outcomes of the simulation. While handling the simulation runs is a very important task, analyzing the results is even more important. World Lines provide a comprehensive visualization view to enable interactive visual analysis (see Figure 2.7). A traditional method used by World Lines is Linking and Brushing, used by many visual analysis applications [46]. World Lines visualizes the simulation data using multiple views which are linked together. The linking ensures that the views are synchronized. If some settings change, all views are updated to have the focus on the same data. Brushing allows the user to interactively select a specific region of interest in the views. Due

to linking, all corresponding regions in the other linked views are highlighted as well. World Lines uses brushing to select a set of tracks to visualize and compare the outcomes in parallel worlds.

The simulation results can be analyzed with different mechanisms. By default the per-frame analysis is used, where the system is synchronized with the active frame and shows the results for one track at the current point in time. World Lines supports three different visualization methods. For our purpose, the most important method is the frame-wise visualization. With this option the user can see the changes of the analysis result as the simulation progresses. Every frame stores its own value and is colored according to a transfer function.

Using World Lines reduces the effort of exploring multiple scenarios significantly. A quick comparison of many possible actions is essential for handling flood situations. With World Lines, a domain expert has the ability to test many countermeasures based on previous knowledge to quickly find a good solution. While navigating through a set of alternatives and comparing them, the expert should gain more information about the performance of each action, which enhances decision-making. World Lines should be used as a control component within a comprehensive simulation and visualization system to unleash its full power.

2.4 Visdom

A simulation-steering system which provides direct simulation analysis and interactive steering could support the user in the decision-making process. The system should be flexible and clear, to enable adaption to various problems and to guarantee easy handling. On the other hand, it should be comprehensive and expressive to provide the best possible support for the user in critical situations. The World Lines concept is implemented as a module in the steering and visualization system Visdom [93].

Visdom [1] is a steerable integrated visualization system for computational fluid dynamics simulations which is developed by VRVis Vienna together with the ETH Zürich. It provides all the features needed for effective problem solving, such as clear interfaces

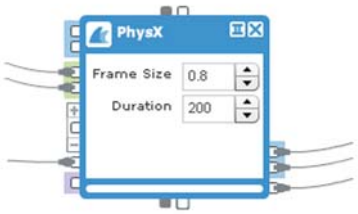

Figure 2.8: The visualization of a node as a box, in this case the Simulation Node. On the left side are the input connectors and on the right side the output connectors.

to control simulations and comprehensive visualizations to view and analyze the results. Using the power of the GPU, Visdom is an effective tool to handle simulations. Visdom is highly modular and supports comprehensive analysis via multiple linked views. This framework simplifies steering simulations and enables quick knowledge gathering and understanding of complex scenarios.

Visdom is implemented as a server-client system which enables web-based usage. This architecture could also enable the involvement of other web services and online data like hydrological and geological information. The server is responsible for the management and simulation of the data, and for the rendering of the results. This means all the heavy computations are handled by the server, which allows the client to be a lightweight application. The client can thus be used on a mobile device, as no extensive resources are required. The client's responsibilities are handling user interaction and controlling the system. If the client needs something to be calculated, it sends a request to the server. The request is an XML structure with information about the desired task. The server completes the task and sends the result back to the client. The used programming languages are different for client and server, and chosen to fit the tasks of the components. The server is implemented using C++ to guarantee high performance which is also assured by the use of CUDA [17]. The client application is based on Adobe Flex [43] which uses MXML for defining layouts and interface behavior, and ActionScript for the logic implementation.

To achieve a high degree of flexibility, Visdom is designed exceedingly modular. The framework is implemented as a node-based system using the data-flow concept to con-

trol the work-flow. Every functionality is encapsulated into a node. A node can be seen as a black box with a well defined input and output interface (see Figure 2.8). Every node processes the input according to its purpose and produces output data. Some nodes do not have input connectors. These nodes are called producers and are used to provide data by creating it according to settings and user inputs, or loading it from a file. The last nodes in the data-flow pipeline are usually view nodes. These nodes have no output connectors and are responsible for the rendering and the display of the results. An example of a node which has input and output connections is the simulation node. The simulation node receives inputs like the geometry or the terrain used for calculating collisions, and outputs the positions of the particles and the velocities for each. Currently this node is implemented using PhysX as an engine, but due to the modularity of Visdom every other simulation method can be integrated. The node-based architecture enables easy integration of new functionality by just implementing a new node representing a new module or an existing external component. In addition to the data-flow, Visdom provides a so-called meta-flow [94], which flows vertically whereas the data-flow flows from left to right. Through the meta-flow, the normal data-flow modules are connected with special configuration modules. These modules have the ability to control other nodes. The meta-flow can also be established between two data-flow nodes in order to add another communication channel. Each node has its own settings and additional to the data-flow output it can produce other results. Both of these new properties can be sent over the meta-flow to communicate with other nodes.

The Visdom client provides two views, a design view for establishing the data-flow and a semantic view for steering and analyzing the simulation. The design view (see Figure 2.9) is an interactive flow diagram to set up a steering system according to the task. The user can construct the data-flow system by inserting nodes and connecting them. This interface is very intuitive and enables a quick creation of the desired work-flow. The settings of the nodes can be accessed in the design view by clicking on the desired node icon. The settings determine how the node processes its input, and the user can change the way the node computes the output by modifying the parameters. Some of the nodes used in the data-flow pipeline create semantic windows. A semantic window is either a view, which displays the result of the simulation, or an interactive component. In the semantic view (see Figure 2.10), the user can see all the semantic windows. They are

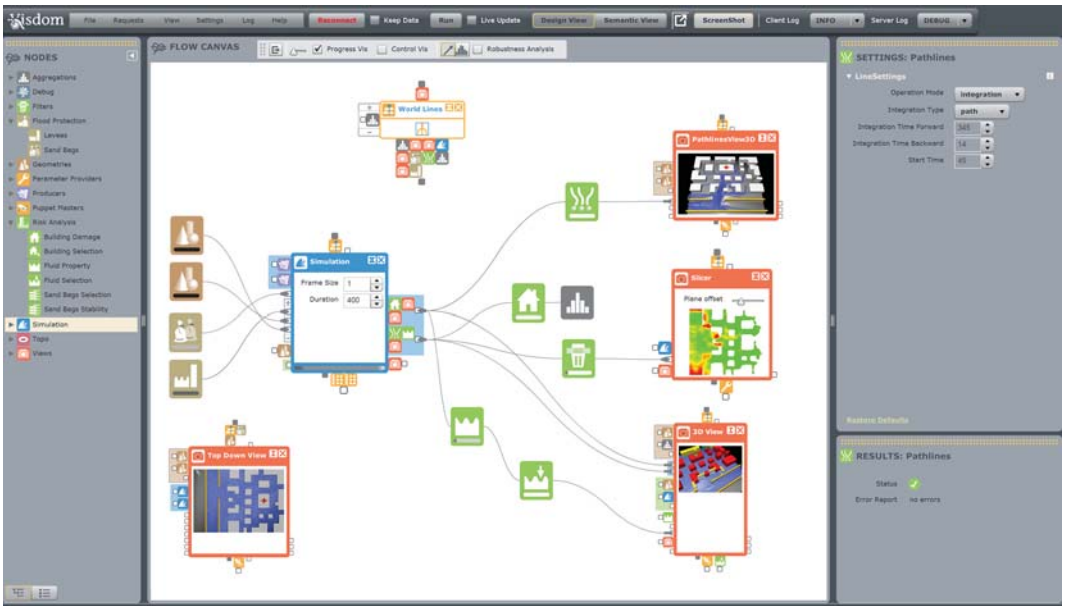

Figure 2.9: Visdom Design View: The center panel provides a canvas where the data-flow can be established. The nodes are accessible from the left panel. They can be added via drag-and-drop. The settings of a selected node is shown in the right panel.

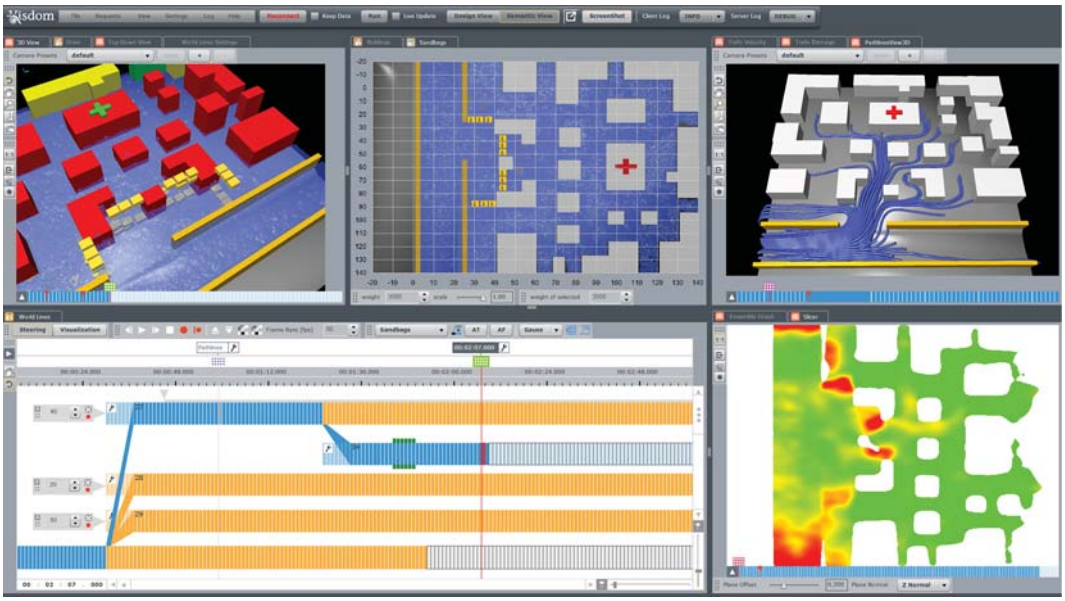

Figure 2.10: Visdom Semantic View: Shows the window layout created by the flow diagram in Figure 2.9

created automatically by the data-flow which is established in the design view. Examples of views are the 3D monitor and the 2D slicer view. An interactive component provides the ability to change the settings of a node via a visual user interface. An example of such a component is the transfer function view. This view is created by the transfer function node, which outputs a vector representing the transfer function. The user can alter the function in the provided semantic window by choosing colors and opacity values which should be used for specific data values. Each semantic window listens to the node which it was created from to update its content if the corresponding node's settings or the settings from a linked node are changed. The semantic windows are linked together. If something is changed in one window, all the other windows also get updated to correspond to the new state of the system.

Visdom is the ideal system to be used with World Lines. As mentioned before, World Lines should be used as a control component to add functionality and new features to a simulation and visualization system. With the concept of the meta-flow, Visdom has the perfect infrastructure for it. From a design viewpoint, World Lines are implemented as a configuration node using the meta-flow to control specific nodes (see Figure 2.11). From the semantic point of view it is another interactive component, so if the World Lines node is created in the design view, a semantic window will be created and integrated into the semantic view panel. The World Lines view provides, as explained in the previous chapter, the functionality to create simulation runs, to manage them, and to analyze the results.

This resulting system consisting of Visdom and World Lines allows for reproducing the flood scenario case study of New Orleans which was the aftermath of hurricane Katrina. The system can be used to investigate the scenarios in a virtual environment to gain insight and to be able to test and evaluate breach-closure procedures. During the development of Visdom a test scenario was established, where a village situated near a river is protected by a levee. The system supports the user handling situations where the levee breaches. The resulting flow diagram can be seen in Figure 2.11. First of all, geometry is needed to fill the scenario with the terrain and the levee. Visdom provides many nodes which are able to produce geometry, for example, the Model Loader Node (second brown node in Figure 2.11) which loads a mesh from a file. This node can be

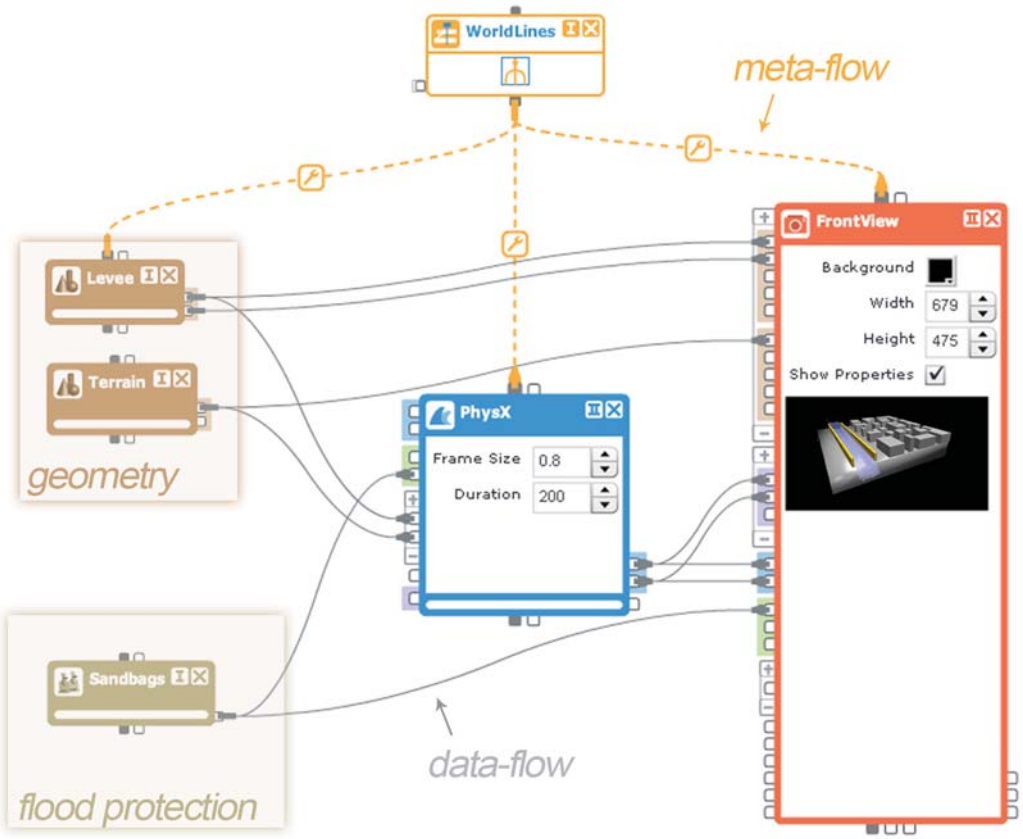

Figure 2.11: Simple Network (Design View). In the center is the Simulation Node (blue). Geometry Nodes (brown) and Flood Protection Nodes (ocher) are connected as inputs. The output of all nodes is sent to the View Node (red) for visualization. The World Lines Node (white with yellow border) controls all other nodes via ropes.

used for the terrain if geological information is available from some source. For simple or dummy objects the user can create the geometry using the Model Designer Node (first brown node in Figure 2.11) and an interactive component. For integrating the SPH simulation, a simulation module, namely the PhysX Node (blue node in Figure 2.11), has to be added. The previously created geometry nodes have to be connected as input to the simulation node to enable correct collision calculation. Afterwards, a view node, in our case the OpenGL Node (red node in Figure 2.11), has to be provided to visualize the scenario. All geometry nodes and the simulation node need to be connected to the view node in order to affect the visualization. Now we have a scene, where a constant ge-

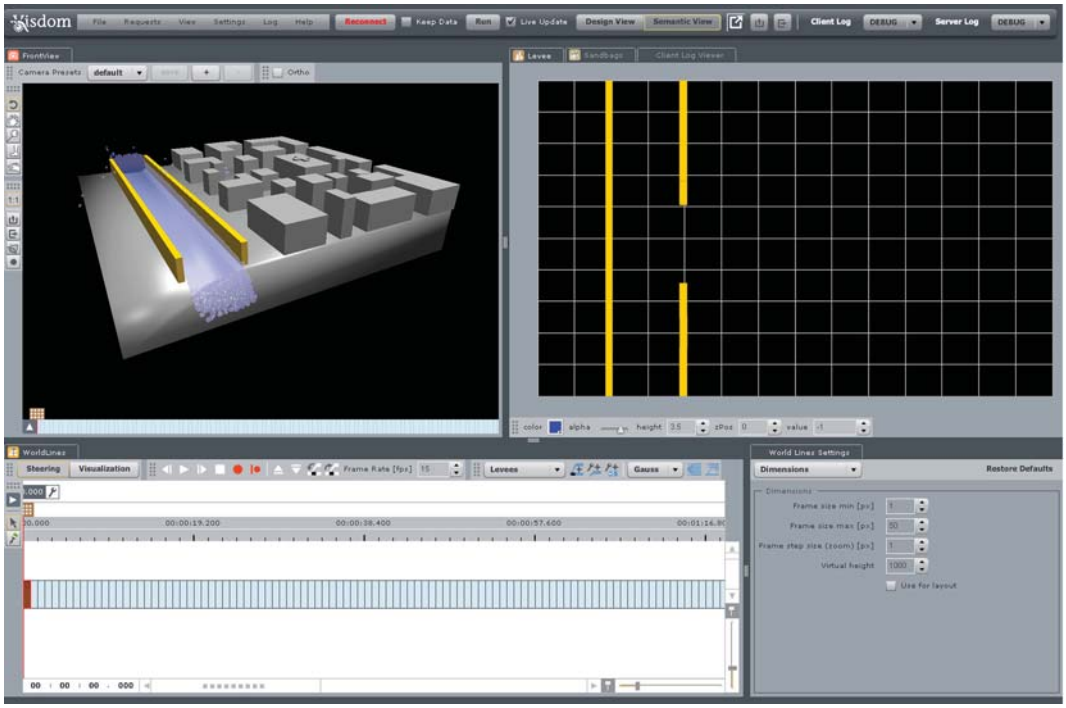

Figure 2.12: Simple Network (Semantic View): Shows the window layout created by the flow diagram in Figure 2.11

ometry exists, consisting of a village and a levee with a breach. To test breach-closure procedures, the ability to drop sandbags interactively must be provided. This can be achieved by inserting the Bag Model Designer Node (ocher node in Figure 2.11) into the data-flow network. The output geometry of this node has to be delivered to both the simulation node and the OpenGL node. The simulation node needs the geometry for collision calculation and the OpenGL node needs it for rendering. Finally, the World Lines Node (white node with yellow border in Figure 2.11) is added to the data-flow and a meta-flow is established to control the other nodes.

To work with the developed system, the user has to switch to the semantic view. Some of the chosen nodes create semantic windows which are accessible there. The geometry nodes and the simulation node do not create views, so settings have to be used if the user wants to change their behavior. The OpenGL node creates a 3D monitor which displays the visualization of all objects connected to the node. To navigate through the scene in

the 3D monitor, tools like pan, rotate and zoom are provided. The Bag Model Designer Node creates an interactive component. It is a 2D view showing the horizontal projection of the entire scene. By clicking the user can position several sandbags which are represented by simple rigid boxes to increase performance. The last important semantic window is the World Line view, which is an interactive component responsible for the steering functionality of the system. Figure 2.12 shows the semantic view corresponding to the data-flow in Figure 2.11.

To test countermeasures for the levee-breach the user first has to simulate the flooding scenario. As mentioned above, all semantic windows are linked together, so if the user adds some bags by means of the Bag Model Designer view, both the 3D monitor and the World Lines view get updated. The 3D monitor shows the created sandbags rendered as boxes dropped into the scene. In the World Lines view, a new branch is created at the current point in time indicating the decision made by the user. An important value for comparing countermeasures is the number of flooded buildings in the village. This value can be visualized directly in the World Lines view. The user can also use more than one frame to analyze the simulation. For example, the path, which specific objects would follow while floating in the fluid, can be analyzed and visualized by combining the information of several frames [80]. Additionally, the user is able to combine even multiple frames of multiple tracks to perform comparative visualization [78] which enables concurrent visualization and comparison of alternative simulation runs with different parameters.

Due to the flexibility and functionality of the system, many scenarios can be reproduced and analyzed in real-time to support decision making. For this reason, Visdom is a powerful framework with several fields of application [92]. First of all, the system can be used for offline planning of actions to mitigate the consequences of flooding, or other actions which should be taken if a flood actually occurs. This means Visdom is capable of creating the flood-risk management plans ordered by the EU [24]. Another offline application is the training of on-site action forces which have to handle emergency situations caused by floods. Due to the possibility to test alternative decisions and analyze the outcome, action force members learn to cope with the problem and gain more insight. Additionally, they are given a better understanding of the behavior of the

flood and the influences of decisions made. The third field of application is the use in on-site operations during floods. Used on a mobile device, the system could support the action-force team-leader in decision making. After loading geological and hydrological data, previous knowledge can be used to try several approaches solving the problem and analyze them to find the one which minimizes the damage in the vicinity.

Chapter Summary

Visdom is a powerful and comprehensive and nevertheless flexible and intuitive application which combines simulation, visualization and analysis of computational fluid dynamics. It can be used in offline and online situations to support the user in the decision-making process. If the user is searching for a solution using the trial-and-error approach, it is essential that the system allows for interaction in an intuitive and quick way. Especially in time-critical cases of operation, during a flood or when imminent danger is present, a sufficient solution must be found as soon as possible to save as many lives as possible.

CHAPTER 3

Sketch-Based Interaction

For efficient and productive steering applications, user interaction is an essential part of the workflow. Therefore, the selection of an appropriate interface has to be well-considered. Most traditional steering applications offer input fields and buttons to the user as a means of setting the parameters. Working in this way with complex scenarios can be very cumbersome. Visdom provides inline widgets and the ability to modify the values by means of two-dimensional linked views and interactive components. This improves the usability significantly, but for some specific tasks the interaction could be even more natural to increase working speed and efficiency at the same time. If the user is currently working on a flood scenario and wants to test different barrier arrangements, the individual sandbags have to be created in the Bag Model Designer. This interactive component allows for the creation of only one bag at a time, making the creation of a whole arrangement a tedious task.

The main contribution of this thesis is the augmentation of the Visdom application by implementing a sketch-based user interface for specific tasks. This has been accomplished during an internship at the VRVis Zentrum für Virtual Reality und Visualisierung Forschungs-GmbH in Vienna. The goal is to allow the user to create desired barriers or to introduce forces by sketching into the 3D view directly. To realize this within the data-flow based system, a new concept of modular interactors has to be introduced. They make use of upstream communication between nodes. It is necessary to provide

visual feedback of how the user's sketches affect the simulation. To visualize the stroke, a spline is displayed in the 3D scene, which can be manipulated afterwards. The effect of the spline, which depends on the further usage of the stroke, is visualized directly inside the view. The feedback visualization should support the user by providing feedback faster, which is especially important in time-critical situations. The interface has to be useful without additionally complicating the workflow.

3.1 Related Work

According to van Dam [89] the evolution of user interfaces can be subdivided into four stages. During the first phase, there was no real user interface because in the early 1950s the computers were controlled by punched-cards. For a single program many of these cards were needed. The users had to prepare the cards as a preprocessing step and after finishing, they stacked them into the card reader and ran the computation. The result of the program was printed using a line-printer. Due to this type of processing, called batch-mode processing, no real interaction was possible during the computation. In the early 1960s the user was able to interact with the computer by typing in commands with parameters for the first time. This type of interaction was also used years later by operating systems such as DOS and Unix. The third phase was the most important one because it was the beginning of the graphical user interfaces which are used nowadays. Xerox developed the first workstation using the so-called WIMP paradigm. WIMP stands for window, icon, menus and pointing device, which are the components of the user interface. Apple popularized this paradigm with the Macintosh which introduced the menu bar and window management. The for the present last phase, which van Dam called the post-WIMP stage, started in 1990. New user interfaces were developed which tried to go further and use more natural paradigms like gesture and speech recognition.

At the time when WIMP was introduced, the focus was on the functionality and the performance of the system and thus of the user interface. When the computer started to be used in every household, the importance shifted to the usability. A user interface had to be clear and easy to learn but also productive and efficient. However, WIMP interfaces tend to become very complicated when the complexity of the application increases. Users often have to pay more attention to the interface than to the task. Another

disadvantage of WIMP is that it is not suitable for interacting with objects in a 3D environment. The paradigm would map all 3D controls to traditional GUI widgets. The user would have to press buttons, use sliders or input numerical values into textfields to manipulate the object, which is accurate but not very intuitive. Furthermore, no interactive feedback visualization is possible if several parameters have to be set before the execution of the modification. Thus, the user has to execute the command again and again until the desired result is produced.

A more advanced solution is to provide the ability to directly manipulate the 3D object. For example widgets which are part of the 3D world could be used. These can allow to modify the object in a specific way while guiding the user visually. An even more intuitive way of communicating with the computer is to use sketching via freeform-user interfaces [36]. Using WIMP interfaces, the parameters have to be set to exact values, which is extremely cumbersome or even not suitable for many tasks. If an idea or request could be sketched directly, it would enable a quick and fluent interaction between the user and the computer. This workflow is perfectly suited for problems where the user has to explore different approaches quickly to find a sufficient solution. The user should be able to concentrate completely on the task without being distracted by the interface.

Sketch-based interfaces are a natural way of interacting with computers. Drawing strokes is familiar to almost everyone and therefore a convenient and efficient way to express requests. Sketch-based interfaces are easy to learn and to use due to their intuitive nature [98]. According to van Dam [89], in history user interfaces were always optimized to the available hardware. Keeping this in mind, we expect that the steady replacement of traditional mouse-keyboard interfaces by more natural devices like touch-sensitive ones would lead to an increasing interest in sketch-based interfaces. Devices which use touch inputs are on the market no less than ten years. The first exponents were pen-based tablet PCs which used stylus interaction to provide more direct control [54]. In the last years, the market for multi-touch devices like smartphones and tablets is booming. This devices use intuitive gestures like wiping to navigate through a menu.

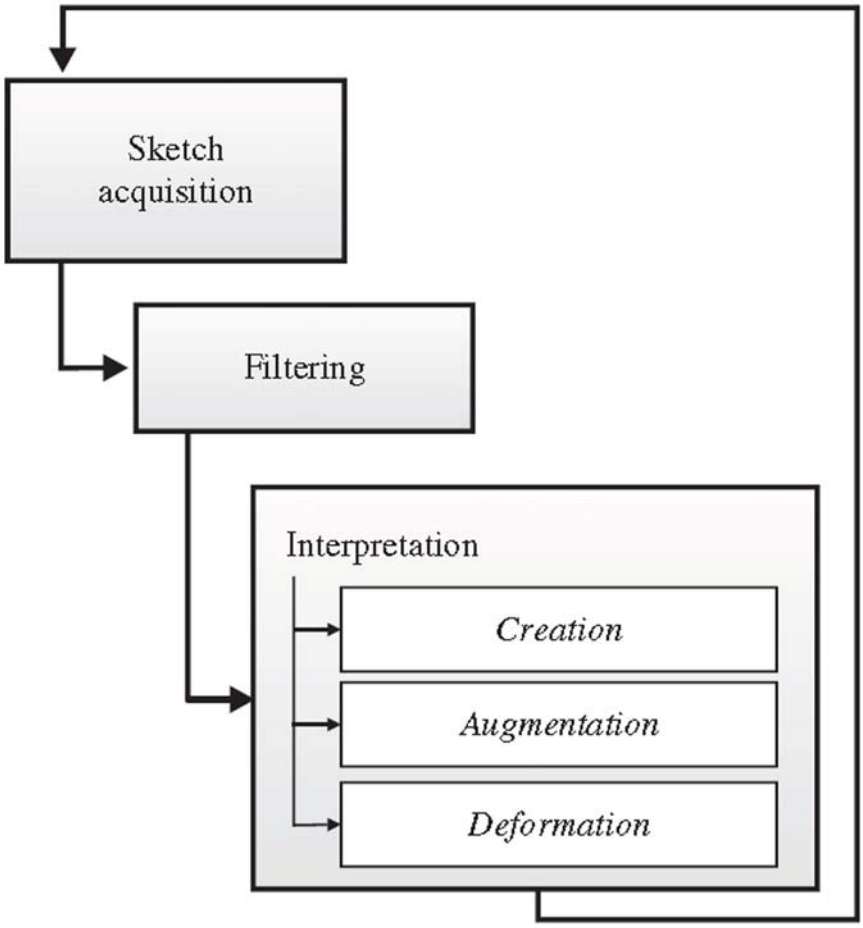

Figure 3.1: Pipeline of a sketch-based interface for modeling [72]

Olsen et al. [72] provide a comprehensive survey of sketch-based modeling, where they also go into detail about the common architecture. All sketch-based approaches are based on a pipeline which consists of three stages: sketch acquisition, sketch filtering and sketch interpretation (see Figure 3.1).

Sketch Acquisition

At the beginning of every interaction, the system has to acquire the sketch from the user. A sketch can be composed of one or many strokes, where every stroke is a set of 2D points. The mouse is a common device for the acquisition of user input, but multi-touch screens are more suitable for freehand strokes because their usage is similar to drawing or writing on paper. The distribution of the points along the stroke depends on the speed of the user's drawing. When the user draws quickly, fewer points are passed to the system due to the regular sampling rate of the devices. The points are represented mostly in window coordinates. In many cases, namely when the user wants to interact with a 3D scene, this representation is not sufficient. The mapping of a 2D stroke into the 3D world is not trivial but essential.

Sketch Filtering

The obtained points cannot be used directly for the interpretation because of their irregularity. Furthermore, noisy or erroneous samples can exist, caused by the user or the used device. Thus in the second stage, the strokes have to be filtered for further use. This stage can contain more than one step necessary to clean and transform the input. An often used filter method is to resample the stroke. This has the effect that the spacing between the points becomes more regular and that the noise is decreased. Igarashi et al. [39] removed undesired points by resampling the input to form a smooth polyline, which is a coarse approximation of the original stroke. The raw input can also be smoothed to get rid of the noise in the signal by applying a filter, for example a Gaussian or a similar one [85]. Another possibility of simplifying the stroke is trying to find an approximation of the original input in another representation. Curve fitting is a common example, where the simplification is achieved by using a curve defined by some control points which correspond to the original input. The appropriate representation can be found using different curve-fitting algorithms like polynomial curve

fitting or least-squares polynomial fitting [49]. A better approach is to fit the points to parametric curves like Rational Bezier curves [74] or B-splines [88]. For all these approaches it holds, if the stroke has to be reconstructed, the user has to evaluate the curve, which leads to a computational effort. Another possible preprocessing step is a technique called beautification [38]. This approach tries to interpret and compose the users strokes considering geometric constraints like symmetry, parallelism or linearity. If the user draws a stroke and wants to modify it by sketching some parts anew, a technique called oversketching has to be implemented [26] [57]. This feature allows the drawing of arbitrary free-hand strokes and preserves important sections which would possibly get lost due to the fitting process. The method introduces a smooth transition between the old and the new stroke section.

Sketch Interpretation

After acquiring and preprocessing the raw points, the next step is to interpret the input. This step depends greatly on the task and can be complicated due to the potential ambiguity of free-hand strokes. This holds especially for 2D strokes, which should be interpreted as commands for a 3D scene. However, in all cases the system has to identify the stroke and run the action corresponding to its meaning.

In the past, sketch-based approaches were used for many different tasks. For example, SILK [52] is an application which supports the user in creating user interfaces. The main idea is based on using early sketches of ideas for an interface. Interface designers are used to drawing some scribbles in order to discuss ideas during a brainstorming session. SILK allows the designers to sketch a draft of an interface directly into the system using a tablet and a stylus. This drawing is flexible and interactive, so the user can modify it if necessary or illustrate behaviors. Finally, the draft can be transformed semi-automatically into an operational interface. SILK uses 2D sketches which are interpreted only as commands for a 2D world, as interfaces are usually planar. Schroeder et al. [81] use a sketch-based interface to allow artists to draw into an illustrative visualization of a 2D vector field. For example, they can add additional streamlines or crop some if they disturb the overall impression of the visualization. Another widely spread use for sketch-based interfaces is the modeling of three-dimensional objects. Tradition-

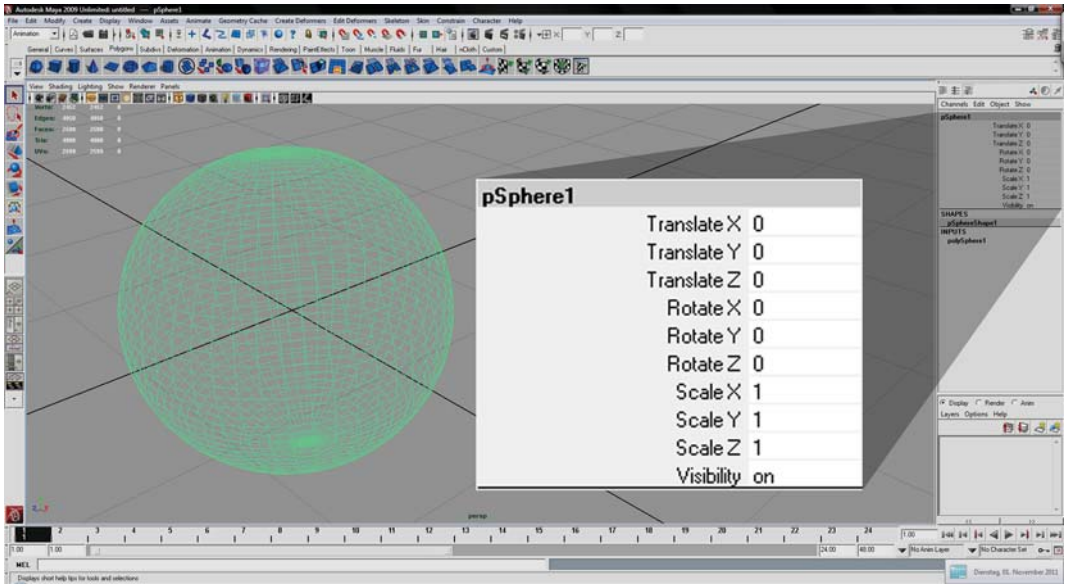

Figure 3.2: WIMP-based Interface in Autodesk Maya [41]

ally, a 3D modeling application like Autodesk Maya [41] has a WIMP based interface to provide accuracy for high quality production. The artist creates objects by constructing a mesh consisting of vertices which can only be modified by moving them individually. The translation, rotation or scale values can be set by entering parameters into the input fields of the interface (see Figure 3.2). This approach is very accurate and allows for the use of scripting for advanced modeling. However, modifying a complex mesh solely in this way is extremely cumbersome and the workflow is not easy to learn and to remember. Therefore, a new set of applications appeared, which use sketch-based interaction for a more intuitive and efficient way of modeling objects. Autodesk Mudbox [42] is an example of such a sculpting application, which tries to imitate the workflow of a sculptor. The artist can paint details onto the surface directly in an intuitive and natural way (see Figure 3.3). This kind of application is actually able to lift or lower some parts of a mesh. Thus a mesh, or at least a basic shape of the object, must already exist.

A more complex approach is the creation of 3D objects from scratch using sketching interaction. Instead of starting with a basic shape like a sphere or a cube, which is often used for modeling, the user sketches the desired shapes which are then automatically transformed into a three dimensional mesh by the system. Sketch-based modeling ap-

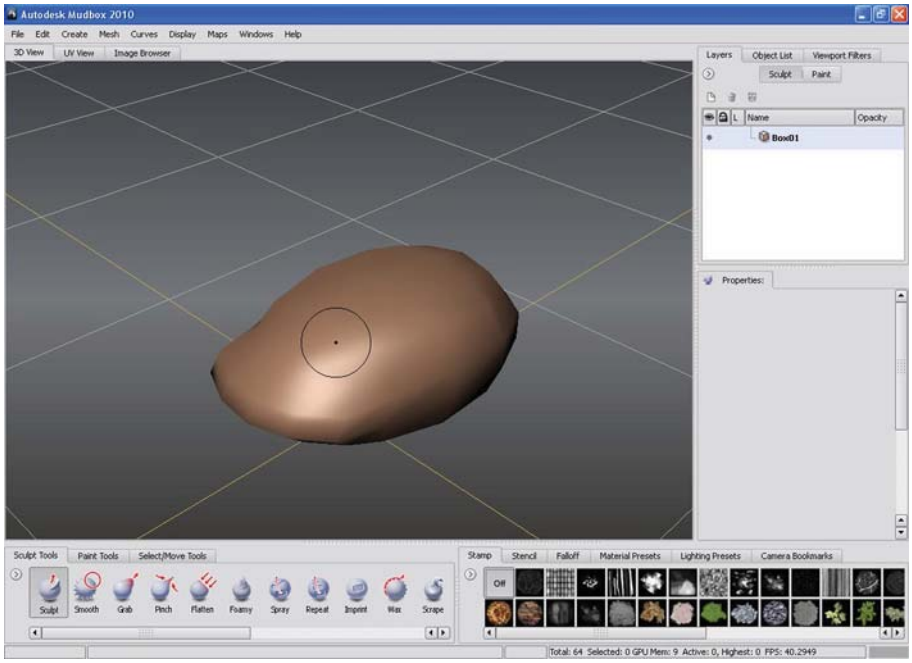

Figure 3.3: Sketch-based Interface in Autodesk Mudbox [42]

plications aim to increase productivity and efficiency through the use of more natural and therefore faster user interaction. These systems can be used to create low-detail models for rapid prototyping or design work [72]. For modifying the model, no vertices have to be touched, the user just oversketches the previous shapes or uses special strokes to trigger commands. Teddy [39] is an application which works this way, providing operations to create, augment, erase, extrude, smooth, transform and cut geometry with simple strokes. When the user sketches a shape, the system inflates it automatically to create the three dimensional object. Another famous sketch-based application is SKETCH [100] which follows a different approach. This system tries to recognize the strokes and interpret them as a command to create a predefined object. In this application, just basic shapes are used, but other systems provide complete models which are stored in a database [82]. Nealen et al. [67] present a method which enables the modification of an existing mesh by selecting an area from a specific viewpoint and drawing a new silhouette for this part (see Figure 3.4). Another approach by the same author is provided in the FiberMesh system [66]. After creating a simple and rough 3D model, the user can refine the mesh by sketching curves directly onto the mesh. These curves

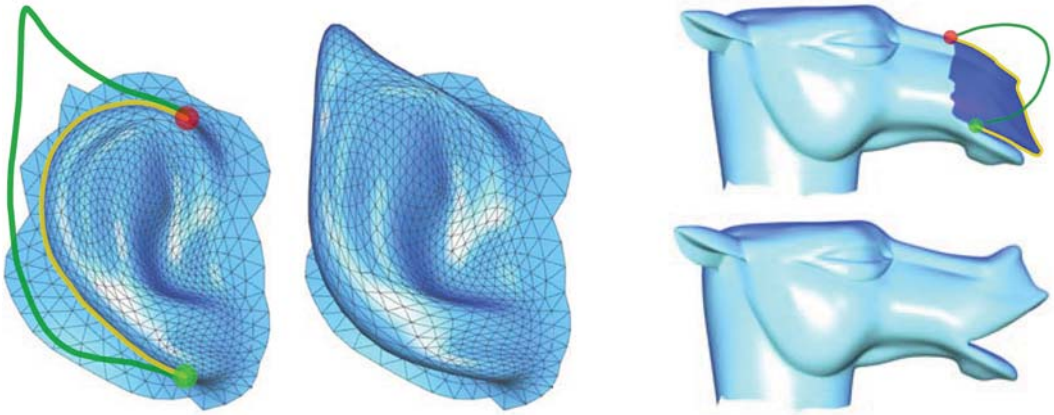

Figure 3.4: Modeling via Silhouette Sketching [67]

can then be used as control handles to modify the shape of the object. Olsen et al. [71] describe a method where the mesh is refined at the positions where additional detail is desired by the user indicated through a sketch. A completely different technique was presented by Pihuit et al. [75] for the modeling of vascular systems. They do not use an interactive process to create the model, but instead a finished drawing which is transformed into a 3D model at once.

Although modeling is the biggest field for the use of sketch-based interfaces, they are also used for other tasks. Davis et al. [21] present a technique which transforms 2D sketches into an animation. The user has to draw a character in the desired key-frame positions to enable the system to calculate the skeleton poses. The result should be regarded as a pre-version which has to be refined in another animation-software package. Another wide field of research is the use of sketch-based interaction to give instructions to the system which can be realized by gestural interfaces. The input stroke has to be preprocessed to recognize the gesture, which then has to be translated into a command. The application must have a database of existing templates enabling the comparison of the current user gesture with the stored gestures in order to find a match which indicates the desired operation. Using gestural interfaces to give instructions can be more intuitive and natural than the traditional way of using menus and buttons. A simple example is the navigation in today's smartphones. With the gesture of wiping over the screen, the action of moving the content is triggered, which is extremely straightforward. Teddy [39]

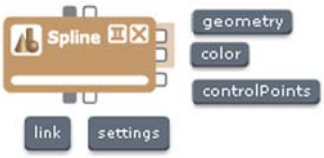

Figure 3.5: Representation of the Spline Node with output and meta-flow connectors.

provides some commands like cutting and erasing executed by a drawn stroke or sketch. An application should give sufficient visual feedback showing what gesture was recognized by the system and which action will be triggered on account of this.

Most traditional applications use 2D strokes, even if they are transformed into 3D commands. Drawing a 3D stroke instead is even more complicated, because every two dimensional point is a representative of infinitely many points in the three dimensional world. The system has to find the appropriate stroke which was intended by the user. Das et al. [20] consider that the 2D input curve is the projection of the 3D curve desired. Thus they propose that the correct 3D stroke should be the back-projection with the smallest curvature. Another approach of defining a stroke in a three dimensional world was introduced by Igarashi et al. [37]. They use existing objects in the scene to project the user input stroke onto their surface, which leads to an unambiguous definition of the curve.

Sketch-based interfaces are an intuitive and fast way of capturing the user's input and commands which makes them suitable for experimental tasks. Using them increases the productivity and supports the user in the decision-making process.

3.2 Modular Interactors

To implement the sketching ability in Visdom, some requirements have to be considered. The sketch-based interface should allow the user to interact with the simulation, for example, by sketching a barrier directly in the 3D view monitor. As mentioned before, Visdom is based on the data-flow concept. Nodes are connected with each other and if a node computes output, it passes it on to the next node. A traditional data-flow

network is set up once and should then perform a specific computation. This leads to the fact, that data only flows downstream from source nodes to sink nodes. To implement the sketching functionality, the user input has to be captured from the 3D view monitor and used by another node in the system. The view node, as sink node, is situated at the end of the data-flow network. The ability to interact with the view node, in order to manipulate the settings of nodes which are further upstream, is required. An example for such a node is the Spline Node (see Figure 3.5). The Spline Node is a node which receives the user input points from the view node and processes them so they can be used to control the simulation as desired by the user. As in the static data-flow graph the 3D view node is situated downstream of the Spline Node, there is no possibility to pass information from the view node to the Spline Node via the traditional data-flow connections. The meta-flow concept has been implemented in Visdom to allow upstream communication [94]. Nodes which are further down the data-flow network are able to update settings of other nodes. The concept enables sending the user input points, which are essential for processing the interaction, from the 3D view node, where they are captured, to an arbitrary node. When a meta-flow rope is established between two nodes, modular interactors, if available, are created automatically. Modular interactors extend the interaction interface of view nodes by providing tools (see Figure 3.6).

A node can make one or more interactors available. The Spline Node provides one interactor called the Spline Interactor. Other examples of interactors are the Camera Interactor and the Selection Interactor which handle camera movement and selection events respectively. An interactor always has a logical relationship to two nodes, which in some cases might be the same node. The first node is the creator node which provides a specific functionality. The second node is the host node which wants to use this functionality interactively. In our case the first one is the Spline Node and the second one is the 3D view node. The Spline Node provides the functionality of manipulating the simulation which can be triggered inside the 3D view node by using so-called tools. An interactor can provide one or more tools to the host node. Tools are represented by icons which can be used to activate a specific functionality of the interactor. The icons are shown in the context of the host node (see Figure 3.7). For example, tools provided by the Camera Interactor are the rotation tool or the move tool. The Spline Interactor provides tools for creating and manipulating the spline.

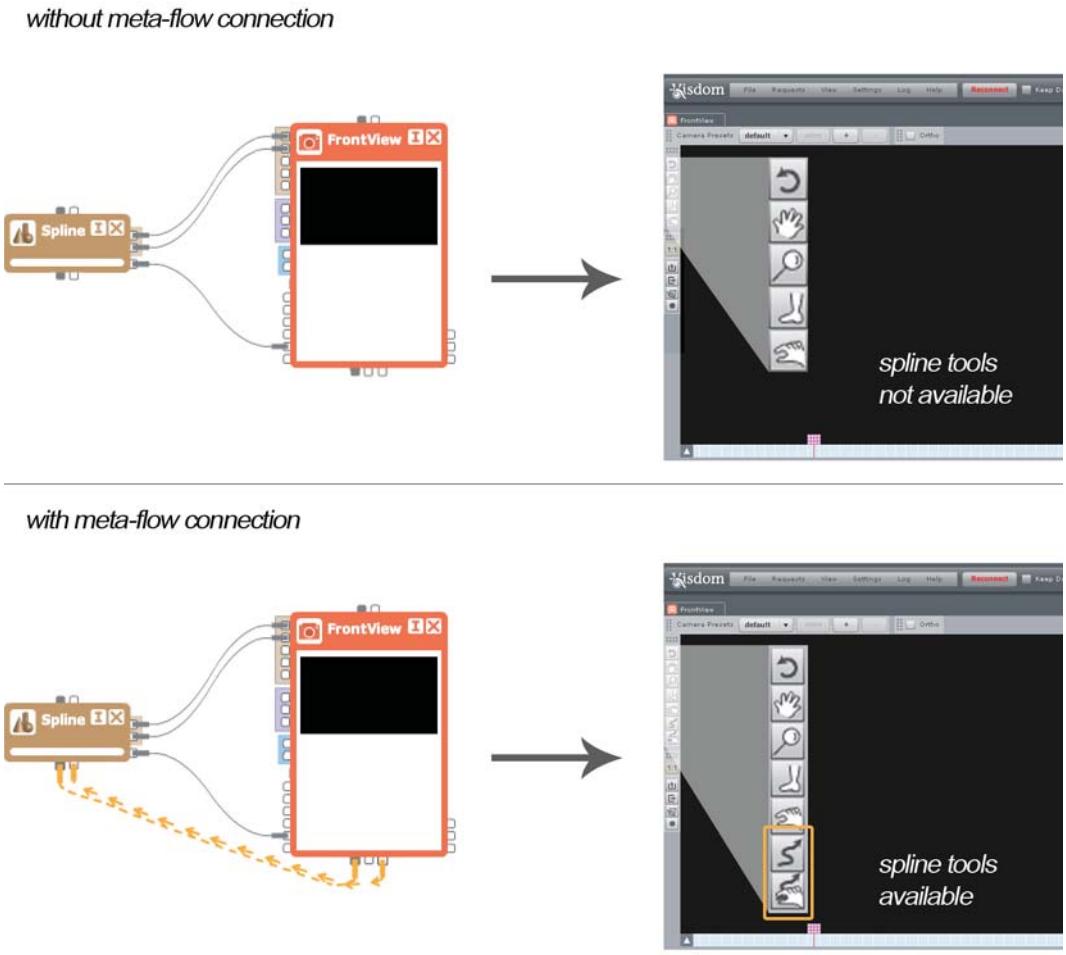

Figure 3.6: Meta-flow connection effectuates creation of interactors (tools) in the 3D monitor

Interactors are implemented on the client-side of the system but the implementation of the nodes' functionality is on the server-side to ensure high performance. This leads to the fact that if the user wants to use a tool, the interactor has to send a method call to the server in order to trigger the computation. Such a request is an XML file with all the information necessary for the server to execute the procedures and to return the desired result. To allow the interactor to communicate with the server we use the linking mechanism provided by Visdom. This mechanism uses the meta-flow between two data-flow nodes to gain an additional communication channel where parameters can be sent and

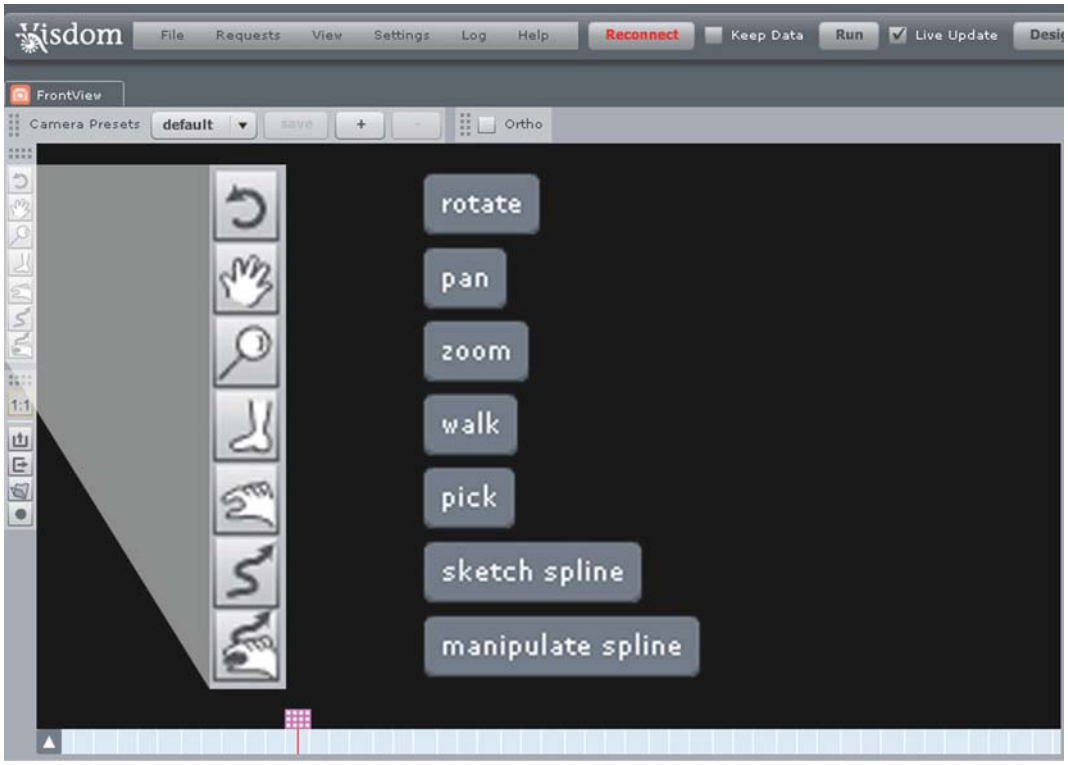

Figure 3.7: Modular Tool Bar in Visdom. Tools are created automatically by interactors when a meta-flow connection is established.

node relations can be defined. The connectors on the top and bottom side of the node allow meta-flow connections between different nodes (see Figure 3.5). When a connection is established, a special linker node is created between the two nodes. With the help of this node, the user can modify the properties of the link. On the meta-flow channel, the communication is not required to follow the traditional source to sink direction of the data-flow. This means that information can be sent to nodes upstream. A node can send its settings object or its so-called results object over the meta-flow. A result is another set of parameters which can be generated by some nodes during execution. The settings or results object is then pushed over the meta-flow to another node where it affects its settings. The source of a meta-flow connection can be either a setting or a result whereas the destination is always a setting. This leads to two different kinds of linkings, settings-to-settings linkings and result-to-settings linkings.

The first type is used for shared settings and executes exclusively on the client-side. If two nodes have a setting which corresponds, a settings-to-settings linking can be established to connect the value of one setting to the value of the other one. If the user changes one of the two settings the system automatically updates the other one. Another example for the use of this linking type is adding additional parameters to some nodes, like adding particle emitters to simulation nodes or light sources to view nodes.

The result-to-settings linking is used to synchronize the client. If a node needs to update the settings of another node, the first one can use the linking to send a result object in order to change the settings of the second node. Everytime a result is calculated by a node and pushed over the result-to-settings linking connection, a run request is sent to the server. This leads to a data-flow execution starting from the node whose settings where changed.

Modular interactors use the communication channel of the result-to-settings linking. This concept ensures that every change at the source is reflected immediately in the destination which fits the task of interaction perfectly. Every interactor, or more precisely every tool, has a different type of request which is sent to the server. Examples are the rotate request or the pick request of the Camera Interactor. These contain all the information the server needs to execute the correct methods. For different types of requests the server produces different types of results, so the rotate request for example leads to a camera change result.

The linking system is based on the publish-subscribe model [4]. Every node can specify which settings or results it wants to publish. These objects are then categorized as publishables and are available to other nodes. In order to gain access to the setting or the result, other nodes have to listen for changes of the publishables. This can be achieved by a subscription to a specific entry. The desired publishables and subscriptions can be defined in the linking settings of each node.

For the purpose of introducing a sketch-based interface into Visdom, the Spline Node was implemented. The node is responsible for the processing of the user input to enable the manipulation of the simulation. The Spline Node provides one interactor, called

Spline Interactor, which can be used in all semantic windows with the category monitor3D, which corresponds to a 3D view node. When the view node activates the interactor, it receives the ability to use the tools. The Spline Interactor provides tools for sketching a spline and for manipulating an existing one. When the user clicks on the 3D monitor, the mouse event callbacks are triggered and a request to the server is sent. The content of this request depends on the selected tool and on the actual event. For example, with the sketch tool selected, the mouse down event signalizes the start whereas the mouse up event signalizes the end of a stroke. As the mouse moves over the screen, events are triggered which send requests to capture the current points. The linking settings of the Spline Node are rather simple. The most important is the subscription for getting the points which are published by the 3D view node. The mouse events always provide the location of the mouse pointer in screen space, which is a 2D coordinate. To use the points in a meaningful way, the Spline Node needs the 3D coordinates in world space. This is the reason why the 2D points have to be preprocessed before they are useful to the Spline Node. The subscription actually listens for these transformed points and not the original 2D user input points. The transformation takes place in the 3D view node and is handled by the Dragger classes which will be explained in detail in the next chapter. It is important that the Spline Node receives the points transformed correctly into world space coordinates.

Figure 3.8 illustrates the communication flow between the view node and the spline node according to the client-server based architecture of Visdom (see Figure 3.8). The lower half of the figure shows the Spline Node and the 3D view node on the client-side. Above are the representatives of the nodes on the server-side which encapsulate the functionality. The first step is an information request, which contains the 2D mouse points, from the view node on the client to the server (1). This event is triggered after the user has clicked into the 3D scene with a spline tool activated. The server-side implementation of the view node receives the 2D points from the client and transforms them into points in world space. Thereafter, the view node sends the interaction information to the server-side representative of the Spline Node (2). This data transfer is an upstream communication and thus has to be realized using the meta-flow. The view node publishes the transformed points as a result for the purpose of making them available to the Spline Node. The linking mechanism causes the modification of the result

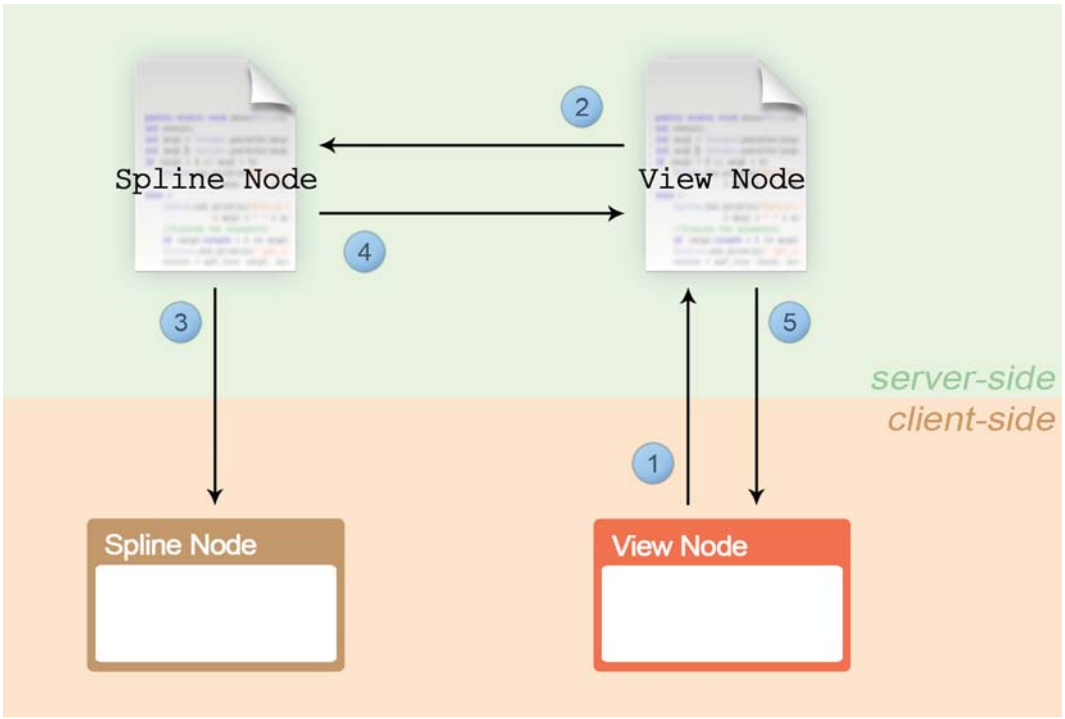

Figure 3.8: Communication flow between server and client while sketching. (1) Information request from client to server. (2) View Node sends interaction information to the Spline Node via meta-flow ropes. (3) Synchronization of client settings with server settings. (4) Data-flow execution. (5) The View Node sends the final image to the client.

to be reflected in the settings object of the destination node. The change of the settings triggers a callback function of the Spline Node to handle the modification and update itself. Next, the client settings need to be synchronized with the server settings. For this reason, the server sends a result object with all the important settings information to the client, which allows for the synchronization of the client-side Spline Node (3). On the server, the Spline Node computes the new output according to the current settings. This update leads to an execution of the data-flow (4). The Spline Node passes its output to all connected nodes which have to be updated as well. The data flows down the network, and after all affected nodes have computed their outputs it finally reaches the 3D view node. According to the input data and its settings, the view node renders the final image which is then transmitted to the client (5). The user receives a visual feedback of the sketch after the 3D monitor is updated.

Modular Interactors allow for the implementation of an interactive feedback cycle in the context of a data-flow based system. Interactivity is essential when handling user inputs and is fundamental for a sketch-based interface. Using the linking mechanism of Visdom, the Spline Interactor is able to ensure that the Spline Node receives all user-input data immediately. The aim of the sketch-based interface is to allow the user to manipulate the simulation via drawing strokes directly into the 3D monitor. As mentioned before, the user input points cannot be used directly to interpret the sketch. Thus, the Spline Node has to prepare the data for later use, and provide sufficient visual feedback and manipulation options to the user.

3.3 Spline Creation and Manipulation

The most important factor of a sketch-based interface is the user input data. The system has to be able to handle and interpret it correctly. Sketched user input is never noiseless. Noise can be induced by the user himself or by devices used. An appropriate system has to reduce the noise and create a clean stroke which corresponds to the intention of the user. The system also has to provide visual feedback to the user in order to make the effect of the sketch visible. Furthermore, it should be possible to manipulate existing strokes quickly and easily. Every component of the system has to be intuitive and efficient to support the user in the solution-finding process.

The whole sketch-based user-interface functionality is split up among different nodes. Figure 3.9 shows the sketching pipeline and the nodes responsible for each step.

The first task is the acquisition of the sketch. As mentioned before, the Spline Node receives the transformed user input points from the 3D view node and saves them into a list in the settings object. The computation of the new points takes place in a component of the view node called the Dragger. This component is responsible for handling and transforming the points which are created when the user moves the mouse pointer over the screen. The task of finding a point in world space which corresponds to a point in screen space is not trivial. A way to resolve the existing ambiguity is to project the 2D point onto a defined plane or an existing plane in the scene. This approach is fast and easy to implement, and suited perfectly to the Dragger. A precarious issue is the choice

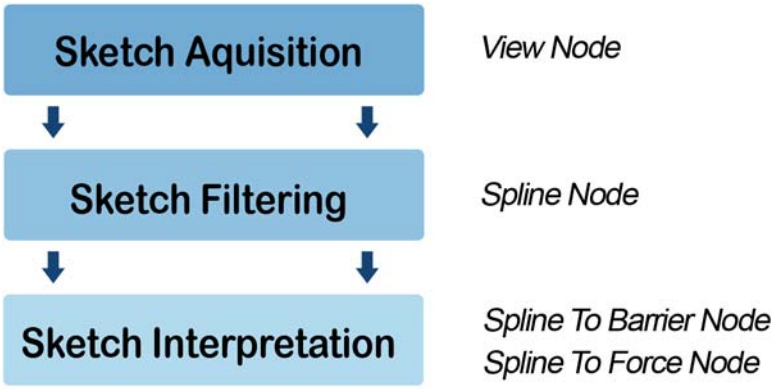

Figure 3.9: Sketching pipeline with responsible nodes

of the plane. In order to define a plane two parameters have to be given. First, a point which should lay on the plane and second, a vector which should be perpendicular to the plane. The plane is defined using the first user input point and the scene's up vector. As a result, the plane is parallel to the floor of the scene. To calculate the 3D coordinate for the first point the system uses the depth buffer information. Reading the buffer is a lot slower than projecting a 2D point onto a plane but necessary for obtaining the first point. The world space coordinates of all following points can then be calculated quickly using the screen-space position and camera information. With this implementation, the points can only be dragged on the same height which can be problematic if the terrain has many bumps or valleys. In our scenario the aim of the user is to build up barrier arrangements of sandbags or mobile protection walls. These are often established on flat regions because of stability reasons. Additionally, the sandbags are not positioned directly but dropped by a virtual helicopter according to the real-world scenario in New Orleans 2005 [15]. The sketch determines the desired dropping target positions. Using this procedure, the barrier will adapt to the terrain automatically according to the physical simulation.

After the sketch acquisition, the Spline Node holds the points in world space coordinates in its settings object, and can proceed with the next step, the sketch filtering. The mouse event of the 3D monitor triggers every time when the mouse changes its position. Therefore the system captures plenty of points and provides them all to the Spline Node. There are too many to be used efficiently, and they also contain noise induced

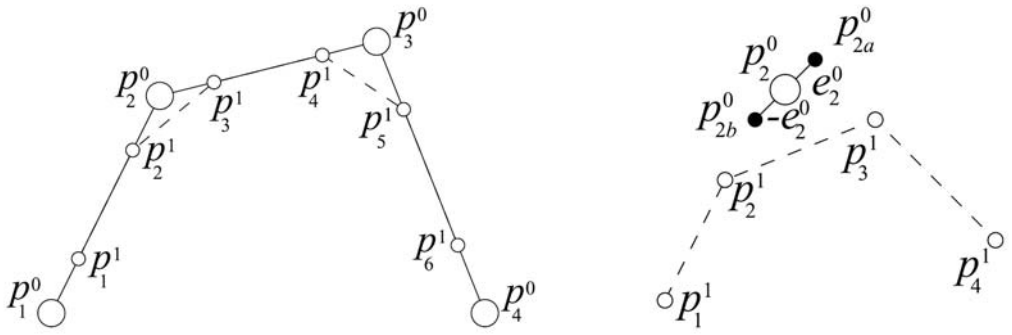

Figure 3.10: Chaikin's scheme (left) and reverse Chaikin scheme (right) [33]

by the user and the device. Furthermore, the points are distributed irregularly along the stroke because their density depends on the drawing speed of the user. The filter should create a smaller and better distributed set of points. It should remove the noise without destroying the features which are intended by the user. To achieve this, the Spline Node implements the functionality of translating the set of points to a curve represented by control points. The advantage is that only a few control points are necessary to represent a complex curve. With this parametric representation, the stroke can later be evaluated on every arbitrary point as densely as desired.

To realize the required filtering, the reverse Chaikin algorithm presented by Hassan et al. [33] is implemented by the Spline Node. This method is a reverse subdivision algorithm which means that it calculates a coarse representation of a given curve. If the Chaikin subdivision scheme is applied to the result, the original curve is reconstructed almost perfectly. This approach is not as accurate as other curve fitting algorithms but it is much faster which is an essential characteristic for our purpose.

Figure 3.10 shows Chaikin's scheme on the left and the reverse Chaikin scheme on the right. The large circles on the left side of the figure are points of an arbitrary polygon. The set of small circles is the result of one subdivision step using the Chaikin scheme. On the right side of the figure the small circles are the points of the original polygon and the large circle is the result of one reverse subdivision step calculated with the help of the two black circles. These two points are candidates for the result. The actual re-

sult is calculated by averaging the candidates, which introduces an error. If the original polygon was created by the Chaikin scheme, the two points would be at the same position which reduces the error to zero. Cherlin et al. [16] filter the input stroke of their sketch-based modeling system by fitting a B-Spline curve to the set of points using the reverse Chaikin algorithm. This is possible because the method is based on a quadratic B-Spline where the coarse representation of the original curve can be interpreted as the control points of the B-Spline. Every time the reverse Chaikin subdivision is applied, the number of points is decreased and the curve becomes smoother which reduces the noise. But every application also increases the error value which represents the distance to the original curve. It is important to find an acceptable number of iterations for sufficient denoising without losing too much information about the original points. Cherlin et al. point out that three subdivision passes are satisfactory for their purpose.

For one point of the coarse representation, the reverse Chaikin algorithm interpolates four points of the original polyline. Following this idea, the information about the first points is lost. The same problem can be observed at the end of the stroke. After every iteration of the reverse Chaikin algorithm, the stroke loses some points at the start and the end which leads to the effect that the curve shortens and changes its general appearance. This problem is not relevant if the method is applied on a closed curve. However, for a sketch-based interface where the strokes are usually not closed, this behavior is unacceptable. Especially for the purpose of sketching a barrier, where the first and the last points of the user input are the most important ones. They determine where the user wants the barrier to start and to end, and this information has to be preserved. The Spline Node implements an adapted version of the reverse Chaikin algorithm to preserve the first and the last point of the stroke by adding them to the resulting curve. Due to this modification, it is no longer possible to obtain the original curve by applying the traditional Chaikin scheme to the result of the reverse subdivision step. The Spline Node provides the ability to change the number of filter iterations as desired. The user can choose a specific filter level. The system converts the spline which was previously calculated with an arbitrary filter level, to the spline, calculated with the new filter level. If the user wants to move from a higher filter level to a lower one, the Chaikin algorithm has to be applied. As mentioned before, the traditional scheme cannot be used if the previous spline was calculated with the adapted reverse Chaikin method. A solution

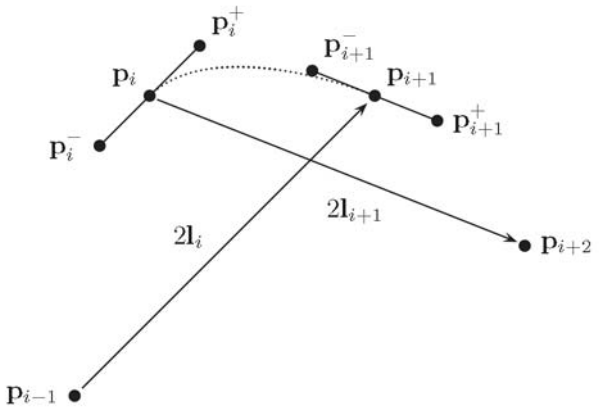

Figure 3.11: Construction of one curve segment with the Catmull-Rom method [12]

is to use an adapted version of the Chaikin scheme which ignores the first and the last point of the original polyline for the calculation. Another possibility is to always use the reverse subdivision method on the original user input. Since the points are stored in the settings object of the Spline Node, the filter process can use them as long as they are available to achieve the desired filter level.

Due to the adaption of the reverse Chaikin subdivision scheme, the resulting curve no longer has the characteristics of a B-Spline. Therefore, another representation for the user input points has to be found. The idea is to combine a set of cubic Bézier curves to construct a so-called piecewise Bézier curve [12]. This representation allows for an easy handling of a long and complicated stroke without the complexity of a higher degree curve. Every pair of filtered points is defined as the start and the end point of a distinct degree-three Bézier curve. A Bézier curve of degree three has four control points which determine its appearance. The first and the last point lie on the curve, but the second and the third do not. If the set of filtered points has an odd number of points, an additional point is added for the calculation. This definition ensures that the curve is continuous and that it interpolates the filtered points. Thus, they all lie on the final curve, which is important because they are used as the control points of the spline. The user should have the ability to manipulate the spline by moving parts of it directly instead of trying to alter the curve with control points lying in its vicinity.

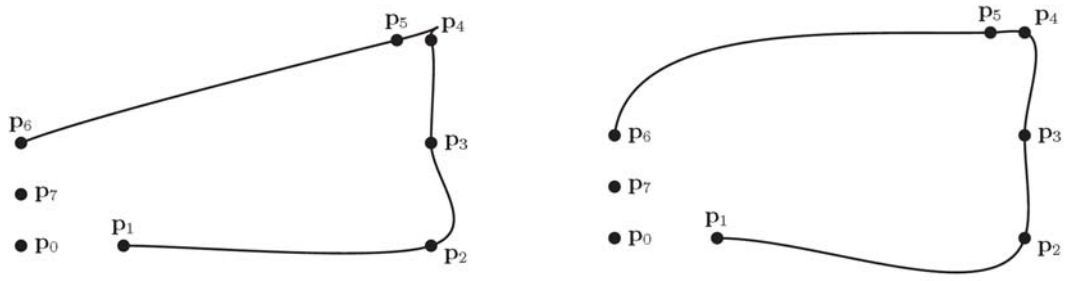

Figure 3.12: Comparison of the Catmull-Rom (left) and the Bessel-Overhauser (right) method for the interpolation of non-evenly spaced point sets [33].

There are several ways to define interpolating piecewise Bézier curves. A common approach is using Catmull-Rom splines which is described by Buss [12]. The result of the Catmull-Rom method is a piecewise Bézier curve that interpolates all the points of the original set except the first and the last point. For this reason, these two points have to be duplicated in order to be taken into account. Figure 3.11 shows the construction of one segment of the Catmull-Rom splines. The start and the end of each segment is given by two successive points p_i and p_{i+1} of the filtered set. To define a Bézier curve of degree three, two additional control points p_i^+ and p_{i+1}^- are needed. These two can be calculated by constructing the tangents for the start and end point. For point p_i the tangent is parallel to the line $2l_i$ which connects the previous (p_{i-1}) and the next point (p_{i+1}) in the filtered set. The additional control points are defined as:

$$p_i^+ = p_i + \frac{1}{3} l_i \tag{3.1}$$

and

$$p_i^- = p_i - \frac{1}{3} l_i \tag{3.2}$$

where

$$l_i = \frac{1}{2}(p_{i+1} - p_{i-1}) \tag{3.3}$$

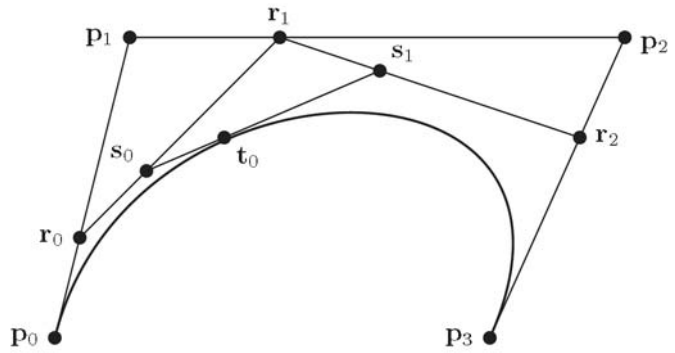

Figure 3.13: Casteljau algorithm: evaluation of the curve point for the parametrization value 0.33 [33]

The Catmull-Rom interpolation type is typically used for point sets which are already more or less evenly spaced. If this is not the case, a bad overshoot can occur, especially in situations where two close control points are next to widely separated ones. The user-input data is not necessarily evenly spaced due to the varying drawing speed of the user. Thus, the Catmull-Rom interpolation might cause problems in special cases. The Spline Node implements an alternative, suggested by Buss [12], which is a generalization of the Catmull-Rom method. The so-called Bessel-Overhauser splines approach uses chord-length parameterization to modify the calculation of the additional control points. In Figure 3.12, the comparison between the Catmull-Rom splines and the Bessel-Overhauser splines is illustrated. The region around point p_4 and p_5 shows the critical segment where the Catmull-Rom method induces an overshoot of the curve. The Bessel-Overhauser method is fast and easy, making it unnecessary to store the additional control points as they can be calculated in real-time.

For the construction of the whole stroke, all Bézier curves have to be evaluated at a single sample rate. To realize this, the Spline Node implements the commonly used de Casteljau's method [12]. This method is much simpler and more stable than using formulas of the Bézier curves. Figure 3.13 illustrates the evaluation of the curve.

After the filtering and with the ability to evaluate the curve on any point, the sketch is ready for the interpretation step. How the system translates the spline to user com-

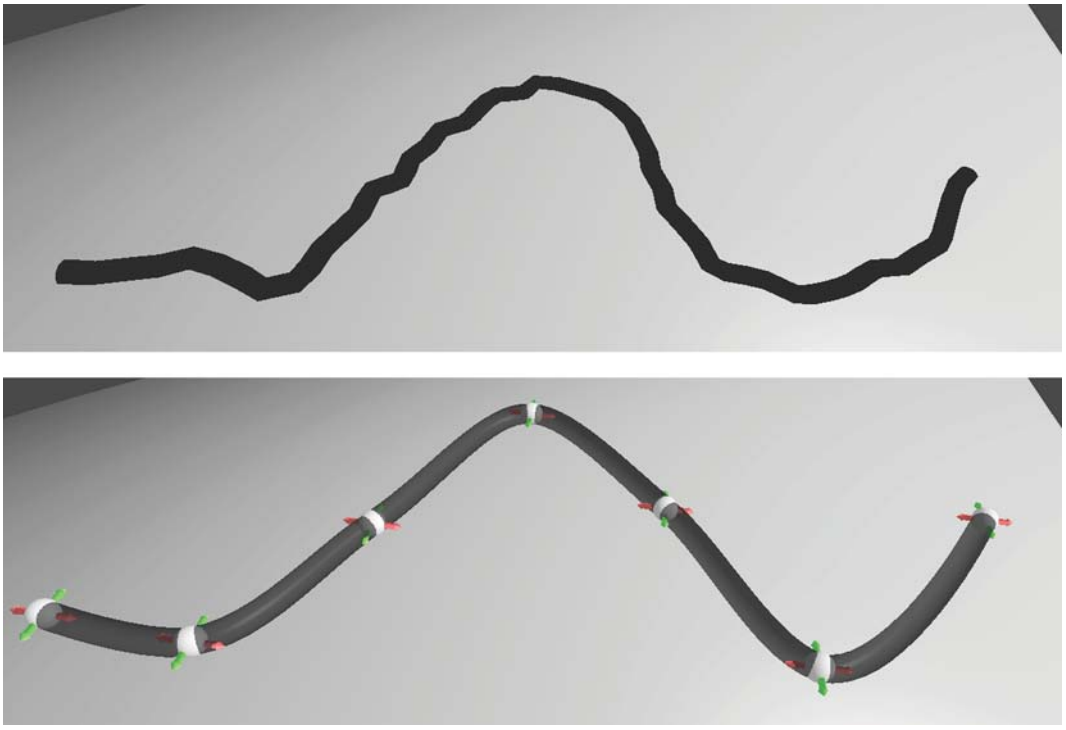

Figure 3.14: Unfiltered preview version (top) and filtered version (bottom) of the spline

mands for manipulating the simulation will be explained in the next chapter.

Another very important issue is the feedback visualization. For an interactive steering system a sufficient feedback concept is essential. It is important that the user receives information about decisions made and their effect on the system. The feedback visualization should support the user to be faster in the decision making process, especially in time-critical situations.

First of all the user's stroke has to be visualized. In order to realize this, the Spline Node constructs a tubular mesh around the calculated curve points. This mesh can be transmitted to the 3D view node in order to be rendered into the scene. The user can modify the appearance of the Spline by changing the settings object. It provides options to change the color, the thickness or the mesh's level of detail. The Spline Node implements an optimization during the spline creation. When the user clicks on the 3D monitor and starts to sketch, the spline is visualized using a preview version. For this visualization,

the sketch is not filtered. The Spline Node skips the reverse Chaikin step and calculates the curve points and the mesh directly from the user input points. This is done to achieve high performance during the sketching process and to provide instant feedback. When the user releases the mouse button in order to finish his stroke, the Spline Node executes the complete filtering process to obtain an optimized and smooth spline. Figure 3.14 shows the difference between the preview version and the final version of the spline.

Fine Tuning

For the final version, the Spline Node provides a mesh which represents the filtered points in order to visualize them in the 3D monitor. An interactive sketch-based interface is insufficient without the ability to change the stroke. The user should be able to sketch an idea, to analyze the visual feedback and then to modify the stroke until the desired configuration is reached. Therefore, the Spline Node provides the functionality needed to manipulate the filtered points. As mentioned before, the filtered points are defined as start and end points of Bézier splines. Since the whole sketch is composed of multiple Bézier splines whose additional control points are calculated interactively, manipulating the filtered points provides full control over the stroke. Thus the filtered points can be interpreted as the control points of the user's sketch.

The control points are visualized as 3D widgets directly on the spline. If the user clicks on a control point, the change of color confirms the selection (see Figure 3.15). Additionally, the Spline Node provides a visual guide which is associated with the implemented manipulation functionality. Moving a point in a 3D scene can sometimes be tricky because of the three degrees of freedom. The user has to communicate in which direction the point should be translated. To provide an easy and unambiguous way of manipulating the control points, the Spline Node implements the concept of view-dependent interaction. The basic idea of this approach is to constrain the degrees of freedom depending on the view direction. If the user looks onto the spline from above, the control points can only be manipulated along a plane which is spanned by the x and y axes. This functionality is implemented by the component called 2D Dragger. When the user moves the camera and views the spline from the side, the constraint changes and the manipulation is only allowed in the z direction, which is the up direction of the

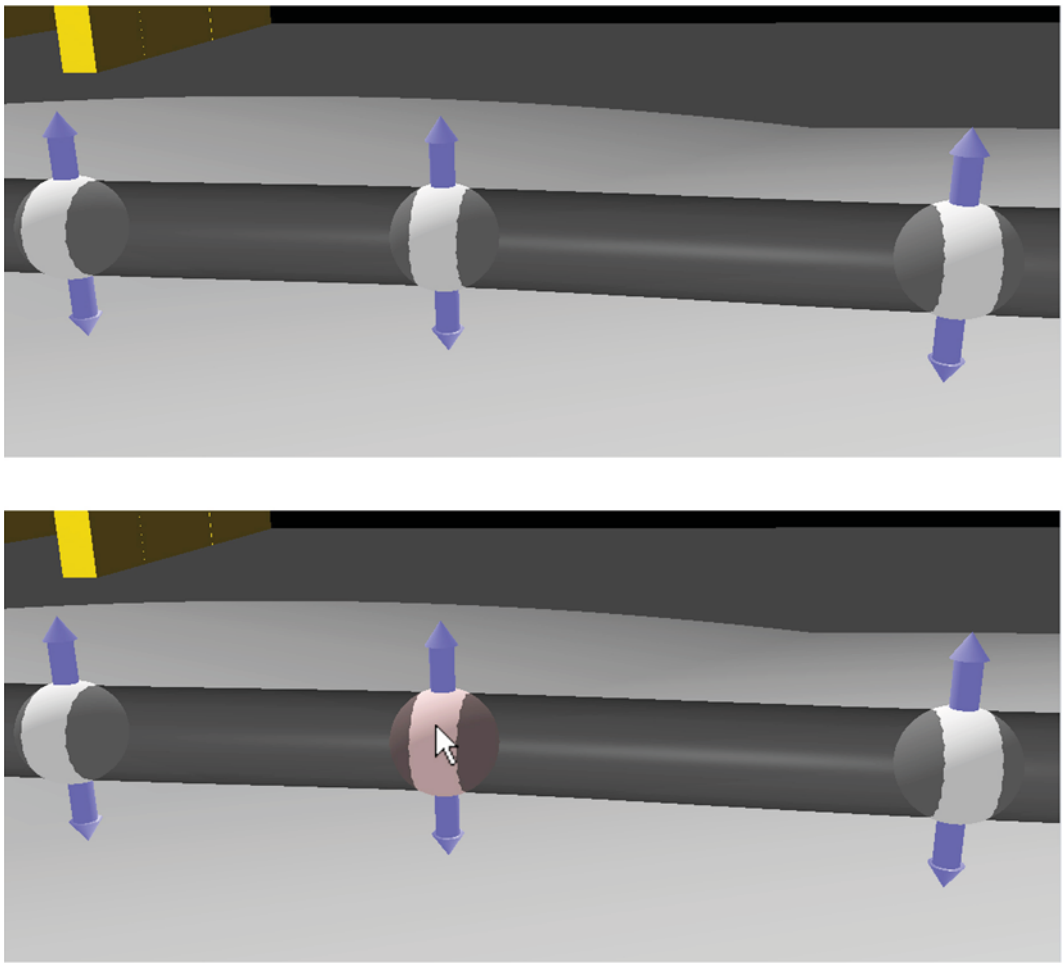

Figure 3.15: Standard control point visualization (top) and selected control point (bottom)

scene. In this case, the 1D Dragger is responsible for computing the new positions. This approach follows the idea that the user first changes the camera to achieve an appropriate view before the control point is manipulated. It is unlikely that the user changes to the top view if the point has to be translated in the z direction. It is more likely that the camera is positioned according to the desired manipulation direction. Visual guides are used to provide feedback which constraint is currently activated (see Figure 3.16). The view-dependent manipulation approach provides a clean and intuitive interface for modifying control points. It provides assistance for the task without complicating the workflow additionally. The user does not have to change the manipulation direction

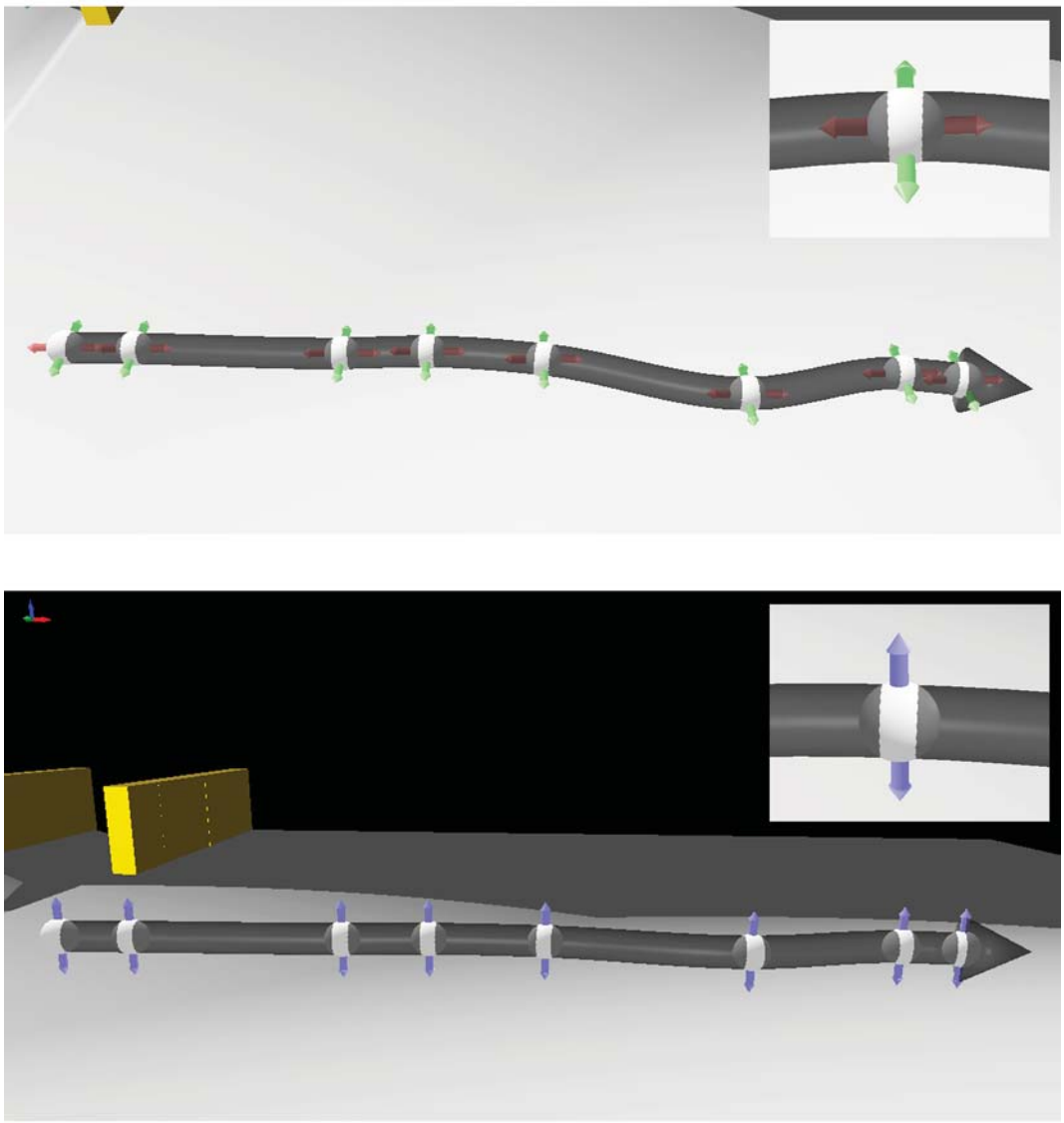

Figure 3.16: View Dependent Manipulation: horizontal (top) and vertical (bottom) manipulation constraints and visual guides

manually which increases the productivity and allows for fast interaction.

Chapter Summary

The Spline Node provides comprehensive functionality to visualize and manipulate the user-input sketch. For the visualization, a spline is displayed in the 3D scene, which can be manipulated afterwards using the control points and a view-dependent approach. The system supports a fast and intuitive interaction cycle by providing sufficient feedback and the ability to modify the stroke. This allows a fast decision making process for emergency situations.

The next task is to use the stroke for influencing the simulation. This can be achieved by sketching barriers or forces into the scene in order to change boundary conditions which affect the SPH simulation. A mapping which translates the spline to commands modifying the conditions has to be found. For this issue, visual feedback is again essential to understand the effect of decisions made. The appropriate mapping and visualization is discussed in the next chapter.

CHAPTER 4

Translation to Boundary Conditions

In sketch-based systems, one of the most challenging tasks is the interpretation of the sketch, which is the last step of the pipeline. The user draws a stroke with the intention to construct something, to modify an existing object, or to trigger a specific command. The system should be able to assign a specific meaning to the stroke. It has to understand the intent of the user and execute the appropriate operations to accomplish the desired result.

Our sketch-based system is used within a simulation-steering application which has the aim to analyze the impact of floods and similar catastrophies. For such applications, it is necessary that the user is able to establish barrier arrangements and test their stability. Furthermore, it should be possible to change the flow velocity and the flow direction. This provides the ability to test different configurations in order to cope with the unpredictability of the flood. To realize these two functionalities, the simulation settings have to be modifiable. The user input, which is already represented as a spline, has to be translated into a modification of the simulation's boundary conditions. Finding a mapping from the spline to commands which modify the conditions allows for the implementation of a fast and intuitive way of changing simulation parameters.

In order to realize this, additional nodes are integrated into the present system. These nodes implement the functionality needed to map the spline to desired parameters. The

first new node is responsible for defining barriers and is called Spline To Barrier Node. The second one, called Spline To Force Node, allows for the direct manipulation of the simulation to change flow parameters. Both types of nodes have to be connected to the output connector of the Spline Node which provides the spline. To provide the result of its computation to other nodes in the data-flow network, the Spline Node defines different outputs. The geometry output is needed to offer the mesh data in order to enable visualizing the spline in a 3D monitor. The other output connector supplies the calculated control points of the spline. Since these points represent the whole spline and contain all information needed to reconstruct it, they are everything that is needed by nodes which have to use the spline for further calculations. Thus both nodes, the Spline To Barrier Node and the Spline To Force Node, have an input connector which expects the control points of the spline. The output computation of these two nodes is the topic of the next two sections.

4.1 Barriers

The most important task during the analysis of countermeasures against the expansion of a flood is testing different barrier arrangements. The user should be able to establish barriers quickly in order to start analyzing and modifying them until the desired result is achieved. The existing barrier-creation workflow is time-consuming. Barriers have to be created in the Bag Model Designer Node which provides an additional 2D interactive component in the semantic view of Visdom. By clicking onto this view, the user is able to place single sandbags (see Figure 4.1). Thus creating a whole barrier arrangement is extremely time-consuming. The sketch-based interface increases the productivity of this process tremendously to provide support for decision makers in time-critical situations. The user should have the ability to draw barriers directly into the 3D scene and modify them afterwards if desired.

The Spline To Barrier Node is introduced to integrate the sketch-based interface into the barrier-creation pipeline. This node is responsible for translating the spline into a barrier. The mapping between the spline and the barrier has to be transparent and clear to the user to enable intuitive usage. Using the spline should feel like modifying the barrier directly. It should be possible to manipulate the spline fast and efficiently without

Figure 4.1: Workflow of the existing sandbag arrangement tool. (1) An interactive component is available to create the barriers which shows the scenario from the top. (2) Via a context menu, triggered with a right mouse-button click, a single bag can be created. (3) A sandbag icon is used to visualize the bag. (4-6) The barrier is established by creating multiple bags using the context menu.

struggling with the mapping itself. The node should use the input spline to determine the positions of multiple sandbags simultaneously. Furthermore, it has to arrange the bags and visualize them as a preview in order to provide visual feedback. If the user is unsatisfied with the result, the node should provide the ability to modify the barriers by manipulating the spline.

First, the Spline To Barrier Node fetches the control points from the input connector. In order to be able to determine the correct positions for the sandbags, the curve has to be evaluated. Therefore, the additional control points of the Bézier curves are calculated in real time. The method for the computation is the same as used in the Spline Node.

The main task of the Spline To Barrier Node is the placement of the sandbags. To achieve real-time performance during the simulation, the sandbags are represented by boxes. The size of the boxes can be modified by using the object-dimension parameters in the settings of the Spline To Barrier Node. All sandbags have to lay on the spline which represents the position of the barrier as desired by the user. In order to realize this, the Spline To Barrier Node evaluates the spline, starting from the tail. After placing a sandbag at this start position, the node evaluates the next point on the curve. This new potential sandbag position cannot be used without considering the previous sandbag. The node first has to calculate the distance between the new evaluated position and the previous position. If this distance is not big enough for a new sandbag to be placed, the current position is skipped and a new one has to be calculated. This procedure has to be repeated until an appropriate position for placing the sandbag is found. The aim of this procedure is to avoid the sandbags intersecting with each other which would lead to undesired behavior during the dropping. The step size for evaluating the curve has to be chosen carefully. A too large value can lead to undesired big gaps between the boxes. Choosing a too small step size means that the evaluation has to be executed often, which can cause a decrease in performance. The distance calculation for placing the sandbags depends on their arrangement. The Spline To Barrier Node provides the ability to choose how the sandbags should be oriented in order to allow for the testing of different strategies. Two different modi are available which are compared in Figure 4.2. The top of the figure shows the first option, where the bags are aligned to the axes of the global coordinate system. The calculation of the distances for this mode is simple be-

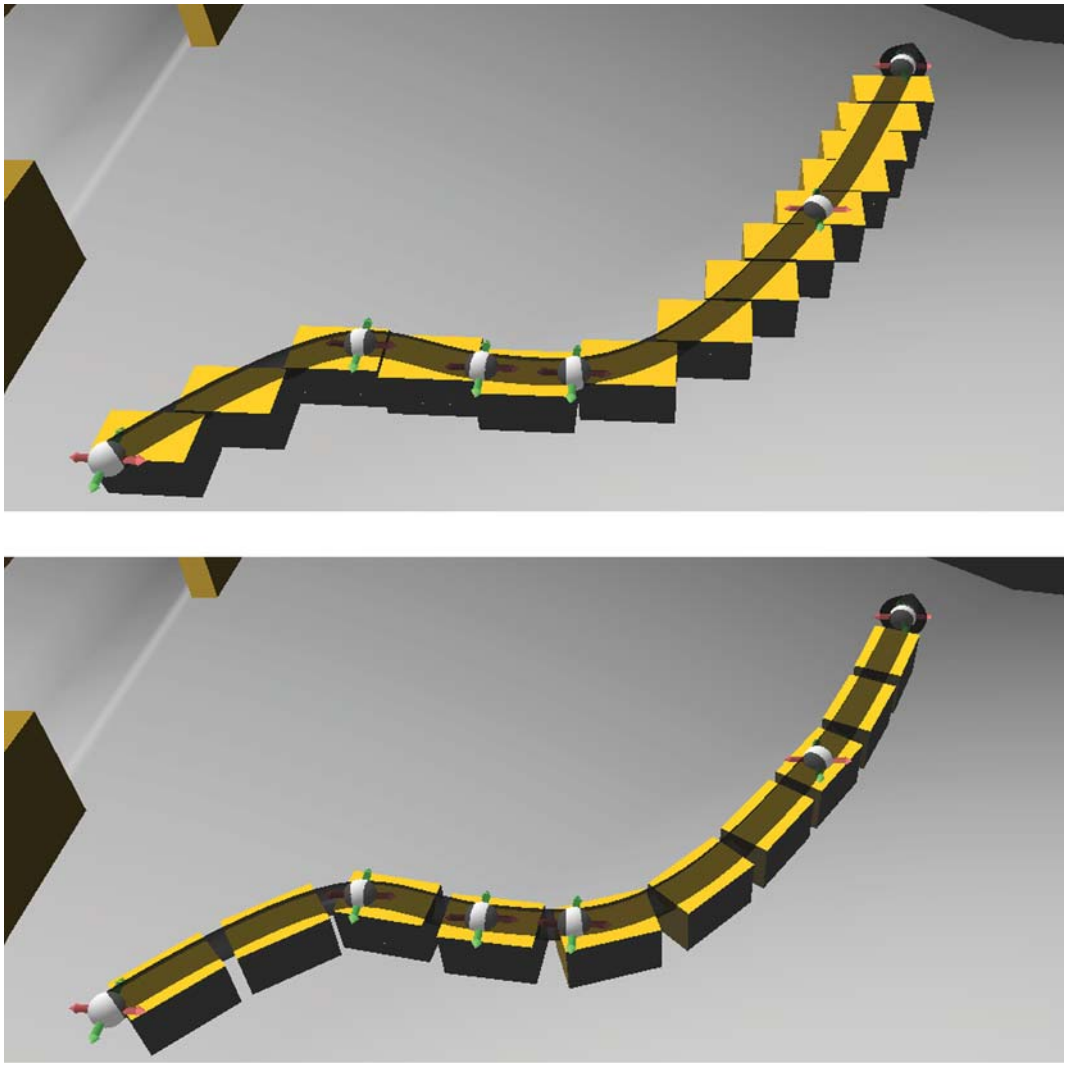

Figure 4.2: Alignment Modi: axis aligned (top) and spline aligned (bottom)

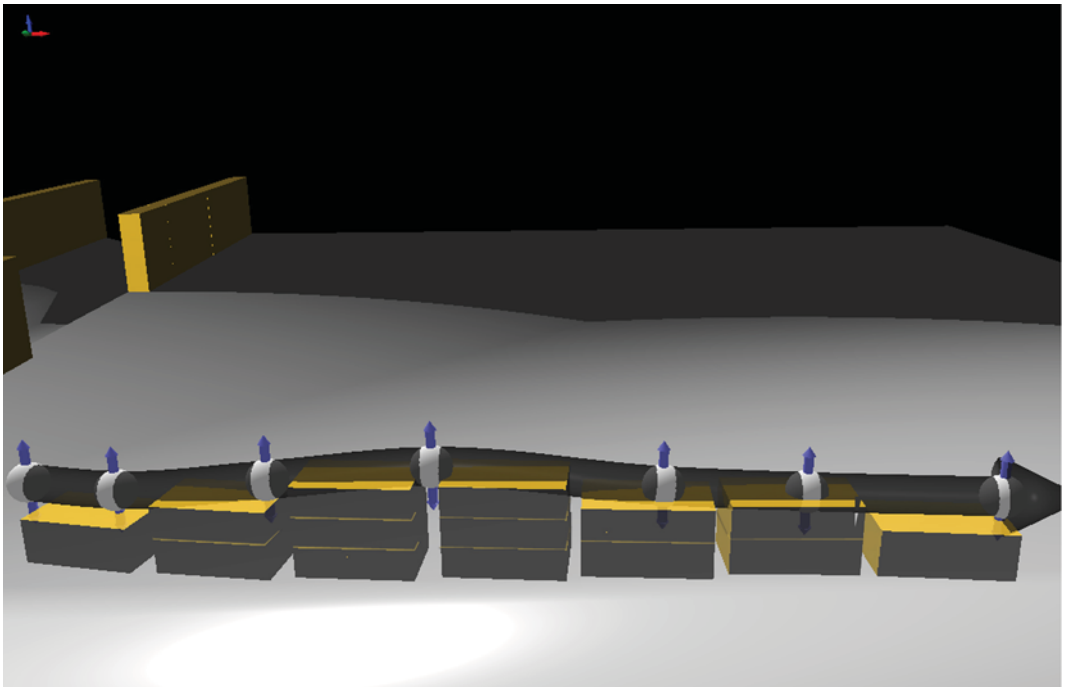

Figure 4.3: Deformed spline to indicate that the sandbags have to be stacked in order to create higher barriers

cause it considers only the width of the boxes. In the other option, shown on the bottom, the sandbags are oriented along the spline. To achieve this, the local x axes of the bags are aligned with the draw direction of the user's sketch. For this mode, the calculation of the distances has to consider the diagonal length of the bags which can be calculated at interactive rates.

A barrier is constructed by placing multiple sandbags on top of each other. Some areas need more bags than others, depending on different circumstances. The user has to place the sandbags according to the behavior of the flood and the resources available. The number of sandbags which should be stacked at a specific point can be determined by changing the height of the spline. After the spline is drawn, it gets lifted by default to obtain at least one layer of sandbags. Employing the control points, the user can modify the height of the spline and stack the bags as desired (see Figure 4.3).

The sandbags are visualized as boxes in the 3D monitor to provide visual feedback. In

the application, the user does not place the boxes directly, but only determines the desired target position. According to the real-world procedure, the sandbags are dropped from a specific height by virtual helicopters. Therefore, the actual positions are determined by the simulation system. Rendering the boxes with transparency is used to indicate the preview mode (see Figure 4.3). After the simulation starts, the visualization changes from preview to live mode, where the sandbags are dropped one after another from the defined height. In this mode the boxes are rendered fully opaque.

The Spline To Barrier Node has two outputs. The first output provides a mesh which represents the geometry of the barrier. This data can be used by view nodes in order to visualize the preview of the arrangement as sketched by the user. Since this data is sent directly to the view node and therefore is not affected by the simulation, it can only be used as a preview.

The second output is responsible for making the barrier available to the simulation node. Since Visdom is implemented in a highly modular manner, many different simulation systems can be integrated. Therefore, an interface has to be defined which enables the communication between the simulation node and other nodes. The Spline To Barrier Node has to provide the information about the positions and the orientations of the barrier's components. The components, which in the standard case are boxes representing sandbags, have to be lifted to the dropping height defined by the system. Furthermore, the dimensions of the bags have to be supplied as well as the weight of each one in order to perform collision detection and calculate physical reactions correctly.

The current simulation system of Visdom is implemented using PhysX [18]. The system is responsible for simulating the fluid representing the flood. In order to create barriers which are able to influence the water, physical representatives which serve as colliders, called actors, have to be created. PhysX provides different types of objects which can be used to represent actors. Most of them are simple geometrical objects like spheres, which provide better performance. For our purpose, the box shape fits perfectly as a representative of the sandbags. PhysX uses the actors to calculate the rigid body interaction between them and the fluid.

The physical simulation depends on the initial states and the boundary conditions of all components. For creating the actors which should represent the sandbags, the initial values have to be determined. These are the position, the mass and the forces acting on the object. The Spline To Barrier Node is responsible for the calculation and the supply of these values. The node computes the position and the orientation of each sandbag and stores them as transformation matrices. A vector of such matrices can then be transmitted to the simulation node and can be used to determine the initial values. Compared to the preview mode, the boxes have to be lifted to the dropping height to be simulated correctly. The Spline To Barrier Node also has to provide the dimensions of the boxes and their weight. The only force which acts on the boxes right from the start is the gravity. This value is provided by the simulation system itself and can be modified using the settings object of the simulation node.

After establishing the initial state of the actors representing the barrier, the simulation can be executed for every frame. The system calculates the new positions of the actors. These values are supplied as matrices and can be accessed by other nodes in order to visualize the result.

The 3D view node uses a renderable called clone renderer to render the boxes of the barrier in real time. The principle of renderables will be explained in the next section. The clone renderer takes a vector of matrices and a geometry mesh as input. Every matrix defines the position and the orientation of one instance of the input geometry.

Instead of using the standard type of barriers which consists of sandbags the user can choose to use other objects. This way, mobile protection walls can be built-up which are another important countermeasure against floods (see Figure 4.4). The user can generate or load any geometry and send it to the input connection of the Spline To Barrier Node. The node calculates the bounding box of the geometry and uses this mesh as the collision object in the simulation.

Using the introduced workflow, the user can analyze the stability and efficiency of different barrier arrangements which can be helpful while testing alternative solutions. For a comprehensive analysis, other influences have to be considered as well, for example,

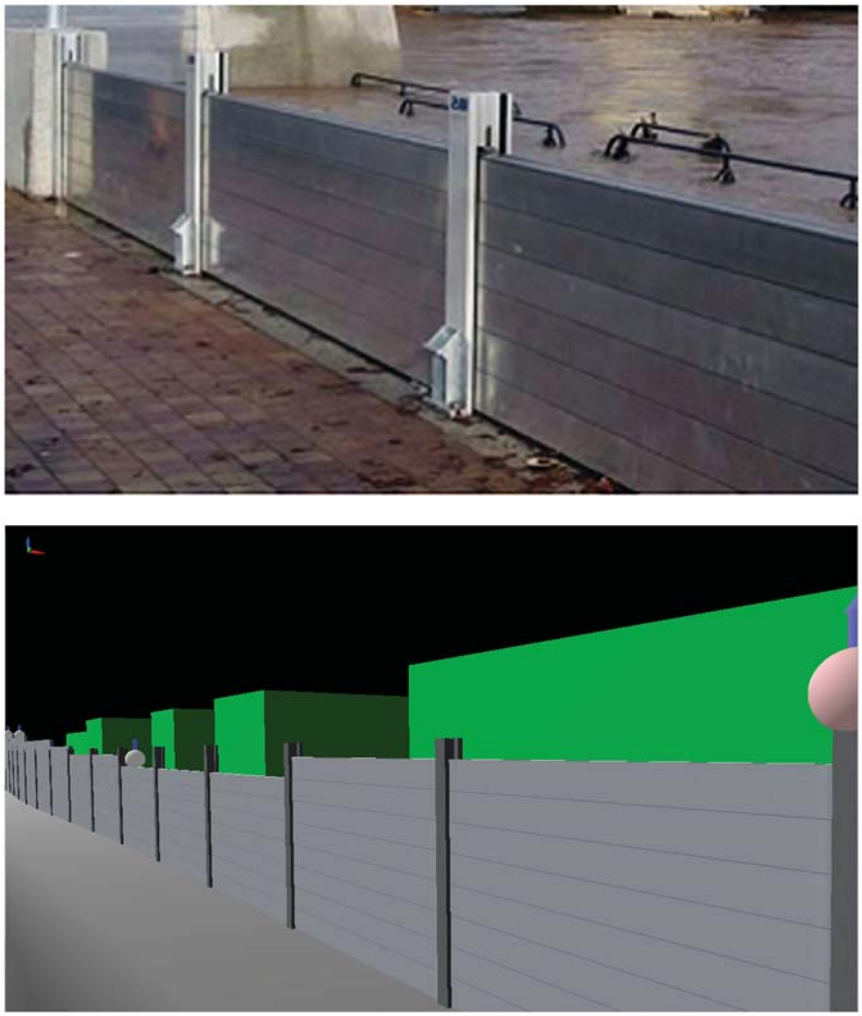

Figure 4.4: Mobile Protection Walls in real-world (top) and created by the system (bottom)

the behavior of the flood itself. The user should be able to change some properties of the fluid in order to test different scenarios. How this is realized in our system is the topic of the next section.

4.2 Forces

During floods, tremendous damage is caused due to the cumulative power which arises when masses of water are set into motion. A high amount of turbulence inside the fluid and a constantly changing environment lead to highly complex behavior. This is the reason why it is almost impossible to predict the actual movement and the characteristics of floods. When analyzing flooding scenarios by testing different barrier arrangements, this fact has to be kept in mind. An established set of barriers might be stable for the current situation but unstable for a later time step since the fluid can change quickly and unexpectedly. For a feasible solution, the user has to test the barriers under different circumstances. To support this, the system should provide the ability to change the parameters of the fluid interactively. Such parameters might be the flow velocity or the direction. Until now, the only way to affect the fluid is by manipulating the emitters. This method is insufficient for accomplishing a comprehensive analysis. A more intuitive and productive solution can be established when using the sketch-based interface. The user should be able to sketch onto the fluid and change or influence its behavior directly. Using this ability, the user can establish different scenarios where some parameters of the fluid could vary in order to test the established barrier arrangement. This allows the user to cope with the uncertainty of the natural phenomena as well as applying upcoming changes of circumstances like additional rain.

To realize the desired functionality, the Spline To Force Node was implemented. It uses the sketch-based interaction system and is responsible for translating the user-input sketch to forces which enable interaction with the fluid by influencing the simulation. The user input should be determined using the Spline Node. We provide two types of manipulation forces which can be used to achieve the desired result. The first one should manipulate the fluid in a natural way. This method influences the existing flow by adding realistic forces. With this method, natural phenomena like wind can be simulated. The second type should manipulate the fluid directly by adding pseudo forces.

These are forces which actually do not exist but induce a constraint on specific regions of the fluid. All forces have to be visualized to provide feedback. The user should have the ability to see how and in which areas specific sketches will influence the simulation. By manipulating the spline, the region where the forces have an effect on the fluid can be modified.

Like every node which wants to benefit from the sketch-based system, the Spline To Force Node has to fetch the control points provided by the Spline Node. The evaluation of the spline, including the calculation of the additional control points for the Bézier curves, is executed using the same functions as the Spline Node.

The main task of the Spline To Force Node is to create a force field around the spline. This field creates forces that influence the fluid simulation in the vicinity in order to realize the user's intent. The node provides the ability to introduce forces which act in two different ways. On the one hand, there are natural forces and on the other hand, pseudo forces. The first type creates forces which are simply added to the currently acting forces. This means the fluid is accelerated by a specific magnitude in the direction desired by the user. With this method, natural forces like wind or additional inflow can be simulated. The second method is a combination of using acceleration forces and additional forces to provide the ability to control the fluid directly. To achieve this, the additional forces act as counterparts to existing forces to cancel them out. With the help of these so-called control forces the user has the ability to modify the flow explicitly. The forces created by the Spline To Force Node influence the fluid only within a specific distance to the spline. The user can determine this range of influence by changing parameters within the settings object of the node.

The Spline To Force Node computes two different outputs. The first is used by simulation nodes in order to introduce the forces into the simulation system to influence the current behavior of the fluid. The second output is responsible for providing information which can be used by view nodes to visualize the forces created by the Spline To Force Node. For a generic realization of this task, the concept of data domains and fields is introduced. The basic idea is to split the information of the force fields into two different parts. The first part contains only the spatial field information and is represented by a

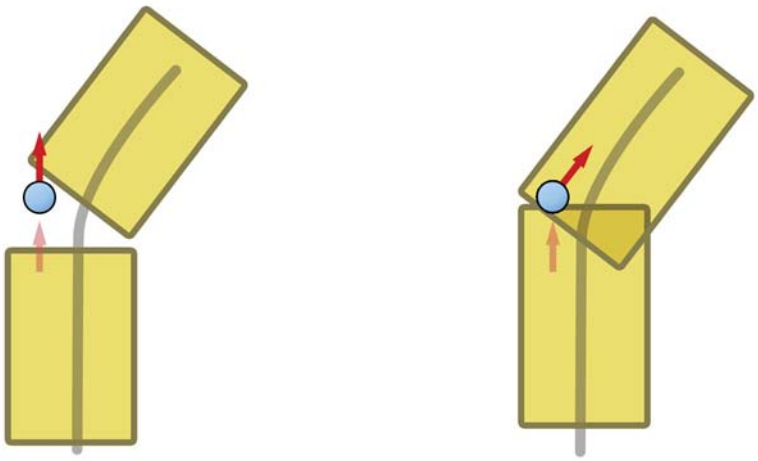

Figure 4.5: Arrangement without (left) and with (right) overlapping boxes. Overlapping ensures the correct application of forces.

data domain, whereas the second part stores the actual information of the forces within a vector field. The advantage of this approach is the ability to use multiple vector or scalar fields representing forces or other important values with the same data domain.

In order to establish additional forces within the current PhysX-based simulation system, so-called force fields have to be set up [18]. These objects are internally represented by simple geometric primitives. Forces act on elements which are inside the primitive. This requirement leads to an approach which is similar to the creation of the barriers. The Spline To Force Node creates boxes to represent force fields and arranges them along the spline. To realize this, the node evaluates the spline at specified intervals and checks if a new box can be created. Whether it can depends on the distance to the previous box. Unlike the boxes for the barriers, the boxes for force fields can overlap, which is actually desired. Overlapping the force fields ensures that the whole area around the spline is covered and that no gaps where the forces might not be applied exist (see Figure 4.5). In contrast to the barrier creation, only one orientation is available. The boxes are positioned along the spline so that their local x axis is aligned to the drawing direction of the spline. The region of influence can be modified by changing the size of the boxes since they represent force fields and forces are only applied on objects which are inside.

As with the barriers, the position and orientation of the boxes are stored in transformation matrices which then are transferred to the simulation node for further use. The first matrix in the vector is used as a helper to transmit additional information. It stores the dimensions of the boxes and the magnitude of the force which should be applied.

In order to realize the desired feature a new method has to be implemented in the simulation node. It should fetch the matrices and establish the PhysX force fields. The actual force which is applied within a field is determined by kernel functions. These functions mainly consist of four parts. The first is a constant force which is directly and additively applied to the existing one. In contrast, the second and the third part are position- and velocity-dependent, respectively. The last part is used to introduce noise if desired. For our purpose, only the velocity-dependent component is used. The basic idea is that the user specifies a desired velocity which should be achieved in the region of influence. The system then applies forces which are necessary to reach it. In order to realize this, the kernel function provides the ability to define a target velocity and a velocity multiplier. The target velocity is a vector which points towards the x axis of the force field, since this is the drawing direction of the spline. During the simulation, for every object, the difference between the current and the target velocity is calculated. Then the multiplier is used to calculate forces which should decrease the deviation. The multiplier is a vector which is multiplied with the difference between the target and the current velocity in order to determine how strong the forces have to be added to the objects.

The vector chosen for the multiplier determines the type of force which is applied in the region of influence. If the user wants to simulate natural forces, only the x component of the vector has to be set to a desired value, which can be interpreted as an acceleration force. This leads to the result that the value of this component is simply added to the existing force. To realize pseudo forces, and to enable the direct manipulation of the fluid, additional control forces have to be added. These forces decrease the velocity of the objects in all directions which are not the main direction of the target velocity. This means that, while the previously mentioned acceleration force increases the velocity in the x direction, the control forces decrease the velocities in the y and z direction.

The force field established by the Spline To Force Node enables the interactive manipulation of the fluid. But the generated forces should not effect other objects in the scene, like the barriers or the terrain geometry. PhysX provides a mechanism where the magnitude of the force can be scaled according to the type of object which is inside the force field. Using it properly, allows the field to act only on the particles of the fluid.

Feedback Visualization

Since force fields are invisible, the user has no visual feedback about the region of influence and other important properties. The aim of the system is to allow for experimenting and testing different solutions by modifying the velocity of the fluid. The sketch-based interface provides a fast way to do this but without appropriate feedback the user can not preview and analyze the decisions made. To solve this problem, the Spline To Force Node provides a data domain and a field in order to enable the visualization of the force field which is transmitted to the simulation node. The data domain can be every spatial structure, for example a set of particles or, like in the case of force fields, a grid. To create the field which contains the actual information, the area around the spline has to be sampled. After calculating the positions for the force field boxes, the Spline To Force Node creates the Grid Domain. The origin and dimensions are chosen in a way that all boxes are covered by the grid. To sample the area around the spline, the node iterates through all grid cells and checks if the current position is inside one of the previously calculated boxes. If it is not, the corresponding value in the vector field is set to zero. Otherwise the Spline To Force Node calculates a vector which points in the direction of the x axis of the box and has the length of the desired magnitude. If the grid cell position is within a region where boxes overlap, the average of the boxes' x axes is calculated to determine the force direction. To avoid skipping a box, the sample rate has to be small enough. This rate can be determined by choosing a specific voxel size during the definition of the Grid Domain. After finishing, both the data domain and the field are sent to the view node in order to be visualized.

The task of the view node is to visualize the vector field which is defined as a combination of the data domain and the field containing the actual forces. In order to realize

this, methods of flow visualization can be used. An overview is provided by Weiskopf and Erlebacher [95]. For our purpose, we use a point-based direct visualization where visual representations for points are created. We choose to map the information to glyphs shaped like arrows and render them in the 3D view in order to visualize the characteristics of the force field. A so-called arrow plot is easy to understand and a clear and simple, but effective, way to present vector fields.

To implement the feedback visualization of the force fields the view node was altered to be able to cope with the data domain and vector field structure. The 3D view node consists of multiple components called renderables. Each renderable is responsible for the rendering of a specific type of object. For example, the clone renderer is able to render multiple copies of the same triangle mesh which is used for the barriers. The geometry renderer is the standard component and is responsible for rendering triangle meshes in a quick and efficient way. We use it to visualize the spline itself. Another important renderable is the particle point sprite renderer which ensures that the fluid is rendered as a smooth surface.

In order to visualize the force field, a new renderable called glyph renderer was implemented. It renders glyphs in desired intervals in order to create an arrow plot. Since the user can choose to connect multiple data domains and fields to the view node, the first task of the renderable is to fetch all the inputs and store them for further use. Then the glyph renderer iterates through the inputs and samples the corresponding fields to obtain the information needed to visualize the glyphs. For most data domains, Visdom provides an appropriate sampler which implements the sampling algorithm needed. The user can define the sampling rate and thus the density of the rendered arrow plot through the settings object of the glyph renderer. The most important values for our purpose are the direction and the magnitude of the force. The direction is mapped onto the orientation of the arrow whereas the magnitude can be mapped onto multiple properties of the glyph. The first option is to visualize the magnitude as the length of the arrow (see Figure 4.6). The user can define a minimum and a maximum length to determine the representation of the points with the smallest and the largest magnitude. Another mapping onto a geometrical property implemented by the glyph renderer is the mapping onto the thickness of the arrows (see Figure 4.7). In this case, a minimum and

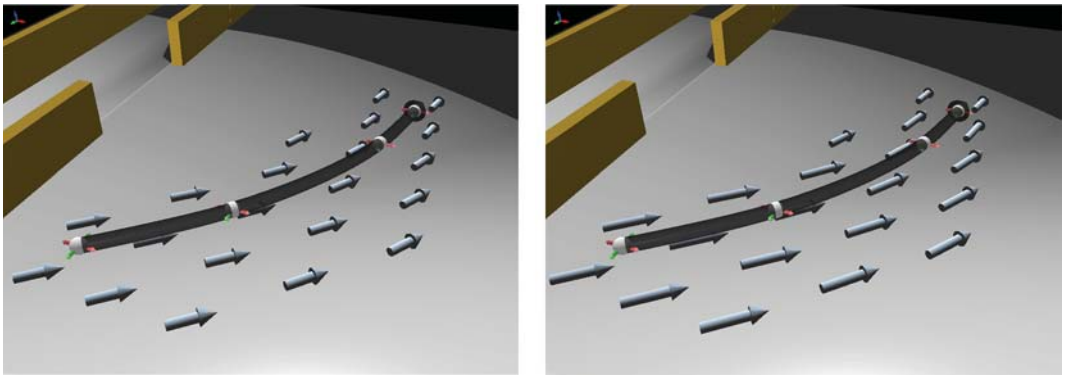

Figure 4.6: Magnitude visualization: The left side shows the default visualization of the force field. The right side shows an arrow plot where the magnitude of the force is visualized as the length of the arrows.

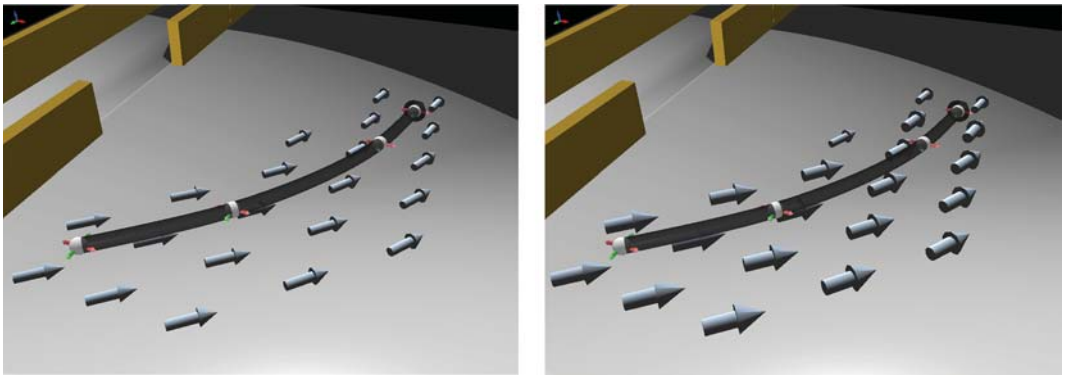

Figure 4.7: Magnitude visualization: default (left) and as thickness (right)

maximum value can also be defined by the user. The last mapping option is the visualization of magnitudes using the color of the glyphs (see Figure 4.8). This method uses a function to determine the color used for a specific magnitude. The user can employ the Transfer Function Node which is already implemented in Visdom. This component provides a transfer function which can be modified in an interactive component within the semantic view (see Figure 4.9). Every input set can have its own Transfer Function Node connected. No matter which option is used, if a point has a magnitude of zero, no arrow is rendered in order to keep the visualization clear. Figure 4.10 shows the combination of all visualization types.

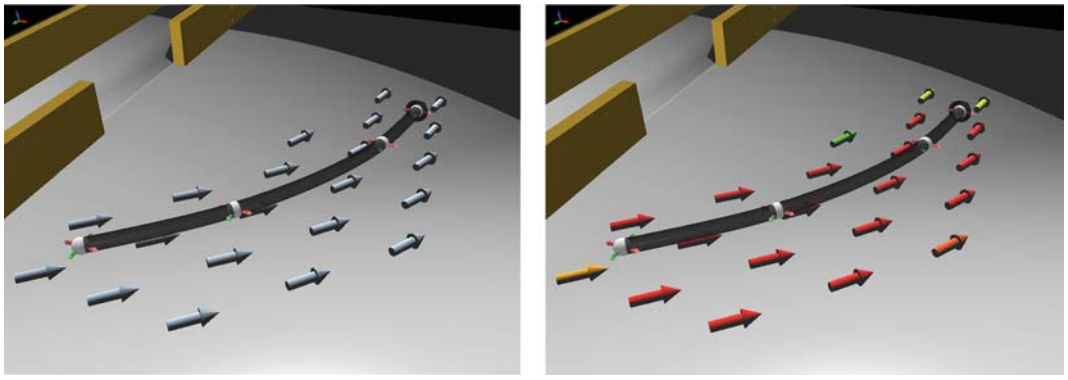

Figure 4.8: Magnitude visualization: default (left) and as color (right). Red and orange arrows indicate high magnitude whereas yellow and green arrows show regions with lower magnitude.

The glyph renderer is also able to cope with a set of particles as a data domain. This allows an arrow plot which shows flow properties of the fluid to be rendered (see Figure 4.11). To sample the particles a grid is used, where the origin and the dimensions are chosen in such a way that all particles are inside. During the sampling process, the particle sampler calculates a so-called kernel sum. This value indicates how dense the fluid at the sampling point is. If this value is under a specific threshold, no arrow is drawn.

Chapter Summary

The presented functionality of the Spline To Barrier Node allows the user to manipulate the fluid representing the flood indirectly by establishing barriers. Combined with the Spline Node, a sketch-based interface supports this task. Multiple arrangements can be set up and analyzed due to the interactive feedback visualization and simulation. The user can modify the barriers using the spline until a desired result is achieved. This intuitive and fast workflow supports the user in emergency situations where human lives are at stake.

The Spline To Force Node enables the user to manipulate the fluid directly by influencing the velocity using force fields. The used sketch-based interaction scheme allows for fast and intuitive control which increases the productivity of the workflow. The user is able to test the previously arranged barriers under different flood conditions. With the

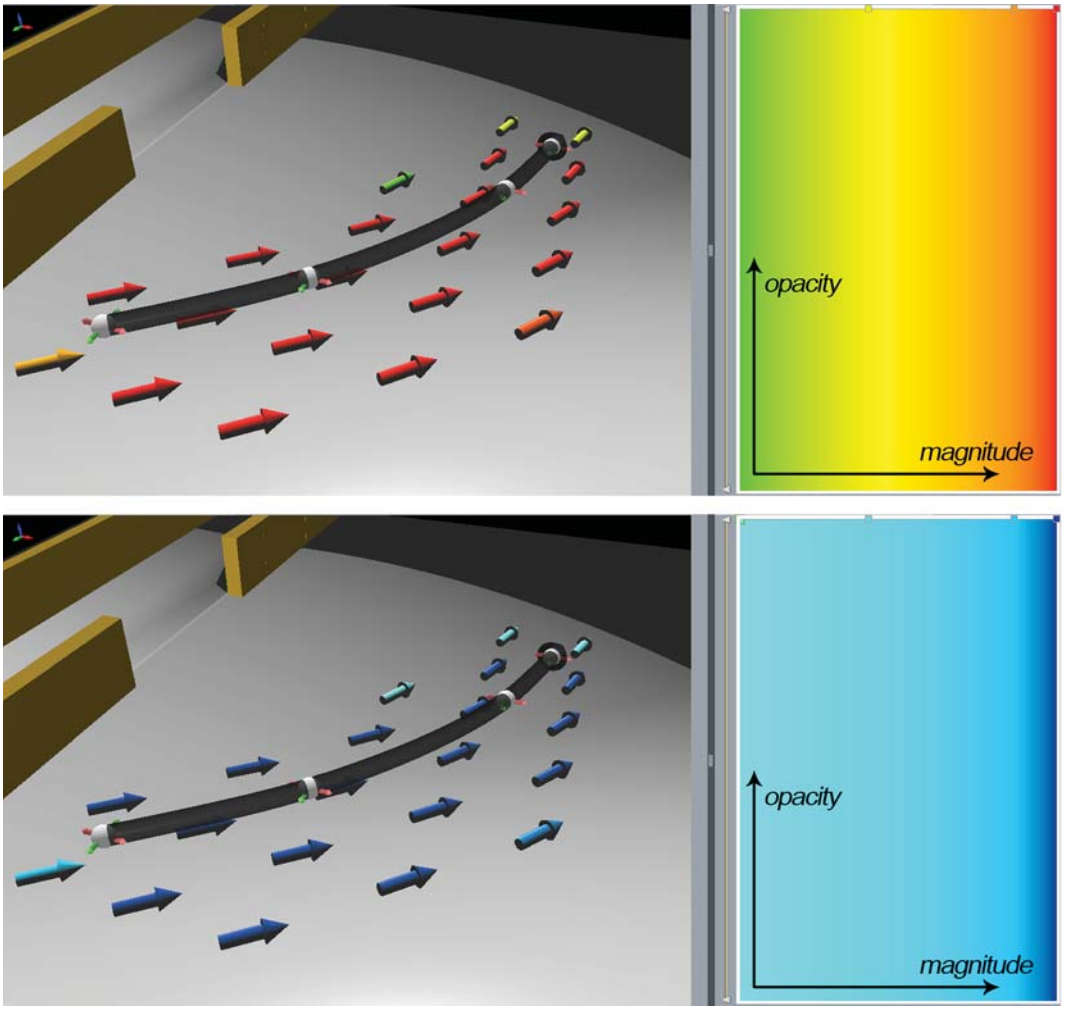

Figure 4.9: Magnitude as color visualization with two different transfer functions

support of the glyph renderer, interactive visual feedback is available which simplifies the creation of the desired force fields.

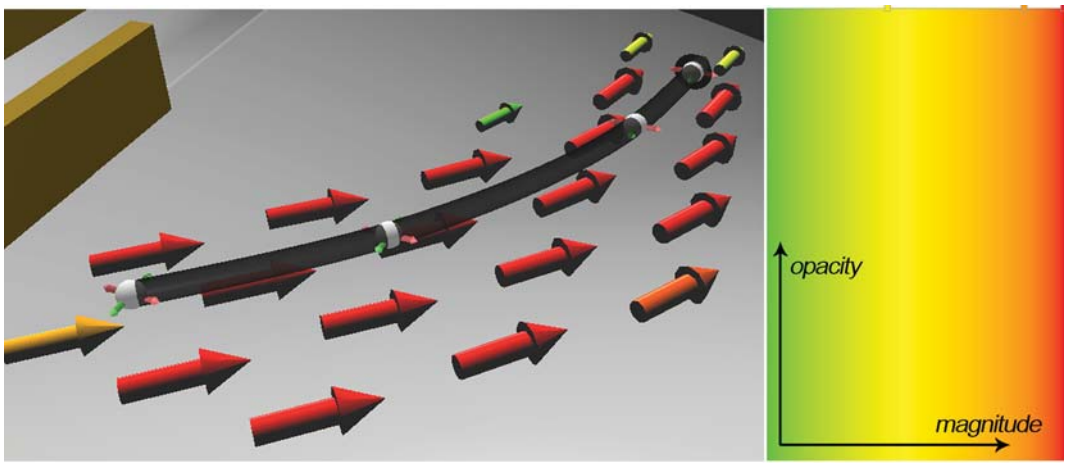

Figure 4.10: Magnitude visualization as length, thickness and color

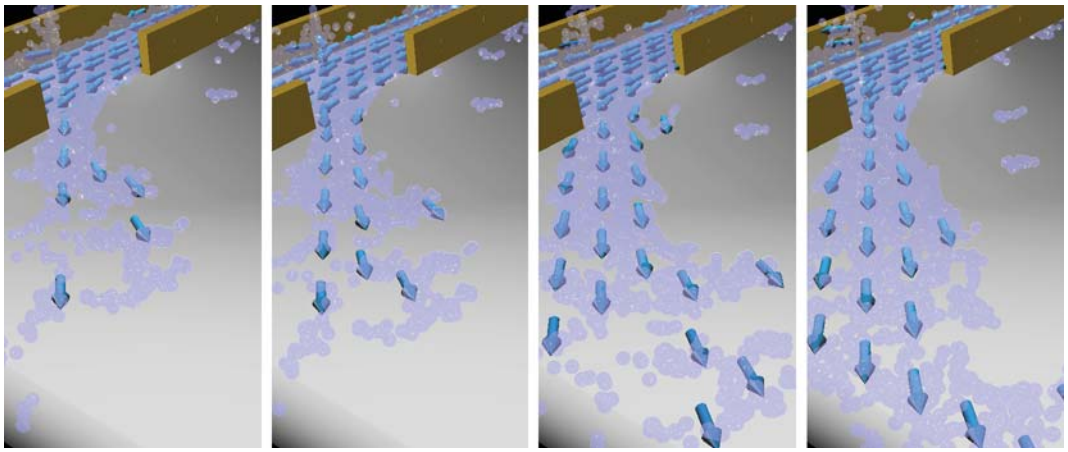

Figure 4.11: Flow visualization using an interactive arrow plot

CHAPTER 5

Case Study

Visdom is intended to support decision makers during the flood management process. Due to changing conditions of the water ecosystem, land use, and population, flood-risk management is a dynamic cyclic process [25]. This cycle consists of three main steps which are of equal importance (see Figure 5.1). Measures are classified in preparedness, response and recovery.

The main goal of the first step is to reduce the vulnerability of people and property to natural hazards. Prevention measures try to avoid damage due to appropriate land-use planning. In case where this is not possible, technical measures have to be established in order to avert or minimize damage. They include protective structures such as levees, embankments or protection walls. Preparation measures instead should help to overcome the disaster. Important tasks are the development of flood-action plans as well as the training of rescue units and action forces. Flood-action plans define measures to mitigate damage during a flooding event.

The second step of the flood-risk management-cycle is the response to the actual disaster. It aims to limit the duration and the extent of damage caused by the flood. Besides alerting and rescuing victims, measures to prevent further damage are included. These can be the establishment of sandbag barriers or the evacuation of people from insecure areas. Barriers have to be constructed by the on-site action force in cases where the ex-

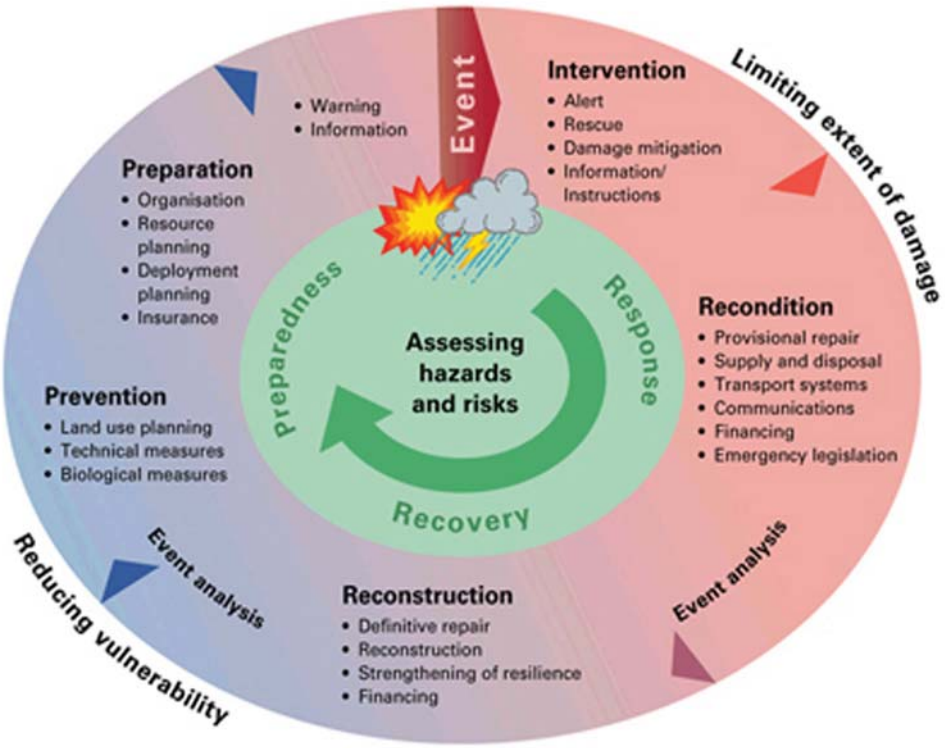

Figure 5.1: The flood risk management cycle [25]

isting flood-protection system fails. Furthermore, this step is responsible for provisional repairs of vital infrastructure such as electricity-supply facilities.

The last step is the recovery phase. Reconstruction of damaged buildings and infrastructure, and the analysis of the disaster are the main issues.

The sketch-based interface implemented within Visdom can be utilized to support several measures of the flood-risk management-cycle. During the preparedness phase, it can be used to design the protective structures such as protection walls. The user can sketch ideas of wall arrangements and test them under different circumstances in an efficient and intuitive manner. Furthermore, the sketch-based system supports the development of flood-action plans. After a complete protection system is established, the temporal coordination of all measures can be developed and tested. Additionally, the sketch-based system can be used to train the action force team.

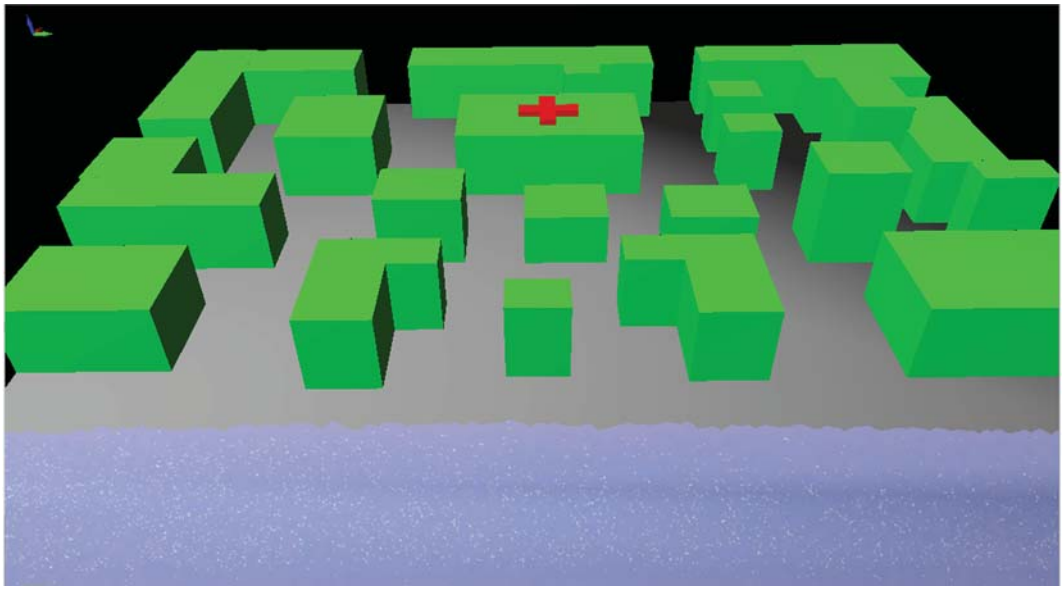

Figure 5.2: Overview of the scenario. Several Buildings and a nearby river.

The phase which benefits most from an intuitive user interface is the response. In emergency situations, fast decision-making is essential. When action forces arrive on site, quick acting is needed to minimize the damage in the vicinity. Weak points have to be found and eliminated by establishing sandbag barriers immediately.

The following case study demonstrates a typical workflow during flooding scenarios. It shows how the sketch-based interface supports the user to increase productivity.

The scenario setup consists of several buildings and a nearby river (see Figure 5.2). There exists a building with higher priority such as a hospital.

We assume that severe weather conditions have lead to a raise of the water line. After loading geological and hydrological data, the on-site action-force team-leader can simulate the scenario. Without any flood protection measures the village is flooded (see Figure 5.3).

In order to protect the neighborhood mobile protection walls are established. According

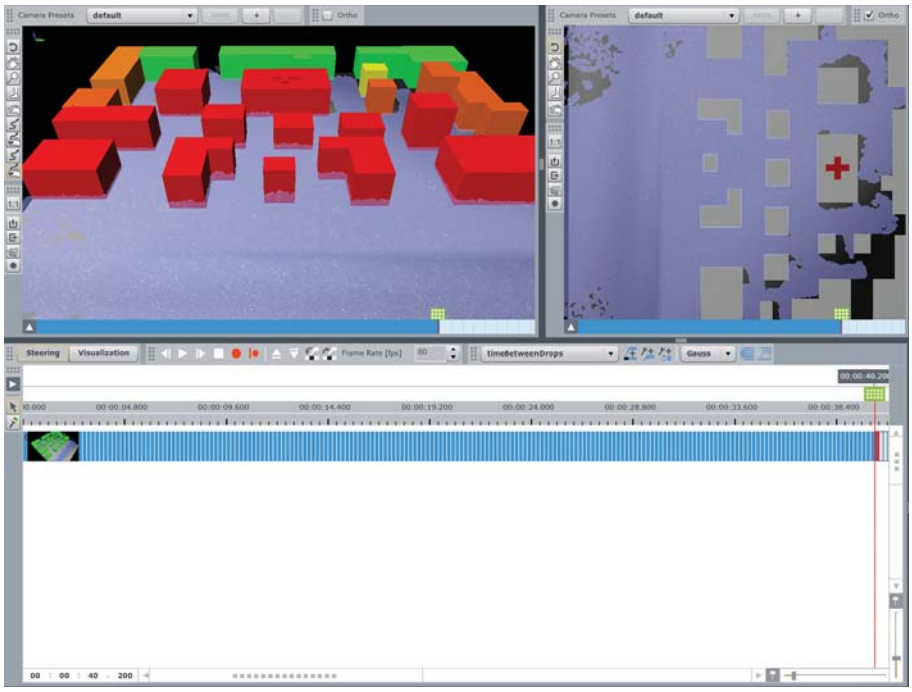

Figure 5.3: The neighborhood is flooded when the water line raises due to the lack of protection structures.

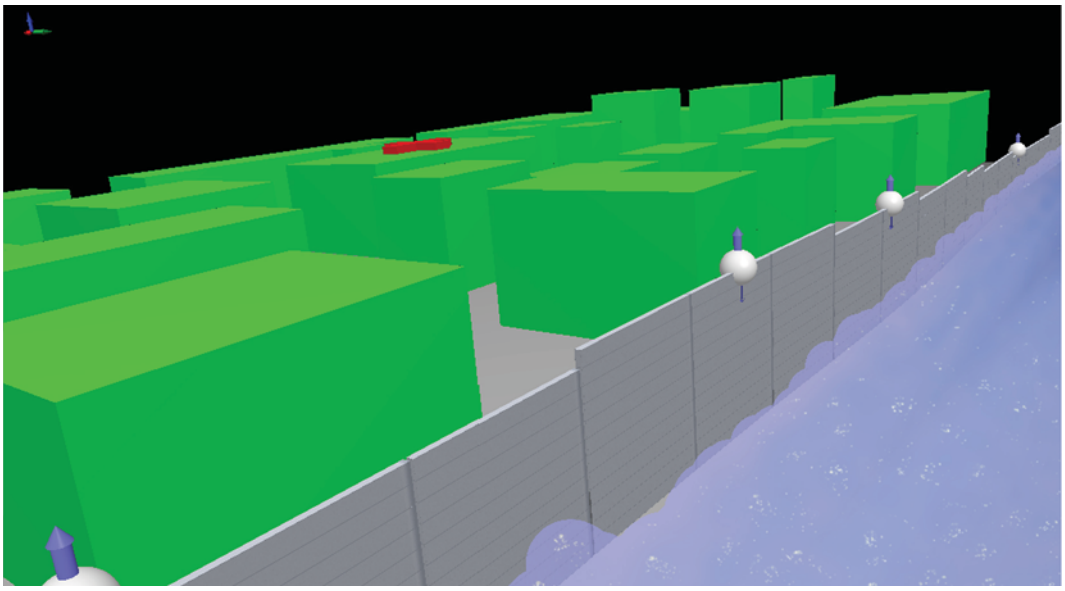

Figure 5.4: Mobile protection wall sketched into the 3D scene. The height of the wall components can be determined using the control points.

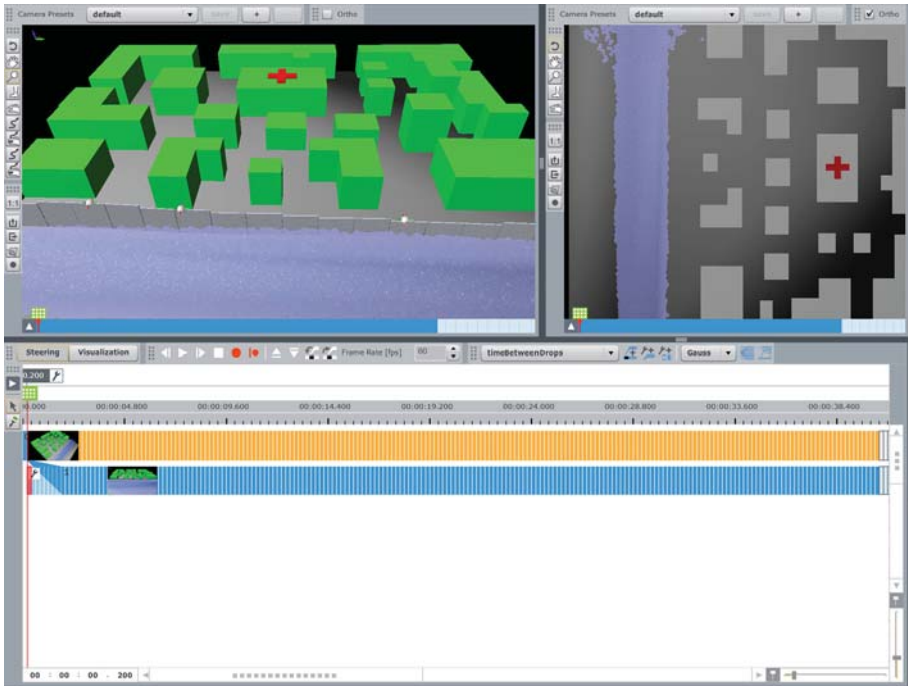

Figure 5.5: Overview of the scenario including a protection wall

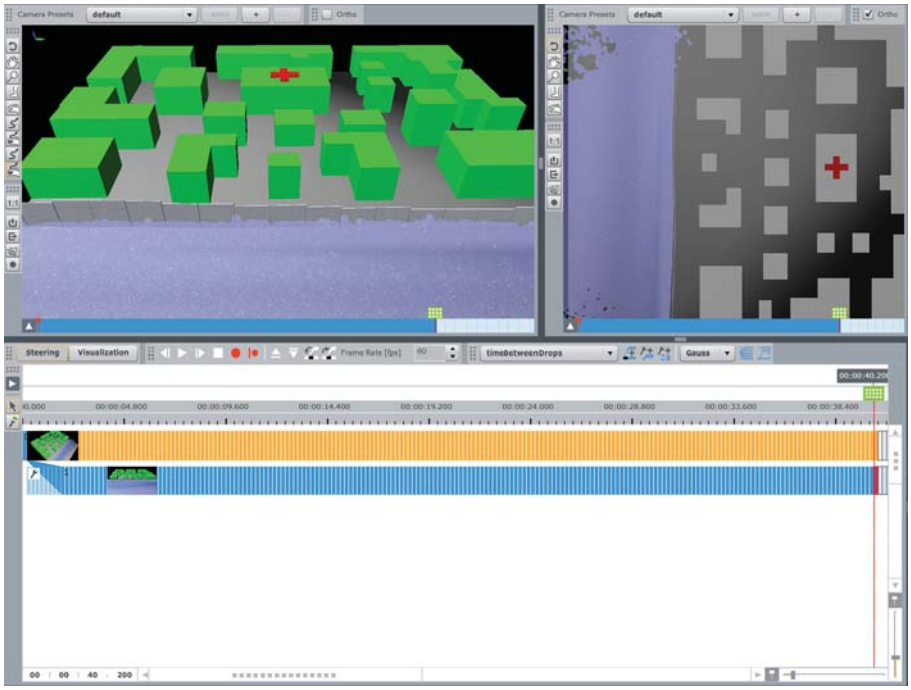

Figure 5.6: The simulation shows that the wall protects the neighborhood sufficiently.

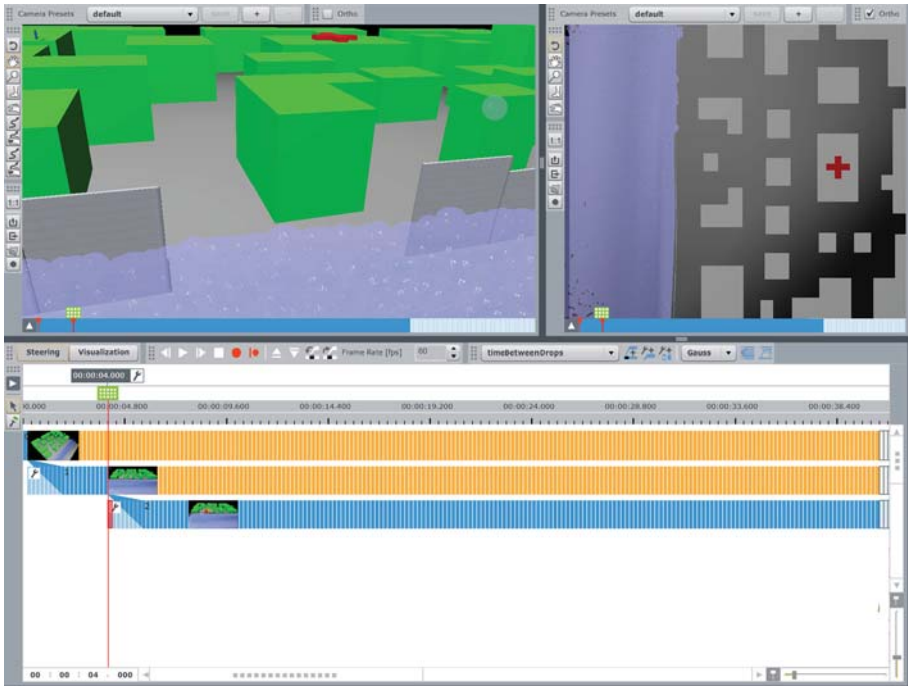

Figure 5.7: The system allows to create breaches to simulate different situations.

to developed flood-action plans, the team leader sketches the walls into the 3D scene see Figure 5.4). Figure 5.5 shows the scenario with the sketched protection wall. The interaction leads to a branch in the world-lines view. Simulating the new scene shows that the buildings are save due to the protection wall (see Figure 5.6).

In order to carry out catastrophe planning, the action-force team-leader investigates wall-breach scenarios. The system provides the ability to create breaches at any position (see Figure 5.7). The introduction of the breach results in the flooding of the neighborhood (see Figure 5.8). The target of the on-site action force is to establish a stable barrier arrangement which protects the important buildings. The protection should hold even if the impact of the flood waves increases.

Relying on previous knowledge and experience, the action-force team-leader tries different barrier arrangements by sketching them into the 3D view (see Figure 5.9). The sketch-based interface allows for a quick investigation of multiple alternatives. The user explores the options until a stable solution is found which protects the high priority

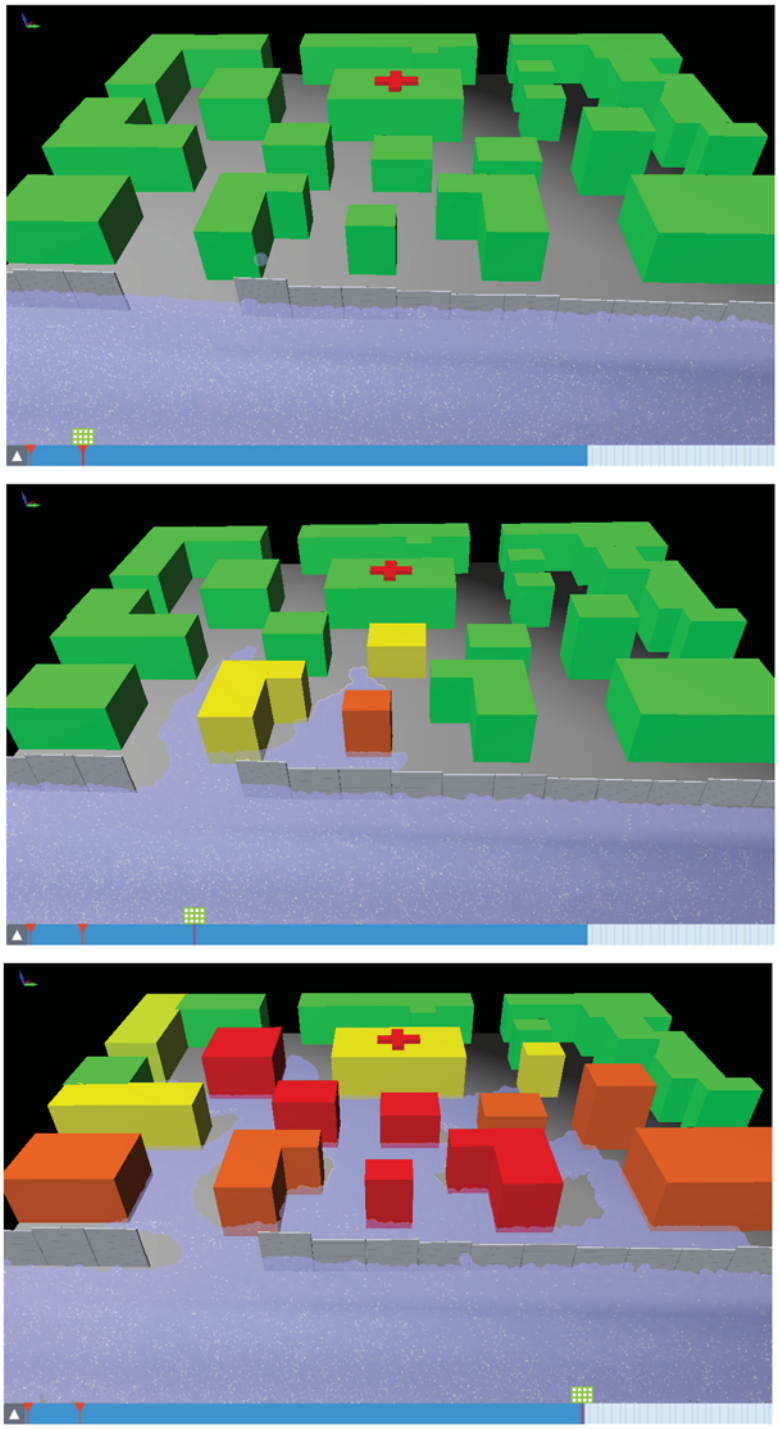

Figure 5.8: Three screenshots of three different time-steps during the simulation of the breach scenario. Due to the destroyed mobile wall component the neighborhood is flooded. Red indicates highly flooded buildings. Orange and yellow buildings are submerged partly whereas green buildings are currently secure.

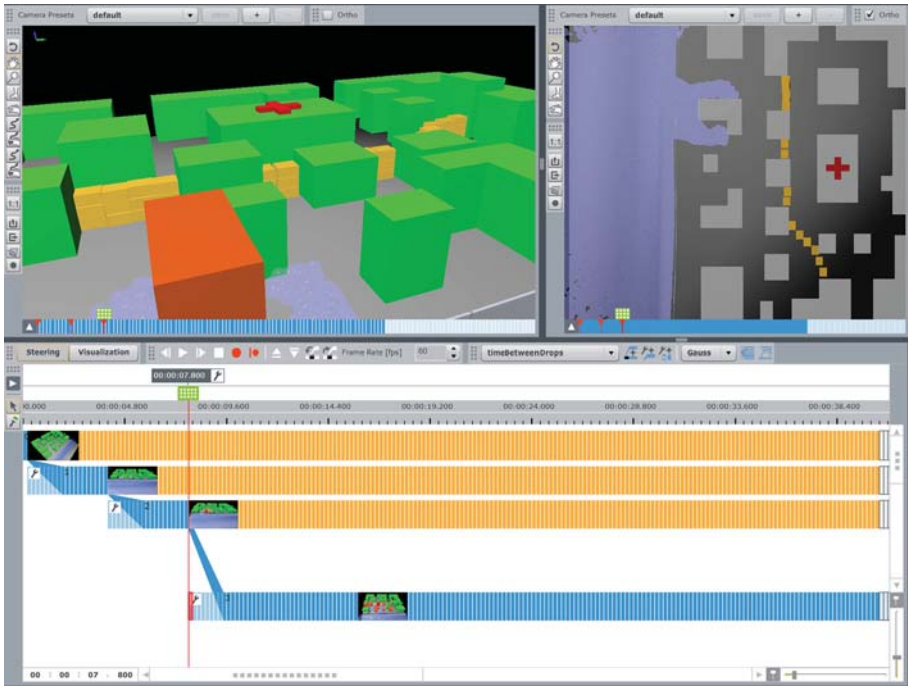

Figure 5.9: Sandbag barriers can be directly sketched into the 3D scene using the sketch-based interface.

building sufficiently (see Figure 5.10).

Due to the time-critical situation, the action-force team-leader cannot simulate for a long time in order to test if the solution remains stable. Using the presented sketch-based interaction system, the user can add forces to check the performance of the barrier under severe conditions (see Figure 5.11). The ability to control the water-flow by sketching forces allows for proving whether the found solution remains stable even if the pressure of the river increases. Figures 5.12 and 5.13 show that the barrier holds for both forces tested.

Using the flow-visualization functionality of the presented system provides further information about the internal behavior of the water. This knowledge can be used to support the solution-development process.

This case study demonstrates the fast and intuitive characteristics of the sketch-based

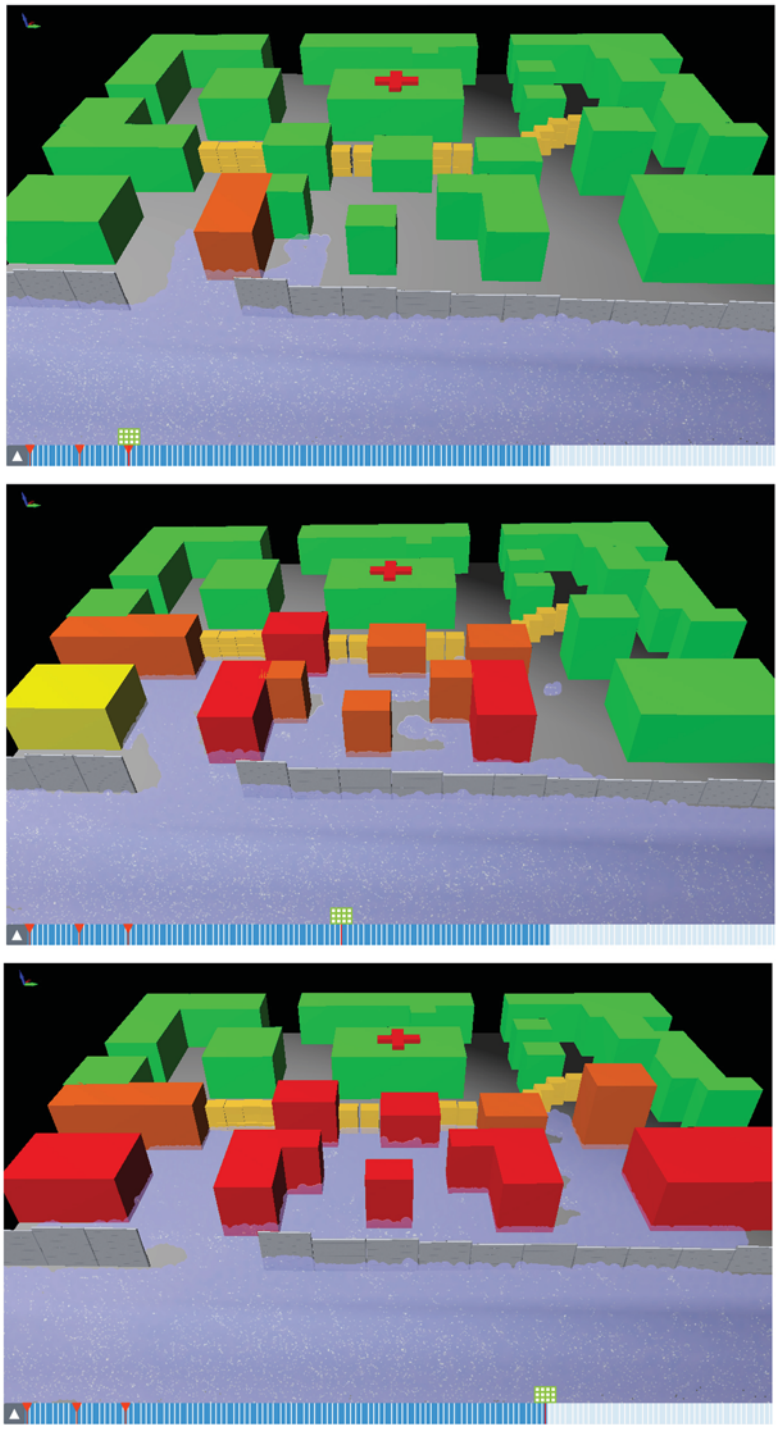

Figure 5.10: Simulating the scenario with the barrier shows that the arrangement protects the hospital sufficiently.

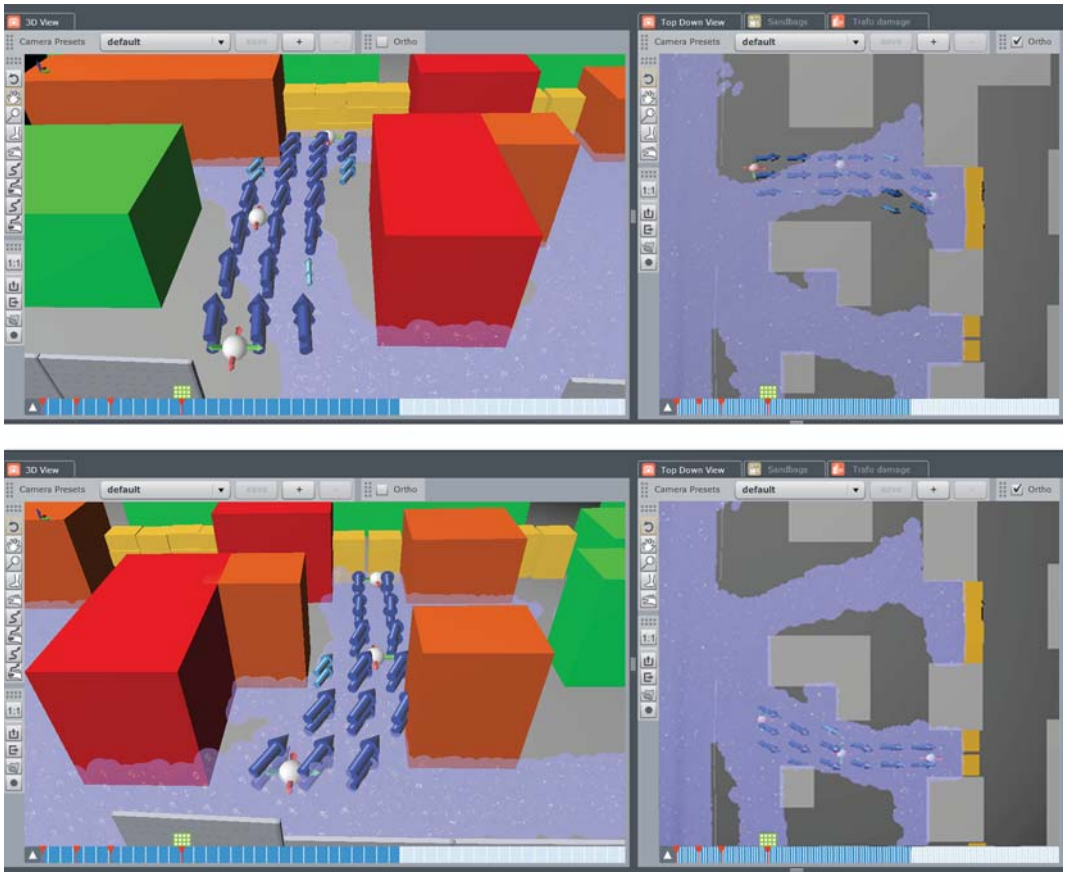

Figure 5.11: Several forces can be sketched into the 3D view in order to test the barrier under different circumstances.

interface. It allows for a productive and efficient work-flow in time-critical situations. The action force can respond quickly to unexpected or anticipated events. They are able to test different solutions in order to find appropriate countermeasures to avoid further damage.

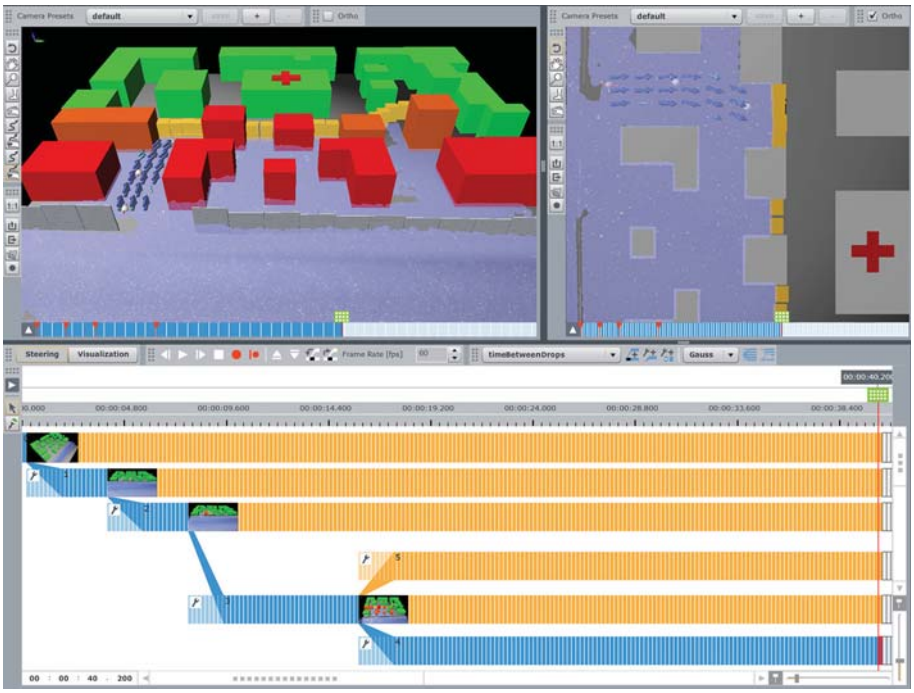

Figure 5.12: Simulating additional forces shows that the barrier holds.

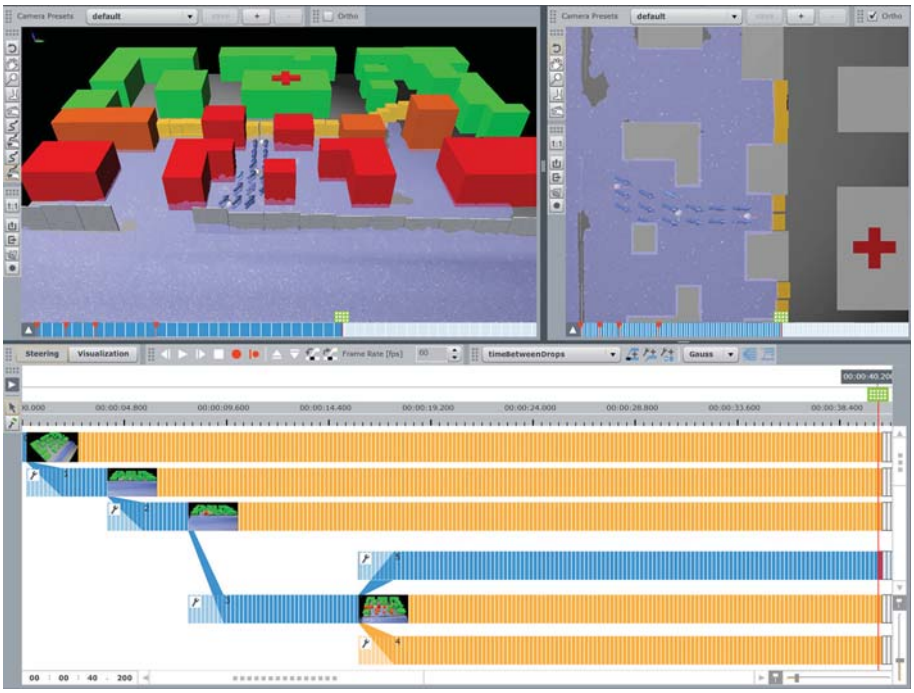

Figure 5.13: Further forces can be sketched to confirm the stability of the arrangement.

CHAPTER 6

Conclusion

The contribution of this thesis is the integration of a sketch-based interaction interface into Visdom which allows for interactive fluid simulation steering. Floods are everlasting and unpredictable hazards with tremendous consequences. In emergency situations, a fast decision-making process is essential in order to protect people living in the vicinity and their property. Visdom is an integrated simulation and visualization application which in combination with the introduced sketch-based interface, forms a feasible system for this task. It also supports the creation of flood-risk plans and strategies as ordered by the European Union and the training of emergency-task forces.

To integrate the interface into a data-flow based system the concept of Modular Interactors is introduced which enables arbitrary node interaction through view modules. Every component of the sketch-based interaction system was implemented in a generic way to ensure reusability. Interactors can be used in combination with any new or existing node if inter-module interaction is needed.

For the purpose of implementing the sketch-based interface, the Spline Node was designed which uses the modular interactors concept to communicate with the 3D view node. It encapsulates the functionality of creating a spline using points which are captured from the user input. Furthermore, the ability to modify the spline by means of control points is provided. To increase the usability, the concept of view-dependent ma-

nipulation was introduced, which constrains the degrees of freedom depending on how the user looks onto the spline. Due to the generic realization of the implementation, the functionality can be used for any other task where a spline is needed.

In order to interpret the sketch to enable the manipulation of the simulation, two additional nodes were introduced. The Spline To Barrier Node is responsible for creating barriers according to the sketched spline. Arbitrary geometry can be used to represent the components of the barriers. This new node provides the ability to sketch sandbag arrangements or mobile protection walls directly into the 3D view. This leads to an increase of productivity in comparison to the alternative way of positioning sandbags one by one.

The spline can also be interpreted as a command to manipulate the fluid directly. The Spline To Force Node allows the user to influence the simulation by adding forces. The user can choose between applying a natural force like wind or a pseudo force to control the fluid explicitly. The spatial information of the generated force field is separated from the actual information about the force. This is the precondition for a generic use of the data domain with an arbitrary vector field containing the necessary values for simulation or visualization.

In every aspect of the presented techniques, a feedback visualization is provided. The user obtains feedback on the commands triggered and on how they will affect the simulation. The user's sketch is visualized by a tubular geometry which can be modified via control points. Barrier arrangements can be previewed as transparent objects and adjusted if necessary. To provide visual feedback for the direct fluid manipulation, the force field is visualized. A new render component was introduced to allow for the visualization of glyphs. The Glyph Renderer gets a data domain of any type and a vector field as input parameters and generates an arrow plot in order to visualize the force field influencing the simulation or the flow of the fluid.

The intuitive sketch-based interface in combination with the interactive feedback visualization supports a fast and productive work-flow. It allows for short interaction cycles and simplifies the modification of simulation states and the analysis of the effects. All

this features are supportive during experimenting tasks. Thus, testing alternative flood scenarios can be done in a more efficient way.

The next step would be to use sketching in order to cope with uncertainty. The user should be able to define several simulation runs at once to perform a comprehensive parameter study. The sketch-based interface can be used to ease the setup and the control of these studies. By sketching the start and end values of an input parameter, a range of simulation runs can be launched via interpolation. The system has to provide a sufficient feedback visualization in order to support the user during the establishment of the parameter studies. Another task for future work would be visualizing robustness of sketched protection walls or barriers. This feedback would support the user during the analysis of flood scenarios.

Bibliography

[1] Visdom - An integrated visualization System. `http://www.visdom.at`, (last visited on 30 October 2011), 2011.

[2] T. Amada, M. Imura, Y. Yasumuro, Y. Manabe, and K. Chihara. Particle-Based Fluid Simulation on GPU. *ACM Workshop on General-Purpose Computing on Graphics Processors and SIGGRAPH 2004 Poster Session*, 2004.

[3] C. F. Andersen and American Society of Civil Engineers. Hurricane Katrina External Review Panel. *The New Orleans hurricane protection system: what went wrong and why : a report*. ASCE, 2007.

[4] R. Baldoni, M. Contenti, and A. Virgillito. The evolution of publish/subscribe communication systems. In *Future directions in distributed computing*, pages 137–141. Springer-Verlag, Berlin, Heidelberg, 2003.

[5] R. Beatson and L. Greengard. A short course on fast multipole methods. In *Wavelets, Multilevel Methods and Elliptic PDEs*, pages 1–37. Oxford University Press, 1997.

[6] L. Begnudelli, B. F. Sanders, and S. F. Bradford. Adaptive Godunov-Based Model for Flood Simulation. *Journal of Hydraulic Engineering*, 134(6), 2008.

[7] G. Blöschl, C. Reszler, and J. Komma. A spatially distributed flash flood forecasting model. *Environ. Model. Softw.*, 23:464–478, April 2008.

[8] R. Bridson. *Fluid Simulation for Computer Graphics*. CRC Press, September 2008.

[9] K. Brodlie, A. Poon, H. Wright, L. Brankin, G. Banecki, and A. Gay. GRASPARC: a problem solving environment integrating computation and visualization. In *Proceedings of the 4th conference on Visualization '93*, VIS '93, pages 102–109, Washington, DC, USA, 1993. IEEE Computer Society.

[10] J. M. Brooke, P. V. Coveney, J. Harting, S. Jha, S. M. Pickles, R. L. Pinning, and A. R. Porter. Computational Steering in RealityGrid. In *Proceedings of the UK e-Science All Hands Meeting*, pages 885–888, 2003.

[11] G. W. Brunner and V. R. Bonner. HEC River Analysis System. Technical Paper TP-147, US Army Corps of Engineers, Institute for Water Resources, Hydrologic Engineering Center (HEC), 609 Second Street, Davis, CA 95616-4687, August 1994.

[12] S. R. Buss. *3D Computer Graphics: A Mathematical Introduction with OpenGL*. Cambridge University Press, New York, NY, USA, 2003.

[13] G. Cameron. Modular visualization environments: past, present, and future. *SIGGRAPH Comput. Graph.*, 29:3–4, May 1995.

[14] P. Chatelain, M. Bergdorf, and P. Koumoutsakos. Large Scale, Multiresolution Flow Simulations Using Remeshed Particle Methods. In M. Griebel and M. A. Schweitzer, editors, *Meshfree Methods for Partial Differential Equations IV*, volume 65 of *Lecture Notes in Computational Science and Engineering*, pages 35–46. Springer Berlin Heidelberg, 2008.

[15] M. H. Chaudhry, M. Elkholy, and C. Riahi-Nezhad. Investigations on Levee Breach Closure Procedures. Technical report, University of South Carolina, Department of Civil and Environmental Engineering, November 2010.

[16] J. J. Cherlin, F. Samavati, M. C. Sousa, and J. A. Jorge. Sketch-based modeling with few strokes. In *Proceedings of the 21st spring conference on Computer graphics*, SCCG '05, pages 137–145, New York, NY, USA, 2005. ACM.

[17] NVidia Corporation. CUDA: Parallel computing architecture. `http://www.nvidia.com/object/cuda_home_new.html`, (last visited on 30 October 2011), 2011.

[18] NVidia Corporation. Physx: Physics simulation toolkit. `http://www.nvidia.de/object/physx_new_de.html`, (last visited on 30 October 2011), 2011.

[19] R. A. Dalrymple and A. Herault. Levee breaching with GPU-SPHysics code. In *Proceedings of the 4th SPHERIC International Workshop*, pages 352–355, 2009.

[20] K. Das, P. Diaz-Gutierrez, and M. Gopi. Sketching Free-form Surfaces Using Network of Curves. In *Proceedings of the Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pages 127–134, Dublin, Ireland, 2005. Eurographics Association.

[21] J. Davis, M. Agrawala, E. Chuang, Z. Popović, and D. Salesin. A sketching interface for articulated figure animation. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '03, pages 320–328, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[22] J. F. Dhondia and G. S. Stelling. Application of one dimensional - two dimensional integrated hydraulic model for flood simulation and damage assessment. In *Hydroinformatics 2002: Proceedings of the Fifth International Conference on Hydroinformatics*. IWA, July 2002.

[23] H. Doleisch. *Visual Analysis of Complex Simulation Data using Multiple Heterogenous Views*. PhD thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, October 2004.

[24] EU. Directive 2007/60/EC of the European Parliament and of the Council of 23 October 2007 on the assessment and management of flood risks. *Official Journal of the European Union L288/27*, 2007.

[25] United Nations Economic Commission For Europe. *Transboundary Flood Risk Management: Experiences from the UNECE Region*. United Nations Publications, 2009.

[26] T. Fleisch, F. Rechel, P. Santos, and A. Stork. Constraint Stroke-Based Oversketching for 3D Curves. In *Proceedings of the Eurographics Workshop on Sketch-*

Based Interfaces and Modeling, pages 161–165, Grenoble, France, 2004. Eurographics Association.

[27] Centre for Research on the Epidemiology of Disasters (CRED). 2010 disasters in numbers. `http://cred.be/sites/default/files/PressConference2010.pdf`, (last visited on 30 October 2011), 2011.

[28] J. N. Ghazali and A. Kamsin. A real time simulation of flood hazard. In *Computer Graphics, Imaging and Visualisation, 2008. CGIV '08. Fifth International Conference on*, pages 393 –397, aug. 2008.

[29] M. Gouda, K. Karner, and R. Tatschl. Dam Flooding Simulation Using Advanced CFD Methods. In *WCCM V, Fifth World Congress on Computational Mechanics*, July 2002.

[30] U. Grünwald. Vielzahl von Hochwasser im Jahr 2010 an Oder, Neiße, Spree und Schwarzer Elster - eine Herausforderung nicht nur für die wasserwirtschaftliche Praxis. In *Hydrologie & Wasserwirtschaft - von der Theorie zur Praxis*, number 30.11, pages 21–28. Fachgemeinschaft Hydrologische Wissenschaften in der DWA, March 2011.

[31] R. B. Haber and D. A. McNabb. Visualization idioms: A conceptual model for scientific visualization systems. *Visualization in Scientific Computing*, pages 74–93, 1990.

[32] T. Harada, S. Koshizuka, and Y. Kawaguchi. Smoothed particle hydrodynamics on GPUs. In *Proceedings of Computer Graphics International*, pages 63–70, 2007.

[33] M. Hassan and N. A. Dodgson. Reverse subdivision. In N. A. Dodgson, M. S. Floater, M. A. Sabin, G. Farin, H. C. Hege, D. Hoffman, C. R. Johnson, and K. Polthier, editors, *Advances in Multiresolution for Geometric Modelling*, Mathematics and Visualization, pages 271–283. Springer Berlin Heidelberg, 2005.

[34] A. Herault, G. Bilotta, R. A. Dalrymple, E. Rustico, and C. Del Negro. GPUSPH (Version 1.0) [Software]. `http://www.ce.jhu.edu/dalrymple/GPUSPH`, (last visited on 30 Octoer 2011),2010.

[35] R. C. Hoetzlein. FLUIDS v.2 - A Fast, Open Source, Fluid Simulator. http://www.rchoetzlein.com/eng/graphics/fluids.htm, (last visited on 30 October 2011), 2011.

[36] T. Igarashi. Freeform user interfaces for graphical computing. In *Proceedings of the 3rd international conference on Smart graphics*, SG'03, pages 39–48, Berlin, Heidelberg, 2003. Springer-Verlag.

[37] T. Igarashi, R. Kadobayashi, K. Mase, and H. Tanaka. Path drawing for 3d walkthrough. In *Proceedings of the 11th annual ACM symposium on User interface software and technology*, UIST '98, pages 173–174, New York, NY, USA, 1998. ACM.

[38] T. Igarashi, S. Matsuoka, S. Kawachiya, and H. Tanaka. Interactive beautification: a technique for rapid geometric design. In *Proceedings of the 10th annual ACM symposium on User interface software and technology*, UIST '97, pages 105–114, New York, NY, USA, 1997. ACM.

[39] T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: a sketching interface for 3D freeform design. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 409–416, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[40] Advanced Visual Systems Inc. Avs - advanced visual systems. http://www.avs.com/, (last visited on 30 October 2011), 2011.

[41] Autodesk Inc. Maya: 3D Modeling Application. http://usa.autodesk.com/maya/, (last visited on 30 October 2011), 2011.

[42] Autodesk Inc. Mudbox: 3D Sculpting and Painting Application. http://usa.autodesk.com/maya/, (last visited on 30 October 2011), 2011.

[43] Adobe Systems Incorporated. Flex: An open source framework for developing web applications. http://www.adobe.com/de/products/flex/, (last visited on 30 October 2011), 2011.

[44] IPCC. 2007: Summary for Policymakers. In *Climate Change 2007: Impacts, Adaptation and Vulnerability. Contribution of Working Group II to the Fourth Assessment Report of the Intergovernmental Panel on Climate Change*, pages 7–22. M.L. Parry, O.F. Canziani, J.P. Palutikof, P.J. van der Linden and C.E. Hanson, Eds., Cambridge University Press, UK, 2007.

[45] S. F. Judice, B. B. S. Coutinho, and G. A. Giraldi. Lattice methods for fluid animation in games. *Comput. Entertain.*, 7:56:1–56:29, January 2010.

[46] J. Kehrer, P. Filzmoser, and H. Hauser. Brushing Moments in Interactive Visual Analysis. *Computer Graphics Forum*, 29(3):813–822, June 2010.

[47] P. Kipfer and R. Westermann. Realistic and interactive simulation of rivers. In *Proceedings of Graphics Interface 2006*, GI '06, pages 41–48, Toronto, Ont., Canada, 2006. Canadian Information Processing Society.

[48] R. D. Knabb, J. R. Rhome, and D. P. Brown. Tropical Cyclone Report: Hurricane Katrina: 23-30 August 2005. National Hurricane Center, August 2006.

[49] H. Koenig. *Modern computational methods*. London: Taylor & Francis, 1998.

[50] O. Kreylos, A. M. Tesdall, B. Hamann, J. K. Hunter, and K. I. Joy. Interactive visualization and steering of CFD simulations. In *Proceedings of the symposium on Data Visualisation 2002*, VISSYM '02, pages 25–34, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[51] R. Lamb, A. Crossley, and S. Waller. *Flood Risk Management: Research and Practice*, chapter 12. Fast 2D floodplain modeling using computer game technology. CRC Press, 2009.

[52] J. A. Landay. Silk: sketching interfaces like krazy. In *Conference companion on Human factors in computing systems: common ground*, CHI '96, pages 398–399, New York, NY, USA, 1996. ACM.

[53] Q. Liang, A. G. L. Borthwick, and G. Stelling. Simulation of dam- and dyke-break hydrodynamics on dynamically adaptive quadtree grids. *International Journal for Numerical Methods in Fluids*, 46:127–162, September 2004.

[54] S. MacLean, D. Tausky, G. Labahn, E. Lank, and M. Marzouk. Is the ipad useful for sketch input?: a comparison with the tablet pc. In *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling*, SBIM '11, pages 7–14, New York, NY, USA, 2011. ACM.

[55] R. Marshall, J. Kempf, S. Dyer, and C. C. Yen. Visualization methods and simulation steering for a 3d turbulence model of lake erie. *SIGGRAPH Comput. Graph.*, 24:89–97, February 1990.

[56] K. Matkovic, D. Gracanin, M. Jelovic, and H. Hauser. Interactive Visual Steering - Rapid Visual Prototyping of a Common Rail Injection System. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1699 –1706, nov.-dec. 2008.

[57] P. Michalik, D. H. Kim, and B. D. Bruderlin. Sketch- and constraint-based design of B-spline surfaces. In *Proceedings of the seventh ACM symposium on Solid modeling and applications*, SMA '02, pages 297–304, New York, NY, USA, 2002. ACM.

[58] E. Mignot, A. Paquier, and S. Haider. Modeling floods in a dense urban area using 2d shallow water equations. *Journal of Hydrology*, 327(1-2):186–199, 2006.

[59] M. Miller, C. D. Hansen, S. G. Parker, and C. R. Johnson. Simulation steering with scirun in a distributed environment. In *High Performance Distributed Computing, 1998. Proceedings. The Seventh International Symposium on*, pages 364 –365, jul 1998.

[60] A. Modi, N. Sezer-Uzol, L. N. Long, and P. E. Plassmann. Scalable Computational Steering System for Visualization of Large-Scale CFD Simulations. In *32nd AIAA Fluid Dynamics Conference and Exhibit*, pages 24–27, 2002.

[61] J. J. Monaghan and R. A. Gingold. Smoothed particle hydrodynamics - Theory and application to non-spherical stars. *Royal Astronomical Society, Monthly Notices*, 181:375–389, Nov 1977.

[62] J. D. Mulder, J. van Wijk, and R. Van Liere. A survey of computational steering environments. *Future Generation Computer Systems*, 13, 1998.

[63] M. Müller, D. Charypar, and M. Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '03, pages 154–159, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[64] Numerical Algorithms Group (NAG). IRIS Explorer: A visual programming environment to develop, prototype and build visualization applications. `http://www.nag.co.uk/welcome_iec.asp`, (last visited on 30 October 2011), 2011.

[65] United States Navy. Army national guard ch-47 chinook helicopter drops 15,000-pound bags of sand. `http://www.navy.mil/view_single.asp?id=27953`, (last visited on 31 October 2011), 2011.

[66] A. Nealen, T. Igarashi, O. Sorkine, and M. Alexa. Fibermesh: designing freeform surfaces with 3d curves. *ACM Trans. Graph.*, 26:41–49, July 2007.

[67] A. Nealen, O. Sorkine, M. Alexa, and D. Cohen-Or. A sketch-based interface for detail-preserving mesh editing. *ACM Trans. Graph.*, 24:1142–1147, July 2005.

[68] MSNBC.com news service. Efforts to revive New Orleans get back in gear. `http://www.msnbc.msn.com/id/9438536/ns/us_news-katrina_the_long_road_back/t/efforts-revive-new-orleans-get-back-gear/`, (last visited on 31 October 2011), 2011.

[69] J.J. Nicolas, K.E. Gubbins, W.B. Streett, and D.J. Tildesley. Equation of state for the Lennard-Jones fluid. *Molecular Physics*, 37(5):1429–1454, 1979.

[70] L. Nyland, M. Harris, and J. Prins. *GPU Gems 3*, chapter Chapter 31. Fast N-Body Simulation with CUDA. Addison-Wesley, 2008.

[71] L. Olsen, F. F. Samavati, M. C. Sousa, and J. Jorge. Sketch-based mesh augmentation. In *Proceedings of the Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pages 43–52, Dublin, Ireland, 2005. Eurographics Association.

[72] L. Olsen, F. F. Samavati, M. C. Sousa, and J. A. Jorge. Sketch-based modeling: A survey. *Computers & Graphics*, 33(1):85–103, February 2009.

[73] S. G. Parker and C. R. Johnson. SCIRun: A Scientific Programming Environment for Computational Steering. In *Supercomputing, 1995. Proceedings of the IEEE/ACM SC95 Conference*, page 52, 1995.

[74] L. Piegl. Interactive Data Interpolation by Rational Bezier Curves. *IEEE Comput. Graph. Appl.*, 7:45–58, April 1987.

[75] A. Pihuit, M. P. Cani, and O. Palombi. Sketch-based modeling of vascular systems: a first step towards interactive teaching of anatomy. In *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium*, SBIM '10, pages 151–158, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[76] Pittsburgh Post-Gazette. Bush returns to gulf. `http://www.msnbc.msn.com/id/9438536/ns/us_news-katrina_the_long_road_back/t/efforts-revive-new-orleans-get-back-gear/`, (last visited on 31 October 2011), 2011.

[77] D. A. Randall. The Shallow Water Equations. Selected papers, Department of Atmospheric Science, July 2006.

[78] H. Ribičić. Comparative Rendering of Simulation Scenarios. Master's thesis, University of Zagreb, Faculty of Electrical Engineering and Computing, September 2010.

[79] A. M. A. Sattar, A. A. Kassem, and M. H. Chaudhry. Case study: 17th street canal breach closure procedures. *Journal of Hydraulic Engineering*, 134(11):1547–1558, 2008.

[80] B. Schindler, J. Waser, R. Fuchs, and R. Peikert. Multiverse data-flow control. Technical Report 720, ETH Zürich, Computational Science, February 2011.

[81] D. Schroeder, D. Coffey, and D. Keefe. Drawing with the flow: a sketch-based interface for illustrative visualization of 2d vector fields. In *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium*, SBIM '10, pages 49–56, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[82] H. J. Shin and T. Igarashi. Magic canvas: interactive design of a 3-d scene prototype from freehand sketches. In *Proceedings of Graphics Interface 2007*, GI '07, pages 63–70, New York, NY, USA, 2007. ACM.

[83] L. Song, J. Zhou, Q. Zou, J. Guo, and Y. Liu. Two-Dimensional Dam-Break Flood Simulation on Unstructured Meshes. In *Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2010 International Conference on*, pages 465–469, dec. 2010.

[84] S. Succi. *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond*. Oxford University Press: Oxford, 2001.

[85] G. Taubin. Curve and surface smoothing without shrinkage. In *Computer Vision, 1995. Proceedings., Fifth International Conference on*, pages 852 –857, jun 1995.

[86] E. F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics, A practical Introduction, 2. ed.* Springer, 1999.

[87] M. Tritthart and D. Gutknecht. 3-D computation of flood processes in sharp river bends. In *Proceedings of the ICE - Water Management*, volume 160, pages 233–247, December 2007.

[88] E. Üstündag and M. S. Celebi. A B-Spline Curve Fitting Approach by Implementing the Parameter Correction Terms. In *International Conference on Computational Science and Engineering*, Istanbul, Turkey, June 27-30 2005.

[89] A. van Dam. Post-wimp user interfaces. *Commun. ACM*, 40:63–67, February 1997.

[90] R. van Liere, J. D. Mulder, and J. J. Van Wijk. Computational Steering. In *Future Generation Computer Systems*, volume 12, pages 441–450, Elsevier North Holland, 1997.

[91] J. Vetter and K. Schwan. High performance computational steering of physical simulations. In *Parallel Processing Symposium, 1997. Proceedings., 11th International*, pages 128 –132, apr 1997.

[92] J. Waser, R. Fuchs, H. Ribičić, and G. Blöschl. Visuelle Aktionsplanung im Hochwassermanagement. In *Forum für Hydrologie und Wasserbewirtschaftung*, volume 30, pages 280–286. 2011.

[93] J. Waser, R. Fuchs, H. Ribičić, B. Schindler, G. Blöschl, and E. Gröller. World Lines. *IEEE Transactions on visualization and Computer Graphics*, VOL. 16, NO. 6, November/December 2010:1458–1467, November 2010.

[94] J. Waser, H. Ribičić, R. Fuchs, C. Hirsch, B. Schindler, G. Blöschl, and E. Gröller. Nodes on Ropes: A Comprehensive Data and Control Flow for Steering Ensemble Simulations. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1872–1881, December 2011.

[95] D. Weiskopf and G. Erlebacher. Overview of flow visualization. In C. D. Hansen and C. R. Johnson, editors, *The Handbook of Visualization*, pages 261–278. Elsevier, Amsterdam, 2005.

[96] J. F. Wendt, editor. *Computational Fluid Dynamics - An Introduction*. Springer, third edition, 2009.

[97] W. Wenli, Z. Pei, and G. Sheng. Numerical Simulation of 2D Flood Waves Using Shallow Water Equations. In *Power and Energy Engineering Conference (APPEEC), 2010 Asia-Pacific*, pages 1–3, March 2010.

[98] E. Wiese, J. H. Israel, A. Meyer, and S. Bongartz. Investigating the learnability of immersive free-hand sketching. In *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium*, SBIM '10, pages 135–142, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[99] H. Wright, K. Brodlie, and M. Brown. The dataflow visualization pipeline as a problem solving environment. In *Proceedings of the Eurographics workshop on Virtual environments and scientific visualization '96*, pages 267–276, London, UK, 1996. Springer.

[100] R. C. Zeleznik, K. P. Herndon, and J. F. Hughes. SKETCH: An Interface for Sketching 3D Scenes. In *SIGGRAPH '96: Proceedings of the 23rd annual con-*

ference on Computer graphics and interactive techniques, pages 163–170, New York, NY, USA, 1996. ACM.

[101] X. Zhao, X. Zhang, T. Chi, H. Chen, and Y. Miao. Design and implementation of a web-based flood simulation and decision support system. In *Proceedings of the 6th Conference on WSEAS International Conference on Applied Computer Science - Volume 6*, pages 59–64, Stevens Point, Wisconsin, USA, 2007. World Scientific and Engineering Academy and Society (WSEAS).