

STUTTGART MEDIA UNIVERSITY

BACHELOR THESIS

---

# Simulation of destruction effects in a production environment

---



*Author:*  
Manuel CARULLO

*First Examiner:*  
Prof. Katja SCHMID  
*Second Examiner:*  
Prof. Dr. Thomas KEPPLER

*submitted in partial fulfillment  
of the requirements  
for the degree of  
Bachelor of Engineering  
at*

Visual Effects  
Audiovisual Media

21. August 2017

# Statutory declaration

Hiermit versichere ich, Manuel CARULLO, ehrenwörtlich, dass ich die vorliegende Bachelorarbeit mit dem Titel: "Simulation of destruction effects in a production environment" selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Ich habe die Bedeutung der ehrenwörtlichen Versicherung und die prüfungsrechtlichen Folgen (§26 Abs. 2 Bachelor-SPO (6 Semester), § 24 Abs. 2 Bachelor-SPO (7 Semester), § 23 Abs. 2 Master-SPO (3 Semester) bzw. § 19 Abs. 2 Master-SPO (4 Semester und berufsbegleitend) der HdM) einer unrichtigen oder unvollständigen ehrenwörtlichen Versicherung zur Kenntnis genommen.

Signature:

---

Date:

---

## Abstract

### *Simulation of destruction effects in a production environment*

Simulations are becoming increasingly common as an element of visual effects in motion pictures. Especially destruction effects belong to the daily tasks of every effects artist. This thesis will give an introduction to the different relevant simulation types used to build up destruction effects including a quick overview of the calculations and algorithms behind their creation. Afterwards the exact techniques used to create them particularly in Houdini will be presented using the practical example of the work done for the student production "The Girl Beyond". Possible issues and common difficulties will be discussed and solutions as well as established best practices proposed. The thesis moves on to a review of the example result, recounting the biggest difficulties encountered during the production and outlining possible improvements. In conclusion, this thesis offers a comprehensive overview over the subject of visual effects simulations, particularly destruction fx, and how best to approach them in a production environment.

## Kurzfassung

### *Simulation von Zerstörungseffekten im Rahmen einer Studioproduktion*

Simulationen stellen einen immer größer werdenden Anteil an Visual Effects in Filmen dar. Besonders Zerstörungseffekte gehören zum Arbeitsalltag eines jeden fx artists. Diese Thesis soll eine Einführung in die verschiedenen, für Zerstörungseffekte relevanten Arten von Simulationen geben und die physikalischen und mathematischen Hintergründe ihrer Berechnung erläutern. Danach werden die genauen Verfahren, die genutzt werden, um Zerstörungseffekte insbesondere in Houdini umzusetzen am praktischen Beispiel der fx der Studioproduktion "The Girl Beyond" vorgestellt. Mögliche Probleme und häufige Schwierigkeiten, so wie deren Lösung und generell bewährte Methoden werden erläutert. In Folge dessen wird das Endergebnis des Beispiels evaluiert, bestehende Probleme aufgezählt und mögliche Verbesserungsansätze geäußert. Im Allgemeinen bietet diese Thesis einen umfassenden Überblick über Visual Effects Simulationen, insbesondere Zerstörungseffekte, und erläutert die Vorgehensweise bei der Erstellung dieser im Rahmen einer Produktion.

# Contents

<b>Statutory Declaration</b>	<b>i</b>
<b>Abstract</b>	<b>ii</b>
<b>Kurzfassung</b>	<b>ii</b>
<b>Table of contents</b>	<b>iii</b>
<b>List of figures</b>	<b>v</b>
<b>Abbreviations</b>	<b>vi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	2
1.2 Document structure . . . . .	3
<b>2 Simulation in visual effetics</b>	<b>4</b>
2.1 Simulation classification . . . . .	5
<b>3 Theory &amp; algorithms</b>	<b>8</b>
3.1 General concepts . . . . .	8
3.2 Particle systems . . . . .	10
3.2.1 Calculating particle systems . . . . .	11
3.2.2 Particle collision . . . . .	11
3.3 Rigid body dynamics . . . . .	12
3.3.1 Unconstrained motion . . . . .	15
3.3.2 Constrained motion . . . . .	16
3.3.2.1 Collision detection . . . . .	16
3.3.2.2 Collision response . . . . .	17
3.4 Fluid dynamics . . . . .	19
3.4.1 Model overview . . . . .	20
3.4.2 Calculating fluids . . . . .	21
<b>4 Production environment</b>	<b>24</b>
4.1 "The Girl Beyond" . . . . .	24
4.2 Shot description . . . . .	25
<b>5 Software &amp; pipeline</b>	<b>27</b>



5.1	SideFX Houdini . . . . .	27
5.1.1	Geometry context . . . . .	28
5.1.2	Dynamics context . . . . .	29
5.1.2.1	POPs / particles . . . . .	30
5.1.2.2	RBD & Bullet . . . . .	31
5.1.2.3	Pyro solver . . . . .	31
5.2	Alembic workflow . . . . .	32
5.2.1	Alembic in Houdini . . . . .	33
<b>6</b>	<b>Implementation</b>	<b>35</b>
6.1	Planning phase . . . . .	35
6.1.1	Previsualization . . . . .	38
6.1.2	Test shoots . . . . .	38
6.1.3	Research & development . . . . .	40
6.2	Scene setup . . . . .	41
6.2.1	Look references . . . . .	42
6.2.2	Modeling . . . . .	43
6.2.3	Fracturing . . . . .	45
6.3	Simulation . . . . .	48
6.3.1	Divide & Conquer . . . . .	48
6.3.2	Rigid Bodies . . . . .	49
6.3.2.1	Forces . . . . .	50
6.3.2.2	Constraints . . . . .	50
6.3.2.3	Replacing with high quality assets . . . . .	51
6.3.2.4	Transfer to maya . . . . .	52
6.3.3	Particle dust . . . . .	53
6.3.3.1	General behavior . . . . .	53
6.3.3.2	Particle source . . . . .	54
6.3.3.3	Forces . . . . .	54
6.3.3.4	Post processing . . . . .	55
6.3.4	Smoke dust . . . . .	55
6.3.4.1	Smoke source . . . . .	55
6.3.4.2	Microsolvers & general behavior . . . . .	57
6.3.4.3	Forces & collision . . . . .	57
6.3.4.4	Smoke post processing . . . . .	58
6.3.5	Optimization & cleanup . . . . .	59
<b>7</b>	<b>Evaluation</b>	<b>61</b>
7.1	Shot review . . . . .	61
<b>8</b>	<b>Conclusion</b>	<b>64</b>
	<b>Bibliography</b>	<b>66</b>

# List of Figures

1.1	Examples of complex destruction effects in recent movies . . . . .	2
2.1	Example for unrealistic, but believable simulations . . . . .	5
2.2	Example: Particles . . . . .	6
2.3	Example: Rigid body dynamics . . . . .	6
2.4	Example: Fluid dynamics . . . . .	7
3.1	Explicit Euler method . . . . .	9
3.2	Body space to world space transformation . . . . .	13
3.3	RBD update cycle . . . . .	16
3.4	Different intersection types . . . . .	17
3.5	Illustration of a binary search tree for a time step. . . . .	18
3.6	Collision response impulse calculation . . . . .	19
5.1	Example: Houdini scene . . . . .	28
5.2	Geometry spreadsheet . . . . .	29
6.1	Storyboard for UEB010 . . . . .	37
6.2	Previs for UEB010 . . . . .	38
6.3	First tests in the set . . . . .	39
6.4	Simulation references for UEB010: plaster and dust . . . . .	42
6.5	Brick wall and reference set . . . . .	44
6.6	Packed Primitives . . . . .	44
6.7	Voronoi diagramm . . . . .	46
6.8	Fractured plaster and bricks . . . . .	47
6.9	Plaster constraints . . . . .	51
6.10	Exchanging sim proxies with HQ geo . . . . .	52
6.11	Result: particles . . . . .	55
6.12	Smoke collision geometry . . . . .	58
7.1	UEB010 Breakdown . . . . .	61

# Abbreviations

<b>Comp</b>	<b>Com</b> positing
<b>CPU</b>	<b>C</b> entral <b>P</b> rocessing <b>U</b> nit
<b>DOP</b>	<b>D</b> ynamics <b>O</b> perator
<b>DoP</b>	<b>D</b> irector of <b>P</b> hotography
<b>HQ</b>	<b>H</b> igh <b>Q</b> uality
<b>Id</b>	<b>I</b> dentifier
<b>FX</b>	<b>E</b> ffects, meaning simulations in visual effects
<b>Previs</b>	<b>P</b> revisualisation
<b>POP</b>	<b>P</b> article <b>O</b> perator
<b>Post</b>	<b>P</b> ostproduction
<b>RAM</b>	<b>R</b> andom <b>A</b> ccess <b>M</b> emory
<b>Res</b>	<b>R</b> esolution
<b>RBD</b>	<b>R</b> igid <b>B</b> ody <b>D</b> ynamics
<b>R&amp;D</b>	<b>R</b> esearch and <b>D</b> evelopment
<b>SOP</b>	<b>S</b> urface <b>O</b> perator
<b>Sim</b>	<b>S</b> imulation
<b>VFX</b>	<b>V</b> isual <b>E</b> ffects

# Chapter 1

## Introduction

Since the early beginnings of film making, visual effects have played an important and integral part in creating exciting, visually interesting movies and their importance has only been growing since then.

But especially over the last few decades the rise of visual effects has accelerated to new heights and their presence in cinema, tv and other visual media is stronger than ever. Nowadays almost every movie relies heavily on visual effects, even the ones you wouldn't expect to. For a lot of blockbusters, they are one of the main selling points as can be seen by looking at the top grossing movies over the last few decades [[Box Office Mojo, 2017](#)].

Reasons for this sudden rise are cost efficiency, more impressive visuals and more controllability, especially compared to expensive and complex shoots. For example many destruction effects wouldn't even be possible to be shot, without causing extreme damage to buildings or harm people [[Stam, 2015](#), p.2]. At the same time computer performance is growing rapidly and the techniques and algorithms behind vfx become more advanced and sophisticated by the minute, making the ultimate goal of vfx, photorealism, more achievable than ever.

An area of visual effects where that progress is especially noticeable is simulation, or so called fx. In the history of film making and visual effects fx is a relatively young discipline, first being used about three decades ago for a planet engulfed by fire in "Star Trek II - The Wrath of Khan" [[Reeves, 1983](#)]. The discipline has come a long way in a short time and it is now possible to simulate a wide range of effects in a realistic way with one of the probably most common ones being destruction effects, meaning destroying buildings, vehicles, environments etc. through external forces(e.g. explosions or crashes).



(a) Destruction of a whole city by an interplanetary weapon in *Star Wars: Rogue One*  
©Lucasfilm Ltd.

(b) Destruction of a mansion by missiles in *Iron Man 3*  
©Marvel Studios, LLC

FIGURE 1.1: Examples of complex destruction effects in recent movies

## 1.1 Motivation

Destruction fx and simulations in visual effects are generally complex and hard to grasp. They are based on real-world physics, and calculated using approximated mathematics, influenced by real-world forces like gravity or friction or completely unrealistic artist-defined forces. Technically an fx artist won't have to calculate simulations himself, but instead setup the software correctly to calculate the simulation the artists wants. Nevertheless, a general knowledge about the mathematical background and the inner workings of simulation algorithms form a good base for implementing high quality fx.

At the same time, there is not one defined or correct approach to create destruction fx. For beginners and inexperienced 3d artists this leads to a lot of trial & error and slow progress, especially when using software with such a steep learning curve as Houdini.

Matching real world reference footage, high quality destruction fx thrive on their complexity and are often created from multiple simulation passes and various simulation types. Additionally, there are a lot of workflow and optimization considerations. Depending on the setup two very similar looking simulations can differ drastically in calculation time, storage needed and rendering times.

Although simulations are intended and implemented to look and behave like reality, they still need to be completely art directable. Often real world physics don't look visually interesting enough or a certain unrealistic behavior is desired by the director. This adds creative requirements to the already existing technical challenges and makes for an even more complex task.

The goal for this thesis is to offer an overview over destruction simulations in visual effects, explain the theory behind them and show an optimal approach in implementing a destruction simulation based on a production example of a shot in the short movie "The Girl Beyond".

## 1.2 Document structure

First an overview over simulations in visual effects will be given in Chapter 2, including a brief classification of the different simulation types. Chapter 3 then introduces the basic physics and mathematics behind the common algorithms of destruction simulations.

In Chapter 4 the production environment and the actual shot that will be used as an applied implementation example is described in detail. Chapter 5 elaborates on the visual effects pipeline of the production and the specific software used for the implementation in Chapter 6. This includes an introduction to Houdini and an overview over Alembic, the interchange format used to get the simulation from Houdini into Maya for rendering.

Finally, the general approach of implementing a destruction simulation will be described and broken down step by step in Chapter 6 based on the example of a shot in "The Girl Beyond" taking up the algorithms, simulation types and software described in Chapter 3 and 5.

At last Chapter 7 is a quick presentation of the final result and an evaluation of the biggest difficulties faced while working on it, followed by the conclusion in Chapter 8.

## Chapter 2

# Simulation in visual effects

When talking about simulations in visual effect, what is meant for the purpose of this thesis is physically based animation. Physically based animations are mainly used for natural phenomena or effects where a multitude of objects move equally but uniquely, like fluids(liquids and gases), dust, explosions, crowds and many more [Gress, 2014, p.389]. These effects would be incredibly hard or almost impossible to create with traditional keyframed animation on a comparable complexity.

The major difference to the classic way of animation is the level and type of control the artists has over the actual animation. In classic animation the artist has very specific expectations about the motion that will be produced on a frame-by-frame basis [Parent, 2012, p.111]. For every frame or a range of frames precise start and end values for positions, scale or rotation can be set, interpolation algorithms can be chosen to fill in missing values between keyframes and everything can be refined and broken down into smaller steps until there is complete control by the artist over every parameter in every single frame of the scene.

Although total control over every movement in your scene sounds desirable, there are also some downsides to it. Controlling the animation on such a low level can be a very time consuming and exhausting process. Even when many values can be interpolated and controlled by keyframing only a subset of the framerate, it can take a lot of time and iterations until something looks the way the artist wants it to. Then there's still no guarantee for a realistic, natural motion, since the animation was created artificially by the artist, trying to match references or maybe even through pure imagination. Multiplying this process times thousand or a million, for every single shard in a breaking window, a grain in a sandstorm or a water drop in a river and it would never be possible to create those effects in a reasonable amount of time.

The solution for creating realistic effects containing these complicated subjects in computer animation is trying to approximate the real world by calculating the underlying

physics behind their movements [Stam, 2015, p.7]. So the general approach for physically based animation is to animate on a much higher level. Instead of the direct movement the artists controls the general quality of the motion through setting physical properties, forces and constraints affecting the objects.

Important for simulations is that forces don't necessarily have to be realistic, but instead the focus lies on the believability or being so called "physically realistic" [Parent, 2012, p.199]. In fact in most cases actually realistic simulations wouldn't be able to create the look wanted by the artist or director. That's why art directability is an important requirement of simulations for visual effects [Stam, 2015, p.2]. The most common approach for simulations in vfx is to start off with a physically accurate, stable setup and then step by step introduce custom forces, constraints and object properties to art direct the shot to the desired look while keeping an eye on believability.



FIGURE 2.1: Fluid effect from Lord of the Rings: The Fellowship of the Ring -  
Example for unrealistic, but believable effects  
©New Line Cinema Productions Inc.

## 2.1 Simulation classification

Simulations in visual effects can be classified into various different categories.

The most basic type of simulations are probably particle systems. Particle systems can best be described as a collection of a high quantity of points with attributes [Parent, 2012, p.205f]. They are only animated with basic physical simulations and depending on the software implementation make various simplifying assumptions regarding their behavior. Therefore, they don't have high computational costs. Because of this, particles can be simulated up to a really high count of millions of points in a reasonable amount of time [Gress, 2014, p.390]. At the same time because their behavior and movements are not as realistic as more sophisticated simulations, nowadays they are mainly used for secondary effects with a different type of hero sim more in focus.

A more advanced type of simulations is rigid body dynamics, also called RBDs. Rigid bodies are used to create realistic-looking motion based on the behavior of non-deformable



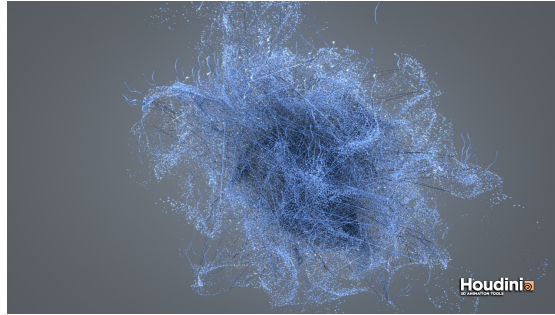


FIGURE 2.2: Example: Particles

objects when affected by common forces like gravity or friction and colliding with each other or different objects [Parent, 2012, p.209]. This type of motion can be calculated efficiently with basic motion equations up to a sizable amount of objects.

RBDs are essential for destruction fx, as breaking buildings, cars or in general everything that can be destroyed and fractured into multiple pieces. Depending on the software there are various different implementations with varying features for rigid body dynamics. The most common one in visual effects is probably the Bullet Physics Library.



FIGURE 2.3: Example: Rigid body dynamics

Presumably one of the most advanced types of simulation in vfx and the one currently still the most researched on is fluid dynamics. Fluids can be described as any material that changes its form in an ongoing manner [Stam, 2015, p.2] or to be more precise: gases and liquids [Parent, 2012, p.270]. Most fluids can be found in nature for example water, fire, clouds or smoke. There are various different models and approaches to simulate fluids depending on the type and the software used. A more detailed and in-depth classification and explanation can be found in Chapter 3. As fluid dynamics and their solution/approximation in computer graphics are still cutting edge technology, a lot of progress has been made regarding the look and performance of fluid simulations over the last few years.

There is a wide range of other simulation categories, that will not be discussed in detail as they are not directly relevant for destruction fx. Just to mention a few of the most important ones there's also character/creature fx which includes secondary animation



FIGURE 2.4: Example: Fluid dynamics

for characters and creatures like hair, feathers or muscles under the skin. Another commonly used one is crowd fx. A technique to simulate medium to huge groups of people, creatures or anything else that is living and moving, and behaving in a believable way [Okun, 2010, p.637].

## Chapter 3

# Theory & algorithms

To get a better understanding for simulations used in visual effects and their different types this chapter will introduce some of the underlying algorithms and calculations.

### 3.1 General concepts

The fundamental equation behind all simulations presented in this chapter is Newtons second law of motion,

$$\vec{f} = m \cdot \vec{a}$$

stating the force  $f$  affecting an object is equal to its mass  $m$  times its acceleration  $a$ . Since in visual effects simulations the sought results usually include position and velocity which in turn can be calculated from the acceleration the more useful term would be

$$\vec{a} = \frac{\vec{f}}{m}$$

The force in this case can be the sum of any kind of external forces, like gravity, wind forces, or custom forces defined by the artist, while the more advanced simulations would also consider contact/collision forces and friction among other [Coumans, 2015].

With known initial values for the position  $p$  and the velocity  $v$ , which are incidentally also set by the artist, the new position  $p(\Delta t)$  and velocity  $v(\Delta t)$  after the time  $\Delta t$  (and under the assumption of constant acceleration) can now be calculated:

$$\vec{v}(\Delta t) = \vec{v} + \vec{a}\Delta t$$

$$\vec{p}(\Delta t) = \vec{p} + \frac{1}{2}(\vec{v} + \vec{v}')\Delta t$$

Keep in mind that all of these equations are vector-valued, as force, acceleration, velocity and position are all vectors [Parent, 2012, p.200].

Opposite to these equations application in regular physics, computer animation is mainly concerned with modeling motion respective to discrete time steps. Those can be set to any length, are usually constant over the course of a simulation and have a distinct impact on the result of the equations and its error.

As in most simulations forces and therefore acceleration and velocity are dependent on time as well as position, often non-linear variations occur over the course of a timestep. In this case calculating the equations with the initial values sampled at the beginning of the time step, using the so-called explicit Euler method, leads to considerable deviations from the ideal continuous path. The introduced error is proportional to the step size and can be reduced by minimizing the timestep, which in turn leads to excessive simulation times [Parent, 2012, p.211f].

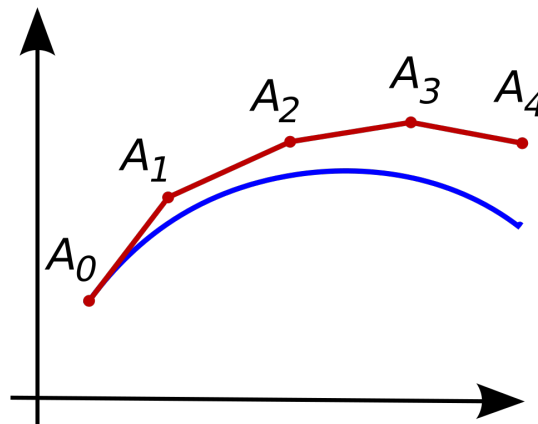


FIGURE 3.1: ideal continuous path(blue) and explicit Euler approximation(red), significant deviation after few long steps<sup>1</sup>

Although computer animation isn't deeply concerned with scientific correctness or numerical accuracy it is possible for visual realism to be compromised by sloppy numeric calculations. To battle this problem there is a wide range of more sophisticated numerical integration techniques available, which help achieving more accurate results with a reasonable amount of substeps.

A more accurate method which delivers stable results even on larger time steps is the implicit Euler. Instead of sampling values at the beginning of the time step, the implicit Euler finds a new position using techniques like the Newton method to update the current value with the derivatives at the new position.

<sup>1</sup> source: [https://commons.wikimedia.org/wiki/File:Euler\\_method.svg](https://commons.wikimedia.org/wiki/File:Euler_method.svg) - access date: 2017-08-19

$$\vec{v}(t + 1) = \vec{v}(t) + \vec{a}(t + 1)\Delta t$$

$$\vec{p}(t + 1) = \vec{p}(t) + \vec{v}(t + 1)\Delta t$$

Because for every step an equation needs to be solved to find a fitting new point, this takes a longer time to compute. Nevertheless it can be more efficient as explicit Euler because bigger time steps can be chosen and therefore less computation is required to fulfill the same numeric accuracy.

Combining the implicit and explicit Euler leads to the semi-implicit or symplectic Euler method. This method strikes the middle between both previous methods having more accuracy than the explicit with much less computational complexity than the implicit Euler. In this case the explicit method is used in evaluating the velocity calculation and the result then used to calculate the position.

$$\vec{v}(t + 1) = \vec{v}(t) + \vec{a}(t)\Delta t$$

$$\vec{p}(t + 1) = \vec{p}(t) + \vec{v}(t + 1)\Delta t$$

Based on its compromise between accuracy and efficiency the semi-implicit Euler method is probably the most popular numerical integration technique in simulations for vfx.

There are a multitude of more advanced algorithms used for solving these initial value problems in physics and other scientific environments. Because of diminishing returns they are hardly used for visual effects for now, if then mostly in more advanced fluid simulations. But as computational capacity and requirements for accuracy increase, so does the possibility of using higher order Runge-Kutta and other more advanced methods [Coumans, 2015].

## 3.2 Particle systems

Particles systems are one of the simplest type of simulations in computer animation. They are basic objects or rather points with a position, velocity and mass, which can respond to applied forces and in their most basic form have no spatial extent or size on their own [Witkin, 2001].

As particle systems by their nature are very procedural and many elements from generation over behavior to shading & rendering are based on simple stochastic processes and random values they allow for detailed, alive models with very little artist design time.

Often parameters regarding appearance and movement can be set by providing a mean value and specifying a maximum deviation [Reeves, 1983].

Common attributes, beside the essential position, velocity and mass include color, transparency, id, and lifetime. In addition any amount of attributes can be added to control custom behaviors.

Lifespan in particular is an important attribute, which controls how long a particle is part of the system. It is generally set at birth and decreased in every simulation step. When the attribute reaches zero, the particle is terminated and removed from the system. In Houdini this functionality is modeled over the two attributes age and life, which state the current age and the total lifespan in seconds. Often these values are used to change the appearance and shading parameters of particles over the course of their life [Parent, 2012, p.207].

### 3.2.1 Calculating particle systems

The fundamental algorithm for each step in a particle simulation can be described as following:

- Generating new particles, born during the timestep
- Generating and assigning attributes for born particles
- Terminating particles exceeding their given lifespan
- Updating remaining particles attributes, including their calculated motion
- (Rendering the particles)

The actual method used to calculate the motion of the particles is interchangeable, but since the need for fast performance with a multitude of objects outweighs the accuracy of individual particle motions, most often a basic Euler method as presented in the previous section is used.

### 3.2.2 Particle collision

In most implementations sophisticated collision between particles is not intended. Nevertheless collision with other geometry or a ground plane is often required. Calculating collisions can be broken down into two steps: Detecting, if and when a collision occurs and calculating the appropriate response.

Collision detection between a particle and a plane is fairly simple. For a point P on the plane, the normal N pointing towards the side with particles and the particle X, the

collision can be detected by evaluating  $(\vec{X} - \vec{P}) \cdot \vec{N}$  and checking the sign. A positive value means the particle is in front of the plane, zero means contact and below zero indicates that a collision has occurred. This needs to be checked after every time step [Witkin, 2001].

If collision has been detected at the end of a timestep there are two possible options to determine the actual time of collision. The first would be assuming the position at the end of the time step as valid and calculating the respective appropriate reaction. This allows penetration and in case of high velocity or multiple occurring collisions in a single timestep can lead to visually significantly different results. The advantage is a relatively easy implementation with mostly acceptable results. The second approach would mean backing up time until the exact moment of collision is found and calculating the appropriate response based on this moment. In case of multiple collisions time is backed up to the moment the first collision happened. In complex simulations with frequent collisions the time spent backing up can have a tremendous impact on simulation time. [Parent, 2012, p.220f]

For particle collision most often the first option is used to calculate a simple kinematic response by negating the normal component of the velocity. Shown based on the previously mentioned particle - plane example the first step is decomposing the velocity vector to its normal and tangential component:  $\vec{v}_n = (\vec{N} \cdot \vec{v})\vec{N}$  and  $\vec{v}(t) = \vec{v} - \vec{v}_n$ . Then the normal component is inverted and if inelastic or damped collision is required multiplied by a damping factor,  $0 < k < 1$ . This leads to the following equation for calculating the particles velocity after collision:

$$\vec{v}(t+1) = \vec{v}(t) - (1+k)(\vec{v}(t) \cdot \vec{N})\vec{N}$$

### 3.3 Rigid body dynamics

Rigid bodies are based on the same underlying principles introduced for particles in the previous sections. But to fully model and simulate rigid body dynamics some additions have to be made.

While the location of a particle in space can simply be described as its translation from the origin, for 3 dimensional objects like rigid bodies the rotation has to be considered as well. This is generally done using a 3 x 3 rotation matrix called  $R(t)$ . The spatial variables of a rigid body can therefore be stated as  $\mathbf{x}(t)$  and  $R(t)$ , saving the position and rotation of an object in world space.

Opposite to particles, rigid bodies also occupy a volume and a certain shape with a theoretical unlimited amount of individual points, but  $x(t)$  and  $R(t)$  only define transformation and rotation per object. Therefore a separate, fixed and unchanging space called body space is used to define the shape of a rigid body. The contents of the body space can then be transformed through our position and orientation value to correctly represent the body in world space. This is based on the assumption that the origin in body space corresponds with the actual center of mass of the rigid body [Baraff, 2001]. This way the world space position  $p(t)$  of any point  $p(0)$  on the rigid body in body space can be calculated by rotating it around the origin and a subsequent transform.

$$p(t) = R(t) \cdot p(0) + x(t)$$

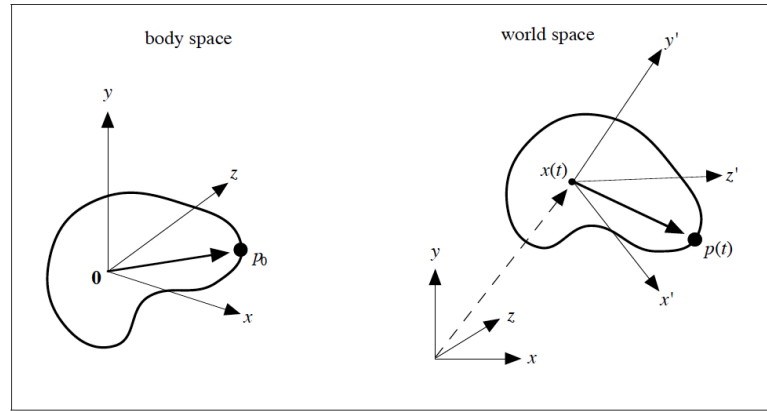


FIGURE 3.2: Body space to world space transformation<sup>2</sup>

Having an orientation leads parallel to the linear attributes position  $x(t)$ , velocity  $v(t)$  and acceleration  $a(t)$  to the existence of angular velocity and angular acceleration. The angular velocity is usually defined by the vector  $\omega(t)$  and describes the spin of an object irrespective of its linear velocity. The direction of  $\omega(t)$  gives the direction of the axis, the body is spinning and the magnitude states how fast it is spinning measured in revolutions per time [Parent, 2012, p.214f].

Because many of the calculations to simulate rigid bodies depend on mass it's necessary to calculate the total mass of each body. This is done by modeling the volume of the body as a finite amount of points, for example its vertices, with individual mass values  $m_i$  attached to them. The overall amount of mass  $M$  can then be calculated with

$$M = \sum m_i$$

The previously presented second law of Newton holds true for rigid bodies as well. To calculate the linear acceleration the overall force  $F(t)$  on the body needs to be calculated

<sup>2</sup> source: [Baraff, 2001]



by summing up all individual forces  $f_i(t)$  affecting individual points on the object, taking their individual position into consideration.

$$F(t) = \sum f_i(t)$$

Similarly the torque, the rotational equivalent of linear force, is calculated, with the difference that torque depends on the location of its application point  $p_i(t)$  relative to the center of mass  $x(t)$ . The total torque  $\tau$  is therefore calculated as

$$\tau(t) = \sum \tau_i(t) = \sum (p_i(t) - x(t)) \cdot f_i(t)$$

While  $F(t)$  contains no information about the locations of the individual forces acting on the body,  $\tau(t)$  contains information about the distribution of forces [Baraff, 2001].

The momentum (mass · velocity) is an important variable in RBDs as is it conserved in a closed system. This means the sum of momentum doesn't change if there are no outside influences affecting the system and can therefore be used to solve for unknown values like velocity. As for previous variables it is divided into linear and angular momentum. The overall linear momentum  $P(t)$  is calculated by the sum of the momentum of all points on the object. Because the center of mass lies in the origin of our body space the equations is as simple as

$$P(t) = \left( \sum m_i \right) v(t) = Mv(t)$$

Because the mass is constant over the course of the simulation a relationship can be built between linear momentum and linear force, stating that an acting force on a body is equal to it's change in momentum [Parent, 2012, p.217].

The concept of angular momentum is quite complex to understand and the variable is solely introduced because of its conserved nature and its resulting simplification of equations. It allows the same approach of calculations as used for the linear attributes and is therefore helpful in creating a unified system. Parallel to the linear momentum  $P$  the angular momentum  $L$  is defined as

$$L(t) = I(t)\omega(t)$$

where  $I(t)$  is the so called inertia tensor, a  $3 \times 3$  matrix, which describes the distribution of mass in a body relative to its center.

Since the inertia tensor is dependent on the orientation  $R(t)$  it would have to be recalculated for each change in  $R(t)$ . Because the calculation involves several complex integrals, this would be highly inefficient during simulation. One solution found is to calculate the initial inertia tensor  $I_{body}$  once in the static body space and then simply

use the current orientation matrix  $R(t)$  to calculate the inertia tensor  $I(t)$  needed at the time Baraff [2001]. This leads to the following equations [Parent, 2012, p.218]:

$$I_{body} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{yz} \\ I_{xz} & I_{yz} & I_{zz} \end{bmatrix}$$

$$I_{xx} = \sum m_i(y_i^2 + z_i^2) \quad I_{xy} = \sum m_i x_i y_i$$

$$I_{yy} = \sum m_i(x_i^2 + z_i^2) \quad I_{xz} = \sum m_i x_i z_i$$

$$I_{zz} = \sum m_i(x_i^2 + y_i^2) \quad I_{yz} = \sum m_i y_i z_i$$

$$I(t) = R(t)I_{body}R(t)^T$$

### 3.3.1 Unconstrained motion

After preparing all those attributes it is now possible to build a so called state vector  $S(t)$ , which describes the state of every rigid body in the system at any time, including the main attributes position, orientation, linear momentum and angular momentum.

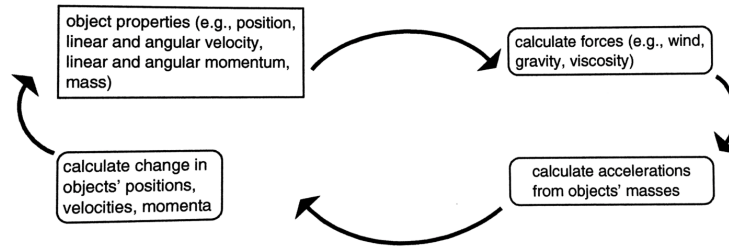
$$S(t) = \begin{bmatrix} x(t) \\ r(t) \\ P(t) \\ L(t) \end{bmatrix}$$

At the beginning of the simulation the constant attributes mass  $M$  and initial inertia tensor  $I_{body}$  are calculated. In addition it is now possible to calculate the auxiliary attributes  $I(t) = R(t)I_{body}R(t)^T$ ,  $\omega(t) = I(t)^{-1}L(t)$ , and  $v(t) = \frac{P(t)}{M}$  at any given time.

Now the derivative of the state vector with respect to time can be build:

$$\frac{d}{dt}S(t) = \begin{bmatrix} v(t) \\ \omega(t) * R(t) \\ F(t) \\ \tau(t) \end{bmatrix}$$

With all these attributes and equations and the possibility to calculate any derivative information available it is now possible to run the rigid body simulation update cycle.

FIGURE 3.3: Rigid body dynamics - update cycle<sup>3</sup>

As described earlier in the "General concepts" section different numerical integration techniques can now be used to apply the actual update step of the state vector depending on the implementation. In the case of the implementation presented in Chapter 6 the symplectic Euler method is used as part of the Bullet physics library [Coumans, 2015].

### 3.3.2 Constrained motion

After the correct way to calculate the underlying physics and the movement of rigid bodies has been established in the previous sections, it is now time to have a look at the more advanced algorithms responsible for interaction between multiple objects in a rigid body system like, collisions detection, collision forces or resting contact.

Because the individual computation of the topics discussed in this chapter would get too extensive for the scope of the thesis, this section is more set up as a broad overview.

The main goal of collision handling is preventing bodies from inter-penetrating by calculating the appropriate responses at the moment of contact. Because RBD simulations are modeled in discrete timesteps, collisions rarely happen at the exact time of evaluation. This means efficient and accurate algorithms have to be applied to faithfully find occurring collisions and furthermore determine the exact moment of the event.

#### 3.3.2.1 Collision detection

Depending on the shape of each rigid body, collision detection can be more or less complex. Whereas primitive shapes like spheres or cubes can be handled easily, for arbitrary polyhedra considerably more computational effort and thorough testing is required. The collision detection process happens on multiple levels, increasing in effort along the way. Thereby every polyhedron has to be tested against every other polyhedron at every timestep. For large simulations containing a multitude of bodies the amount of tests can get excessive. This is why preprocessing steps are taken to limit the amount of evaluated polyhedra to the ones, which are actually likely to collide.

<sup>3</sup> source: [Parent, 2012, p.209]

A simple approach to narrow down possible collision pairs is using a bounding box, or also called min-max test. A bounding box is very easily created for most shapes by looking for minimal and maximal values of  $x, y$  and  $z$ . If the bounding boxes of two bodies don't overlap, then there is no possibilities of the actual bodies overlapping. In case of big deviations between actual shape and bounding box constructed, more fitting bounding shapes like spheres or slabs can be used [Parent, 2012, p.223f].

Once the possible collision pairs have been thinned out, the next step in collision detection is taken. An efficient way to find most collisions is to check if any vertices of a polyhedron lie inside of another polyhedron and vice versa. For convex polyhedra this test is fairly simple. For each tested point the planar equations of each face are evaluated with the coordinates of the point. The checked point is inside the polyhedron, only if all equations are evaluated negative.

For concave shapes this technique is not reliable so other methods have to be used. One possible approach would be to construct a semi-infinite ray from the point and test for intersection with the faces of the object via their planar equations. The point lies inside the polyhedron if there is an uneven amount of intersections, otherwise it lies outside [Parent, 2012, p.224].

A last test to be done is concerned with edge-face intersection and the other way around. These intersections are not detected via the previous vertex tests, so a different method has to be used, where each edge is tested for intersection with the plane of another objects face and subsequently checked for containment in that face.

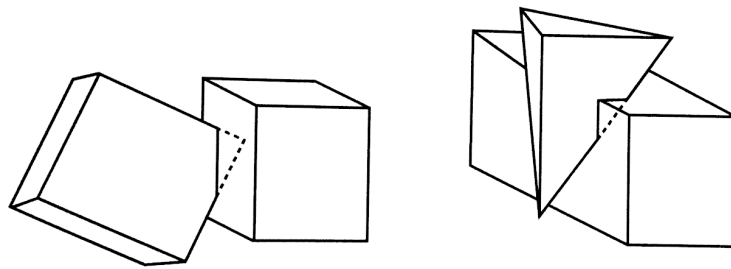


FIGURE 3.4: Different intersection types: vertex-face(left) and edge-face(right)<sup>4</sup>

Using the described methods for collision detection leads to relatively stable and accurate results sufficient for computer animation and visual effects use.

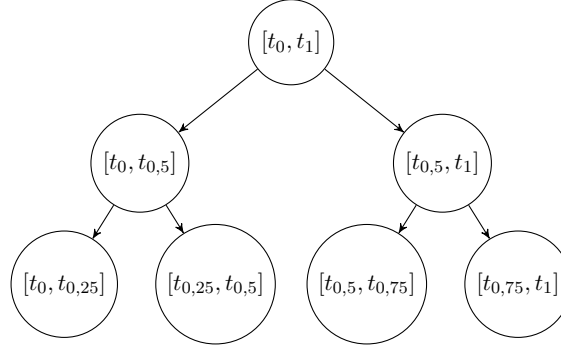
### 3.3.2.2 Collision response

To calculate an accurate response to rigid body collision it's usually necessary to "back up" time to the actual moment of collision  $t_c$  instead of using the time at the end of the

<sup>4</sup> source: [Parent, 2012, p.225]

timestep  $t_1$ , where it got detected. To find this moment a binary search strategy can be used. The relevant timestep is divided into two periods. At the dividing time  $t_{0,5}$  contact is checked and it can now be determined in which period the time of contact lies. This period then is further divided into 2 steps and the search goes on until a moment of time is found, where the collision object lies within a certain threshold near the collision plane [Baraff, 2001].

FIGURE 3.5: Illustration of a binary search tree for a time step.



A faster, but less robust method to find the moment of collision could be observing the position at the beginning and end of the timestep, creating a liner path between them and then calculating the intersection point with the collision surface. Based on this the time of collision  $t_c$  can be approximated.

Once the moment of impact  $t_c$  has been found up to a certain accuracy, it is now time to calculate the actual collision response. To prevent interpenetration and model the behavior of real rigid objects when colliding, the velocities of both bodies have to change immediately to enforce an immediate reaction. Since any applied contact force would need some time to effect the velocities appropriately, the impulse  $J$  is used to calculate the necessary change in momentum and therefore lead to the velocities. An impulse can be imagined by a very large force acting in a small amount of time.

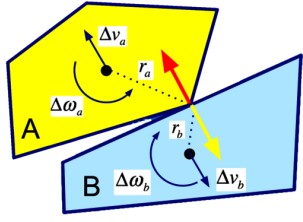
$$J = F\Delta t$$

$$J = F\Delta t = Ma\Delta t = M\Delta a = \Delta(Mv) = \Delta P$$

And based on these equations the resulting velocities can be calculated by  $\Delta v = \frac{J}{M}$ .

The actual calculations behind  $J$  are quite complex and are presented in the following figure 3.6.

To simulate a fully fleshed out rigid body system a number of additional considerations would have to be made, which are not discussed in detail here, for example calculating friction and resting forces or linked hierarchies of objects.



The diagram shows two rigid bodies, A (yellow) and B (blue), in contact. Body A has a center of mass \$r\_a\$ and a point of contact \$r\_b\$. Body B has a center of mass \$r\_b\$ and a point of contact \$r\_a\$. The contact normal is \$n\$. The desired velocity change is \$\Delta v\_{desired}\$. The impulse is \$p\_{correction}\$. The diagram also shows the angular velocities \$\Delta \omega\_a\$ and \$\Delta \omega\_b\$ and the velocity changes \$\Delta v\_a\$ and \$\Delta v\_b\$ at the contact points.

$$p_{correction} = M_{effective} \Delta v_{desired} \cdot n$$

$$p_{correction} = \frac{-\Delta v_{ab} \cdot n}{n \cdot n \left( \frac{1}{M_a} + \frac{1}{M_b} \right) + \left( \frac{r_a \times n}{I_a} \right) \times r_a + \left( \frac{r_b \times n}{I_b} \right) \times r_b}$$

FIGURE 3.6: Collision response impulse calculation<sup>5</sup>

### 3.4 Fluid dynamics

While the underlying physics are the same for particles, rigid bodies and fluids, the algorithms and exact calculation vary significantly for the latter. Computational fluid dynamics (CFD) concerns itself mainly with liquids and gases. Opposite to the previously introduced simulations fluids have no definite individual points or a constant shape. Instead they are composed of almost infinitely small molecules. The assumption is made that throughout these molecules, certain properties are defined, which can be promoted to one continuous medium with smoothly varying values. This assumption is the so-called continuity assumption [Parent, 2012, p.270].

The properties defined inside the fluid can include things like density, velocity, temperature or momentum. All of those are constantly changing around and vary depending on position in the fluid as well as time.

The most important tool to simulate the behavior of fluids are probably the incompressible Navier-Stokes equations, taken from [Bridson and Müller-Fischer, 2007]:

$$\frac{D\vec{u}}{Dt} + \frac{1}{\rho} \nabla p = \vec{g} + \nu \nabla \cdot \nabla \vec{u} \quad (3.1)$$

$$\nabla \cdot \vec{u} = 0 \quad (3.2)$$

To better understand them, some symbols have to be explained.  $\vec{u}$  is usually used in the context of fluid dynamics to represent the velocity of the fluid. The term  $\frac{D\vec{u}}{Dt}$  is called material derivative and defines the change of momentum based on time and space.  $\rho$  simply denotes the density of the fluid, approximately  $1000 \text{ kg/m}^3$  for water or  $1.3 \text{ kg/m}^3$  for air. The letter  $p$  is the pressure, the force per area the fluid exerts.

<sup>5</sup> source: [Coumans, 2015]

$\vec{g}$  is usually used to represent the gravitational acceleration. In case of computer animation other "control" accelerations added by the artist to create a certain behavior are added under this variable as well, also called body forces.

Lastly the letter  $\nu$  stands for the kinematic viscosity of the fluid. It defines how much resistance the fluid offers towards deformation of its flow.

The first equation (3.1) is called momentum equation, whereas the second (3.2) is the incompressibility equation. Broken down, the momentum equation basically states the same as Newtons second law of motion  $\vec{F} = m\vec{a}$  [Bridson and Müller-Fischer, 2007]. This is more obvious when multiplying by  $m$  and doing some rearrangements.

$$F = m\vec{g} - \frac{m}{\rho}\nabla p + m\nu\nabla \cdot \nabla\vec{u} \quad (3.3)$$

Based on the term (3.3) the individual forces can now be considered individually to see which forces affect a fluid. The term  $(m\vec{g})$  describes all external forces, gravity and artist defined force, affecting the fluid. The other two terms describe forces applied by the fluid on itself. Whereas  $(-\frac{m}{\rho}\nabla p)$  states the force applied from higher pressure areas towards lower pressure areas, the second term  $(m\nu\nabla \cdot \nabla\vec{u})$  describes the force applied due to viscosity. The viscosity tries to keep the velocity of a bit of fluid similar to nearby parts and tries to minimize differences in velocity.

Understanding the contents of the Navier-Stokes equations is essential to understand how the motion of fluids is built-up and how it can be solved numerically.

### 3.4.1 Model overview

Overall there are two fundamental models to describe a fluid system. The grid-based model, also called the Eulerian approach, and the particle-based model, also called Lagrangian approach. Their main difference lies in how the attributes of the fluid are tracked and how it is partitioned.

For the Eulerian model the fluid is divided into a fixed grid of individual cells. Every cell samples all attributes of the fluid and the flow between cells is calculated. Then the cells are updated with the new values. The density of the cell is the main indicator of how much of the fluid is present and how visible it is for the final rendering. The flow out of each cell is calculated based on velocity, density, size and external forces. The flow into each cell is determined by evaluating density values nearby. A disadvantage of this approach is that the dimensions of the simulations have to be set beforehand or alternatively a more complex dynamically adapting data structure has to be used [Parent, 2012, p.271f]. This approach is mostly used for gases.

The Lagrangian approach is based on tracking individual particles building up the fluid. It is comparable to the concept of tracking the individual molecules of the fluid. Each particle carries the necessary attributes and flows through space. Since this approach almost equals the one used for particles and RBDs, familiar and simple equations can be used and updating the acceleration and velocities including external forces is relatively simple. The disadvantage of this approach is for once the amount of particles needed to simulate accurate results and also the issue of converting the particles to a consistent surface for rendering, which can lead to disappointing results [Parent, 2012, p.272]. This approach is often used for liquids, thanks to recently emerging advanced smoothed particle hydrodynamics (SPH) algorithms [Bridson and Müller-Fischer, 2007, p.65].

### 3.4.2 Calculating fluids

In addition to the presented first Navier-Stokes equation (3.1) some simplifying assumptions are made in order to increase efficiency and get rid of unnecessary complexity regarding the calculations.

The first assumption is established through the second Navier-Stokes equation (3.2). It states that the fluid is incompressible, meaning that its volume doesn't change. Although technically real fluids can be compressed under extreme conditions, this effect is minimal to non-existent under normal conditions and can therefore be disregarded for computer animation [Bridson and Müller-Fischer, 2007, p.7]. A velocity field that satisfies the incompressibility condition is called divergence-free. Since the velocities of a fluid are updated throughout the simulation, it has to be ensured that it stays divergence-free. This is done via the pressure field [Bridson, 2015, p.12].

Another simplification that can be made is dropping viscosity. Because it plays a minor role in most situations beside simulating thick syrup or very small scale droplets, it can simply be left out of the momentum equation (3.1) leading to the simpler Euler equations (3.4 & 3.5) often used for actually calculating the fluid simulation.

$$\frac{D\vec{u}}{Dt} + \frac{1}{\rho}\nabla p = \vec{g} \quad (3.4)$$

$$\nabla \cdot \vec{u} = 0 \quad (3.5)$$

Incidentally when numerically simulating fluids some errors are introduced called numerical diffusion, which have the same effect as actual viscosity and therefore dropping it out of the equations has even less of an impact [Bridson, 2015, p.39].

A last theoretical consideration has to be made before an actual approach for numerical simulation can be presented. Up until now all equations were based on parts of the fluid



moving inside of it. However for most applications in computer animation an important part of fluids is their behavior regarding other objects colliding or bordering. Therefore handling so-called boundary conditions is one of the most important parts of simulating fluids [Bridson, 2015, p.13].

Same as all other previously presented simulations, fluids are modeled in discrete timesteps. Choosing a timestep is the first step of the simulation. With the increase of complexity regarding the calculations of the simulations, the importance of determination of a good timestep increases as well.

There are various approaches of numerically simulating fluids most using the fundamentals presented in this section up to now. Some vary only in detail some take completely different steps than others. One thing many still have in common is splitting up the calculation into several steps based on the Navier-Stokes or Euler equations. This simplifies the individual calculations necessary for simulating a fluid significantly.

A common division is the separation of the advection step(3.6), the body forces(3.7) and the incompressibility/pressure part(3.8) [Bridson, 2015, p.19f].

$$\frac{Dq}{Dt} = 0 \quad (3.6)$$

$$\frac{\partial \vec{v}}{\partial t} = \vec{g} \quad (3.7)$$

$$\frac{\partial \vec{v}}{\partial t} + \frac{1}{\rho} \nabla p = 0 \quad (3.8)$$

to ensure  $\nabla \cdot \vec{u} = 0$

For the first part (3.6) an algorithm can be developed which advects any quantity through the velocity field over a certain time interval. The body force equation(3.7) can be calculated via a simple Euler integration. And finally an algorithm has to be developed which calculates and applies the amount of pressure necessary to keep the velocity field divergence-free and enforces all boundary conditions(3.8).

The order in which theses individual steps are calculate and applied is important, because of the incompressibility condition. To keep volume in a fluid constant throughout advection, it has to happen in a divergence-free velocity field. This means the advection step can only happen after the appropriate pressure has been calculated and applied, to prepare the velocity field.

In conclusion this leads to the following basic fluid solver [[Bridson, 2015](#), p.20]:

- Start with divergence-free velocity field  $\vec{u}_0$
- For every timestep ( $n = 0, 1, 2, \dots$ ) do:
  - Determine optimal timestep  $\Delta t$
  - Create new velocity field  $\vec{u}_a$  by advecting  $\vec{u}_n$  with itself
  - Add body forces:  $\vec{u}_b = \vec{u}_a + \Delta t \vec{g}$
  - Make velocity field  $\vec{u}_b$  divergence-free and prepare it for next step  $\vec{u}_b = \vec{u}_{n+1}$

## Chapter 4

# Production environment

The destruction simulation implemented as part of this thesis in Chapter 6 was intended for use in the short film "The Girl Beyond". The short film is a student project developed and produced as part of the "Visual Effects Studio Produktion" at Stuttgart Media University.<sup>1</sup> This is a university course where a team of 10-15 students spend one year realizing a whole film project from developing and writing a story, over planning and preproduction, executing the actual shoot and finally post production including working on some heavy and complex visual effects shots, all under the supervision of Prof. Katja Schmid.

### 4.1 "The Girl Beyond"

"The Girl Beyond" is an almost 10 min long, short drama about escapism, prostitution and life in despair. The story mostly takes place in a small, dark and run-down hotel room, obviously part of a brothel in the city. The prostitute "Amelie" meets a new customer, they exchange money, start to make out and eventually have intercourse. Amelie seems very cautious and frightened. At the same time the customer is determined to take what he paid for and roughly forces himself onto her. The scene is intercut with scenes in a bathroom where Amelie is furiously scrubbing every part of her body and breaking down crying.

In an effort to escape the situation she's forced to endure, her mind slips away and she transitions into a beautiful dream-scape where her younger self and her grandmother enjoy a calm and soothing afternoon together. She stays for a while and watches until she's ripped back into reality, where her customer is just finishing. He immediately gets up and leaves, while she stays in bed deep in thought. Contemplative, but determined

---

<sup>1</sup> <https://www.hdm-stuttgart.de/vfx/>

she steps to the window as if to jump to her release, when the next customer knocks on the door.

The team working on the production consisted of students with a wide range of experience levels in various areas and varying interests for the post production. The more experienced team members expertise lied in compositing, motion graphics and fx, whereas the novice members had some basic all-round knowledge in compositing and 3d, but almost no experience in working in a real production environment. The lack of experience was compensated by motivation, efficient division of tasks and constant exchange of knowledge. Although, a flat hierarchy is intended for this kind of student production, establishing lead artists, who offer supervision and guidance, helped immensely to make sure the less experienced members could reach their full potential quickly.

As the shot containing destruction fx was the most complex one of the movie it became mainly a team effort between the more experienced members of the team, one each for fx, lighting/shading and compositing with some additional help from most of the rest of the team.

Because the whole actual production has to take place in one semester, the effective time spent on post production is very limited. Leaving out three weeks for cutting, grading, footage preparations and mastering, the net time to create all the visual effects comes down to about five weeks. This leads to a few weeks of very long and exhausting days for most of the team to achieve the highest amount of quality possible in such a short amount of time. Fortunately for the destruction fx a lot of research & development could already be done before the shoot or the picture lock and in the end at least two full months were spent on the simulation implemented in Chapter 6.

## 4.2 Shot description

The actual shot containing the destruction fx is intended to be one of the big moments of the movie with impressive visuals and a strong impact. It is set at the moment between Amelie's suffering and her escape into her mind, causing the transition. The viewer first gets tricked into thinking her interaction with the customer is over, and she wakes up alone in her brothel room. She then gets up and everything around her starts shaking and trembling. She walks toward the wall, when the wall suddenly breaks apart by a supernatural force and opens up a passage straight onto a meadow in a beautiful landscape where her grandmother and younger self appear.

On the one hand, the destruction effect has to establish that this scene isn't actually happening in reality by using unrealistic destruction forces in form of an anti-gravity

effect and others. On the other hand this supernatural effect should still look and feel realistic and believable in its supernatural way without it becoming over the top.

This leads to the big challenge of balancing these two objectives throughout the pipeline with departments seemingly working against each other. While lighting/shading and comp is concerned with photorealism, the supernatural nature of the fx of the shot make this significantly harder.

The implementation discussed in Chapter 6 focuses on the fx part of the shot and describes how supernatural forces were mixed with physically accurate behavior to create the required simulation.

## Chapter 5

# Software & pipeline

A wide range of tools, software packages and applications were used to create the visual effects and in particular the destruction effects in "The Girl Beyond"

All simulations and fx work were created with Side Effects Houdini. While particles and dust were rendered with Mantra directly inside Houdini, all rigid bodies were exported to Maya via Alembic and then rendered in V-Ray. Various other applications were part of this process, as among others Mudbox, Mari, Substance Painter and Photoshop were used for UVs, texturing and shading. Additionally, Syntheyes and Mocha were used for camera tracking and finally Nuke for compositing. Exchange between most of these was done via Alembic for 3D or EXR for images.

All visual effects work was done in the VFX Labs of Stuttgart Media University, where twelve mid-range workstations were available and access to another 20 low-range render slaves possible, if they were not in use by other students working in the public computer rooms.

### 5.1 SideFX Houdini

SideFX Houdini is a flexible, procedural, node-based 3d application with powerful particle and dynamic environments developed by Side Effects Software. Especially used by fx artists and technical directors it's also offering a solid and complete toolset for various other 3d tasks even including compositing.<sup>1</sup>

Houdini was first released 20 years ago and based on its procedural approach has been known as a very unusual and complicated 3d application compared to main competitors. At the same time, its advanced simulation and fx capabilities caused it to gain traction among artists especially in those areas.

---

<sup>1</sup> <https://www.sidefx.com/products/houdini-fx/>

Over the last decade, after more and more artists grasped the advantages of procedural work flows and while continuously various improvements have been made to the interface, usability and learning materials, Houdini’s popularity in the industry has been growing tremendously. At the same time, the demand for experienced Houdini artists by vfx studios is higher than ever.

Houdini is organized in multiple different node trees each containing operations for a specific context. Depending on what needs to be achieved a different context has to be used. Data between contexts can and often has to be exchanged in various ways. Most of the times this workflow means that the initial setup of most tasks takes a little bit longer the first time, but once built, it can be reused, changed and adapted infinitely.

The top level context/network is called the object or ”scene” level. There the different object nodes are created to build the scene. These nodes could be cameras, lights or geometry amongst others.

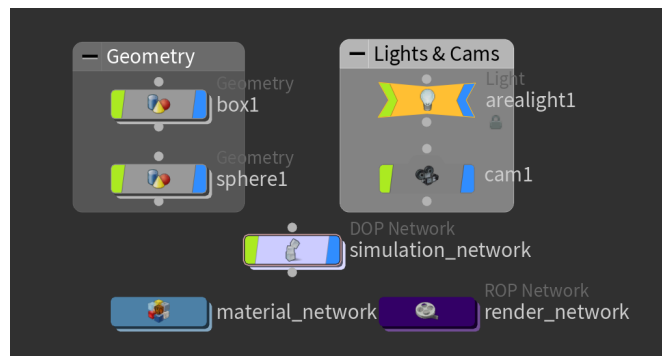


FIGURE 5.1: This is a typical Houdini scene node tree with different networks for various contexts. Most of these contain additional nodes inside, while the camera and lights are mostly used on the object level.

Two of the most used contexts are the geometry and the dynamics context and a quick introduction to both of them will be given in the next subsections.

### 5.1.1 Geometry context

The geometry context is where everything modeling-related and most general purpose operations are implemented. This is also where the geometry is prepared and customized for simulation. The operators available in the geometry context are called surface operators or SOPs and they work with geometry data.

In general, working in Houdini can be described as working with data. Every context varies slightly in handling the data, but it can be roughly described like this: At the top of a node tree data is created(e.g. by a Box SOP) or read in(e.g. with a File node). The data gets passed along to the next node in the tree, this node carries out

its operations on the data and outputs the (changed) data to be sent to the next node. This goes on until the end of the network is reached. At each step along the way, the complete data can be read out or manipulated by the artist. This is extremely useful and essential for building and debugging networks.

Data in Houdini is represented as attributes. There are four different classes of attributes: point, vertex, primitive and detail. The attributes can be checked by looking at the geometry spreadsheet, which has a table for each attribute class and lists all geometry and their attributes. What's special is that each attribute of each point can be individually accessed, changed, shuffled around and even promoted/demoted between classes. This allows for a very low-level access and direct control to the geometry.

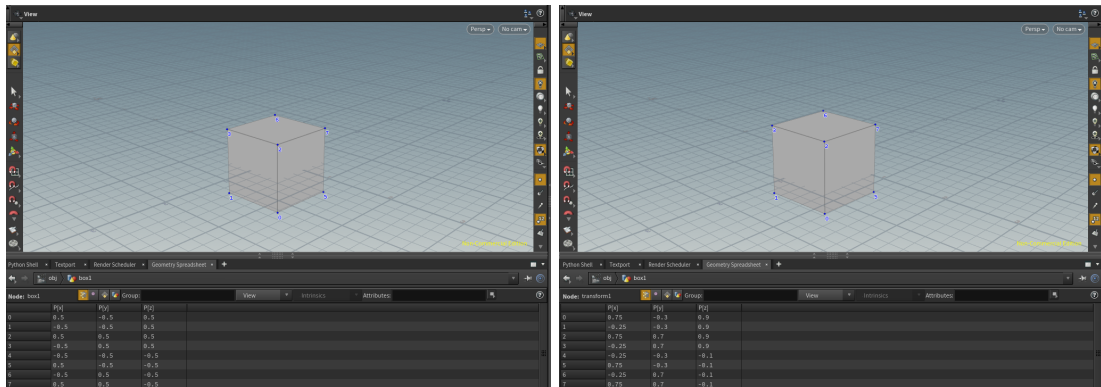


FIGURE 5.2: All geometry data can be accessed in the geometry spreadsheet. Each row is a point and the columns are its attributes, in this case the position value for each axis(left). When the geometry is changed, for example through a transformation, the data in the spreadsheet changes accordingly(right).

### 5.1.2 Dynamics context

The dynamics context is used for all simulations and time-based tasks, where the position, movement or any attribute of an object is dependent on its status in a previous frame. The operators available in the dynamics context are called dynamic operator(DOPs).

To create simulations, geometry is imported from SOPs into dynamic objects or in rarer cases created directly in DOPs. Then data is added to the dynamic objects like attributes, constraints or forces. Depending on the type of simulation a solver has to be chosen. There's a wide range of solvers available for different purposes. For example there is a wire solver for movements of plants or trees, a FLIP solver for simulating fluids like water, a FEM solver to simulate soft/deformable tissue and most importantly for destructions there are rigid body, particle and pyro solvers.



The solver takes the dynamic object and the data attached to it and based on that calculates position, velocity and depending on the solver various other attributes for the next frame. Because all solvers work in the same context it's possible to have different kinds of dynamic objects and solvers interact with each other. For example rigid bodies being blown away by an explosion calculated from a pyro solver and at the same time flames colliding with and flowing around those rigid bodies.

Because calculations always depend on the status in the previous frame, all data, including a wide range of internal attributes for calculations, has to be preserved in memory as a cache. For larger simulations with a lot of objects or resolution this can lead to a large part of the system memory being blocked by caches instead of used for the actual simulation. This is why often during simulations, the simulated data is merged back into a geometry context and saved out as a file cache. This also allows for easy playback and also further use of sim data without having to re-simulate everything every time.

The solvers used in implementing the destruction fx in Chapter 6 will now be described in more detail.

#### 5.1.2.1 POPs / particles

Particles in Houdini are simulated as POP Objects with the POP Solver. The POP Solver is one of the most basic and also efficient solvers in Houdini. It makes full use of multi-threading and is very memory efficient. If an effect can be solved via particles that's always the best option because of their performance, ease of use and art directability.

Particles can be birthed in specific locations, based on imported geometry or even based on other particles. For each timestep the solver loops through every particle and updates them based on velocity and the forces applied.

In addition to simple particle dynamics Houdini allows for more advanced functionality like differing mass among particles or a range of collision behaviors between particles and/or other surfaces.

Based on the limitations of particles compared to more sophisticated simulations like rigid bodies, they are especially useful for secondary effects like very fine debris, dust and general "sub-pixel" effects, where a sense of movement and irregularity support the more pronounced effects of actually visible pieces. Another important use of particles can be sourcing of a smoke/pyro simulation. In this case particles are simulated to generate the base motion and starting parameters for a more computationally expensive fluid simulation. Both of these techniques will be described in more detail in Chapter 6.

### 5.1.2.2 RBD & Bullet

There are multiple rigid body solvers available in Houdini. Since Houdini 12 the default and most-used one has been the Bullet solver. The Bullet solver is based on the Bullet Physics Library<sup>2</sup>, an open-source physics engine, focusing on efficient collision detection, rigid body dynamics and the solid implementation of various constraints [Coumans, 2015]. Because of its deep integration in Houdini it works flawlessly with all of Houdini's RBD features like forces, constraints and glue and can interact with other solver types like cloth and wire solvers.

The bullet engine is optimized for fast calculations in games and visual effects and can handle large data sets. It works best with primitive collision geometry like boxes and cylinders but also with convex hulls. This means before the actual simulation an efficient convex collision geometry is created for every object in the sim. Although concave collision is also supported through Bullet in Houdini, it is not recommended and can lead to instable, explosive simulations with a high computational effort [SideFX, 2017c].

For most RBD simulations especially destruction fx geometry needs to be pre-fractured in SOPs into many smaller pieces, varying in size and shape. Then they are glued back together and brought into DOPs as packed objects. Packed objects are single points, referencing back to their original geometry. Their attributes and the representative collision shape are used to save and calculate simulation relevant data. This makes simulating more efficient than using the complete original geometry. Another advantage is, because every piece in the simulation is now treated as a single point, similar to particles, the whole range of particle operators can be used on the rigid bodies. This helps in making the simulation easily art directable.

More about RBD workflow, model preparations, simulation parameters and post-sim procedures will be discussed in Chapter 6.

### 5.1.2.3 Pyro solver

The two main solvers for smoke, fire, dust and most gaseous fluids in Houdini are the Pyro solver and its simplified variation the Smoke solver, which works without combustion or flames. It is generally recommended to use the pyro solver for fluids sims even if there's no need for flames, because it has some additional controls and settings which make it superior to the smoke solver.

Both solvers work with the grid-based(*Eulerian*) simulation approach introduced in Chapter 3. The maximum grid size and resolution have to be set in the beginning

---

<sup>2</sup> <http://bulletphysics.org/wordpress/>

of the simulation and the current size can then be dynamically resized depending on the content of the simulation.

Similar to the rbd sim, data has to be prepared in SOPs to source into the pyro sim. The most important attributes to source are density and velocity. For fire or combustion simulations in addition to the previous attributes there's fuel and temperature. There are various ways to source attributes into the sim. The most simple one, would be to create a volume as a starting point including the relevant attributes and simulate from there.

Another more common approach is to create a particle sim with the wanted look for the final smoke sim. These particles can then be directly used to control the movement of the fluid via sourcing their attributes into it. This way a lot of time can be saved by iterating on the fast to calculate particle sim instead of the heavy pyro sim.

Houdini is fully compatible with VDB a highly optimized file format for dynamic volumetric data [Museth, 2013]. A lot of computation time and storage can be saved by working with VDBs instead of Houdini's own volume data type while staying in SOPs as the latter is not as efficient. While sourcing and collision is compatible with VDB, unfortunately the pyro solver works with Houdini volumes internally for its calculations. This means the output of a sim is in Houdini volumes as well and can greatly benefit from converting to VDB before caching to disk.

The usual DOP Forces can be applied to pyro sims, but most often the better way to take control of the motion is by sourcing velocity or collision into the sim. In addition, the behavior of the fluid can be changed and controlled drastically by adding microsolvers. Microsolvers are additional solvers that work on top of the functionality of the pyro solver to influence certain behaviors of the sim. A lot of them are already built into the pyro solver but only with a more restricted and less flexible functionality.

## 5.2 Alembic workflow

It is often necessary to exchange data between 3d applications. Especially Houdini is often used for simulations, but rendering is done in a different software because of more efficient renderers or more artists being available for other applications. One of the most adapted and functional formats to exchange data between applications is Alembic.

”Alembic is an open computer graphics interchange framework. It distills complex, animated scenes into a non-procedural, application- independent

set of baked geometric results. This ‘distillation’ of scenes into baked geometry is exactly analogous to the distillation of lighting and rendering scenes into rendered image data.”<sup>3</sup>

The main goal behind alembic is to store the data in a very lightweight and efficient fashion, keeping all the important informations, structuring them in an organized way and getting rid of unnecessary proprietary data. In addition because of its open-source structure, Alembic is fully customizable and many visual effects studios build their own toolset to interact with its data.

Theoretically and when setup correctly this allows geometry data to be imported seamlessly into all supporting application, while using minimal disk space and computational power.

### 5.2.1 Alembic in Houdini

Alembic support is well integrated into Houdini and offers a wide range of options and additional functionality [SideFX, 2017a].

An important feature is ”demand-loading” alembic data. In this case, Houdini loads the geometry contained in the alembic directly into the scene/viewer without having to first save it in RAM. The same works for Mantra, rendering geometry directly from disk, instead of using up rare memory, can make a huge difference when rendering big scenes with millions of polygons. When working with packed primitives/alembics Houdini loads only the ones needed at the moment, making the geometry i/o even more efficient.

The export options offer a similar range of functions. There are two different output formats, HDF5 and Ogawa, the latter being the newer format with less space requirements and better, multi-threaded performance. HDF5 is still supported for backwards compatibility. A very useful option is to structure the output via a path attribute on the geometry. When chosen, this separates the alembic into several different groups/nulls, one for each distinct path value and also takes into account the whole hierarchical structure. So instead of one mesh, the alembic can be imported into other software as one mesh per path value, for example each breaking piece of a wall individually, grouped by material. A practical example of this will be shown in Chapter 6.

Another important setting for the alembic export is how to handle packed transforms. One option is to save the packed geometry in full for the first frame and from then on only save out the occurring transformation. Of course this is only possible if the actual geometry is not deforming and the point count is not changing. The other option would

---

<sup>3</sup> <http://www.alembic.io/>

be to save the transformed geometry for each frame. The latter leads to a much higher file size because the whole geometry has to be saved every frame, whereas for the first option it only needs to be saved ones and from then on only the less expensive transform data.

The disadvantage of option one is that the geometry first needs to be "calculated" for a frame before it can be accessed in another software. So the geometry of the first frame will be read and then transformed to the position, rotation and scale saved in the accessed frame. Depending on where the performance bottleneck lies, storage space and read speeds(for example over a network) or computational performance and longer scene setup times, when rendering, one setting makes more sense than the other.

## Chapter 6

# Implementation

The following chapter describes in detail how the destruction fx for the transition shot in "The Girl Beyond" were created. The official shot name during production was UEB010 and for readability it will be called the same throughout the chapter. Most individual subsections are split up into two parts, first describing the general approach in implementing destruction fx and subsequently explaining the particular steps taken for UEB010.

The focus lies mostly on the simulation work, also including the necessary preproduction relevant to the effects and some post-sim procedures to increase the overall quality. This is in no way meant as a definite guide on how to create destruction fx, but more along the lines of an example approach for simulations in visual effects, including some industry best practices, necessary considerations and possible issues coming up, when working on this kind of project.

A lot of the practices presented in this chapter were partly developed while working on the shot and partly gathered from online resources, mainly a Houdini user forum<sup>1</sup> and the excellent tutorials of Steven Knipping<sup>2</sup>, Senior RBD / FX TD at Industrial Light & Magic.

Since the rendering portion was mostly done as a team effort, partly in another software package and lies beyond the scope of this thesis, it will only be brushed upon.

### 6.1 Planning phase

The groundwork for any visual effects happens in the pre-production and planning phase of a movie. Important tasks include testing, research and development, concept work,

---

<sup>1</sup> <http://forums.odforce.net/>

<sup>2</sup> <https://www.cgcircuit.com/instructor/steven.knipping>

look decisions and planning of the actual shoot [Okun, 2010, p.33f]. This is especially important for complex shots with multiple different visual effect layers like the transition shot for "The Girl Beyond".

One of the most important parts in creating high quality visual effects is to design the shot to the same detail as any another regular shot regarding composition and other creative decisions, but keeping the visual effects in mind. It's essential to figure out how the visual effects can support and help communicate the director's vision without overshadowing the original point and becoming the sole focus of the shot.

A way to ensure this is to work together in collaboration between the director, the DoP and the VFX Supervisor and to imagine shooting a real, finished scene without any vfx, that would work on its own as part of the movie. In that regard it can be helpful to first approach the storyboard without any restriction regarding feasibility and release full creative freedom for the shot. Once the initial design of the shot is up to a certain level and a first storyboard exists, it is now the job of the vfx supervisor to break it down to it's individual tasks and estimate the time, manpower and costs needed to realise the vfx [Okun, 2010, p.37ff].

After consultation with the relevant decision makers, changes are made to the shot in order to appease visual effects difficulties and bring down costs and time constraints until the shot conforms to its allocated budget. If the shot can't be changed anymore without losing its purpose in the movie, adjustments to other shots or the budget have to be made until the movies vfx are feasible as a whole inside the available scope.

Although it is not necessary to make all the decisions final at this point, indeed the more decisions are locked and the less variables there are left open, the better. This leads to better planning and preparations for the shoot and post production and in conclusion makes a high quality finished shot more probable [Okun, 2010, p.39].

But the development of a vfx shot is an ongoing process, there is still a lot that can change at this point in time up to the final vfx for the shot are delivered. Nevertheless a common vision and concept for the shot should now be established and fundamental changes should only happen under extraordinary circumstances.

The next step in the planning phase is the creation of a previsualisation to figure out the details and start planning the shoot.

Despite knowledge of the previous explained process, the planning phase for the vfx of "The Girl Beyond" and the transition shot in particular went on differently. As it was a student production with the main objective of creating impressive visual effects and with a team having only little experience in that area, the shot was approached in a different mindset than a regular director would. A lot of different ideas from a lot of

different people were brought together to create a shot with overwhelming amounts of interesting vfx with less focus on its purpose in the story.

This led to two major problems. The more obvious problem was the excessive workload of the shot. In its conception stage it was overstepping its boundaries in every aspect, including time constraints, manpower, existing experience, available computational resources and even feasibility during the shoot. The second issue was only discovered after several meetings trying to concretize the shot. Because of the amount of ideas that were discussed beforehand, there was no clear vision about the contents of the actual shot. This led to unnecessary talks and meetings about details that were never intended to be part of it.

To get the shot back on track the next step was to further define the shot by making final decision on some of the following questions and based on this create a first storyboard:

- Where is Amelie and what does she do?
- Where is the camera and if its moving, where does it move?
- What exactly happens with the wall?
- Which part of the wall is affected?
- What happens with the window?
- What is the timing for everything?
- What happens to all other objects in the scene?
- What is visible in the shot?

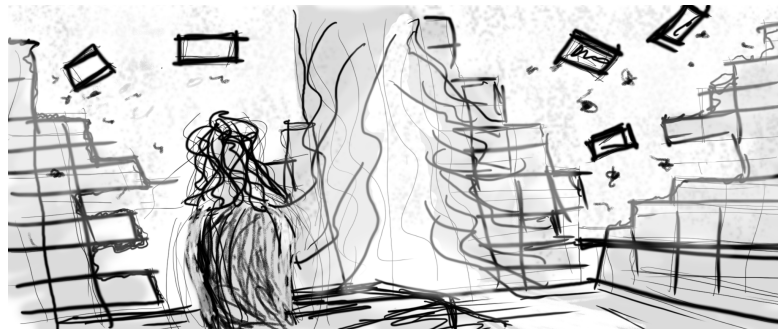


FIGURE 6.1: Initial storyboard for UEB010: Amelie can be seen standing in front of her bed towards the window. The walls left and right of the window are breaking apart and flying away, the window in the middle stays intact, while the curtain flutters in the wind. Particles and dust are implied in the air.

Once some of those questions were answered, a storyboard could be drawn and the shot narrowed down further. A lot of decisions regarding non-vfx concerns were made then while leaving most of the actual visual effects questions still vague and open. The scope at this point was still over the upper limit and in hindsight would not have been feasible, but as focus shifted towards other parts of the movie, more decisions regarding vfx were pushed back further towards the shoot.



### 6.1.1 Previsualization

After an advanced version of the storyboard is available it is strongly recommended to create a previs of the whole movie and especially all vfx shots.

Previsualisation, or previs, is a very rough representation of a shot, sequence or whole movie [Okun, 2010, p.53f]. Nowadays, it is mostly created using very low-quality 3d renderings with basic models and rough animation. Previs is used to improve production efficiency by uncovering possible issue for the shoot and to help with the planning of all departments [Gress, 2014, p.6]. It provides a first opportunity to see everything in motion, including a very first placeholder for visual effects. Based on the previs a lot of relevant decisions regarding timings and visible elements in the shot can be made and the shot as a whole can be fleshed out.



FIGURE 6.2: Frames from an advanced previs version of UEB010, the frame in the middle shows the same moment as the storyboard in the last section

For "The Girl Beyond" the previs helped immensely to unify the differing expectations in the team and to figure out the general timing. It also provided a good reference to begin with shooting preparations and first fx R&D.

Additionally the previs was used to get valuable advice by an experienced vfx supervisor. By showing the sequence in motion it was possible to get the point of the shot across easily and receive detailed feedback and tips on how to approach it. Most of the feedback for the movie was positive and supportive of what was planned so far. Only UEB010 was seen as a big issue and suggested to be reduced in complexity in order to achieve the level of quality that was aimed for.

Instead of reducing the complexity of the shot, the conclusion of the feedback was to split the workload up over more people.

### 6.1.2 Test shoots

Although probably a luxury for many professional productions (and replaced more and more by virtual production) proper testing of visual effects shots can be essential to find possible issues of a shot before it's too late to fix them. Only when shooting at the actual location with the actual available equipment and crew, the feasibility of the final

shot under the given environment can be judged conclusively in regards to for example spatial and temporary confinements.

Another advantage of proper testing is the practice gained by the team and the general awareness towards established workflows on set. The shoot of non-vfx shots is already a highly choreographed, painstakingly planned procedure including countless issues to keep track off. Adding vfx to the mix brings up many more issues with the further challenge that a big part of the set team has little to no understanding of the inner workings behind vfx. This means the vfx team on set has to be extra careful, taking care of proper green screen illumination, installing enough relevant tracking markers and noting any possibly useful measurements and metadata among other tasks.

In case of "The Girl Beyond" both of those points had a hugely positive impact but also fell short in certain ways.



FIGURE 6.3: First tests for UEB010 in the set. The goal was to recreate the camera movements from the previs while planning the exact camera setup

Because the transition shot was by far the most complex shot regarding visual effects and shooting complexity, it was the chosen shot for a test shoot a few days prior to principal photography.

There it was discovered that the camera move originally planned and previsualized could not be replicated on set because not enough room was available for the dolly and also the movement in general would have been too extreme to work smoothly. Additionally because of fixed walls it was impossible to shoot in the direction planned and get the original framing wanted.

Quick decision-making and appropriate judgments lead to a simplification of the camera move. Instead of a 180° curve including a 90° roll the movement was changed to a simple dolly in. Instead of an originally planned almost 270° coverage of the complete room over the course of the shot the camera direction was locked into one angle, showing only parts of two walls and then in the end a tilt upwards, but only after most of the upper half of the image was supposed to be replaced by vfx. This led to huge time savings in every step of the vfx pipeline, including matchmove, comp, but also fx, lighting/shading and rendering.

Another consequence of the change was a completely different layout for the green screen, tracker markers, lights, etc. On the test shoot this meant a lot of time spent on modification and testing for a new setup. But at the same time it meant a lot of time saved for principal photography, instead of losing half a day for alterations of the whole shot setup.

Unfortunately there were also issues unnoticed on the test shoot, that if remedied, would have saved a lot of time in post production and pushed the shot to a higher quality. First, the overall lighting conditions on the plate looked obviously like fake studio lighting and matched in no way to the later inserted outside environment created. This led to several complications in compositing and limited the believability of the final result noticeably. Secondly, the time needed for the greenscreen and tracker setup went unnoticed by most of the crew. Instead of offering help or increasing the available time frame for the real shoot, no adjustments were made, which led to frustration on all sides regarding the setup time and quality. This introduced unnecessary difficulty into parts of the match move and compositing.

### 6.1.3 Research & development

Research and development, often referred to as R&D, is one of the crucial steps when working on simulations or other complex effects. Specifically meant is the development and implementation of new algorithms or vfx techniques and building customized tools and software to create a certain look or effect [Okun, 2010, p.34]. More generally the term R&D is also used for any kind of research and testing that is done to figure out how to create or improve an effect.

R&D should start as soon as possible when the type and scope of the effects for a shot are known. The more complex, cutting-edge or unique the effect, the earlier in pre-production R&D should start.

The first step in R&D is looking for previous instances of similar effects being created. Almost every effect has been done before and almost every fx artist has faced similar issues when working on an effect. This does not mean that their approach and solution is the "right" one or that it should be imitated, but nevertheless a lot of knowledge can be gathered by looking at previous takes on a shot.

A great resource can be general cg or specific software user forums, where the used tools and settings inside an application are discussed in detail. Other possible sources are advanced learning materials supplied by the developer, talks and courses from conferences or for particularly progressive effects technical papers (for example from the ACM Digital Library) with a relevant topic.

At the same time as external resources are researched, it is important to start working on the effect itself. Various different approaches should be tested and evaluated based on advantages and disadvantages with a focus on the aspects important for the shot. Then depending on the progress made, the further outlook, found research and references an informed guess can be made on which approach to pursue further.

For "The Girl Beyond" the R&D phase was straight forward as apart from the supernatural forces it was an ordinary destruction effect and therefore the standard Houdini tools could be used without having to develop anything new. Most research went into how to optimize the simulations regarding performance and the wanted look. Also some testing had to be done on the general setup and hierarchies of the simulations, the various different forces and on constraints between the separate elements of the sim.

All in all R&D played a minor role and most time was spend on art directing and fine-tuning available tools instead of developing new techniques.

## 6.2 Scene setup

Although a lot of testing and R&D can be done beforehand, when the actual fx work on a shot starts, some time should be spent on setting up the scene correctly to allow for smooth working inside the pipeline and avoid some unnecessary problems.

The first thing to consider when setting up the scene is file location and file name. Usually there is a naming convention predefined by the production and an according file structure. If not handled by project & pipeline management software like Autodesk's Shotgun and its several software integrations it's important for the artist to keep track of it manually. This is essential for a clean pipeline and helps with compatibility to pipeline tools as well as easy exchange between artists and departments.

In most applications it is possible to set a project path for a 3d scene. In Houdini this is the \$JOB variable. When this is set correctly all other paths inside the scene can reference it and by using relative paths unnecessary updating of file paths can be avoided when changing the location of the project.

Another important setting is the scene scale. Especially in a pipeline with various different 3d applications it's essential to set the right scene scale. It determines which internal unit corresponds to which real life unit. So for example with a scene scale set to meters (Houdini's default setting), 1 internal unit equals 1 meter. This is important to avoid scaling issues when exchanging geometry between scenes. For Houdini it's especially important to set the right scale because the unit scale has a dramatic impact on the look of a simulation. Other areas that can be affected are lighting and rendering, because a lot of calculations in that area are dependent on distances.

A lot of fx work can be done without a dedicated shot camera, but to avoid unnecessary work on areas not seen, an approximated shot camera should be set up, based on the storyboard, previs or informations from the DoP. This includes setting up focal length, resolution and film gate and should be suitable for fx during the first steps. After shooting and as soon as there is a picture lock, it is time to bring the fx and the plate together. This is done by match moving the plate, importing the matchmove camera and bringing the undistorted footage into the scene as a background image or on an image plane. Now the exact dimensions, timings and actions of the shot can be judged and the fx created accordingly.

For the fx shot in "The Girl Beyond" the exact approach described in this section was followed.

### 6.2.1 Look references



FIGURE 6.4: Main simulation references for UEB010, showing plaster and dust behavior, when broken off a brick wall <sup>3</sup>

Although looking at references should already be part of designing the shot and previous R&D work, there is a conscious step at the beginning of the actual shot work, where references should be revisited and evaluated.

Collecting a definite portfolio of reference footage is essential before starting serious fx work. Especially when photo realism is desired, matching real life references is a great tool to achieve believable effects. When collecting references at first a wide range of different looks and aspects like movement, type, material qualities etc. should be considered. This pool of references can then be evaluated regarding the desired look and cleared up to keep only the best references for each aspect.

Now artists from every department can access the collection of references, look at the specific reference related to their task and use it to guide their vfx work.

<sup>3</sup> source: (a) <https://www.youtube.com/watch?v=KroAfHVx4zo> - access date: 2017-08-19  
(b) <https://www.youtube.com/watch?v=Ns7Rs1hS0Zk> - access date: 2017-08-19

For "The Girl Beyond" references were mainly used for the movement and behavior of the breaking wall and also for the shading and texturing of the individual materials. YouTube turned out to be especially useful for finding detailed footage of plaster and dust behavior when breaking of a wall but also more general building destruction and demolition to guide the overall look of the smoke sim.

### 6.2.2 Modeling

Modeling for fx has some distinctions to regular modeling. The main ones being differences in complexity, topology and shape of a model. While most models used for visual effects and integration into live footage need to be as detailed and high quality as possible to be believable, there are multiple issues when using the exact same models for fx work.

The more complex a geometry is and the higher the polygon count, the longer will a simulation calculate. More polys lead to a dramatically increased effort for simulation calculations especially for collisions. At the same time, there is little to no improvement in the actual result of the sim. As performance is a huge issue for complex simulations, low resolution or proxy geometry is used to calculate a simulation and later exchanged with the final geometry. This approach is described under the RBD section of this Chapter.

Another concern for models is the shape. Some solvers require a certain shape to work efficiently, as is the case for the bullet solver, Houdini's main RBD solver described in Chapter 5. The best way to ensure the right shape is to start the modeling process with the fx requirements in mind and making sure to model in individual objects, for example depending on material, instead of one coherent mesh.

If the model is already done and there are some issues regarding fx compatibility, it is part of the fx artist's job to prepare the model in a way suitable for the simulation.

As part of the production design and set building for "The Girl Beyond" a reference model for the room already existed. But because it was just a rough approximation made to reference the general dimensions of the room and the various objects placed in it, it was only used as a general reference on where to place the geometry for the sim and also partly for collision.

The actual geometry needed for the simulation was then modeled by the fx artist. This consisted of the individual bricks of the brick wall, the plaster attached to the bricks and the mortar between the bricks.

For the brick wall a procedural setup was build that creates a brick wall based on a given length, the number of rows and the number of bricks per row. This was done by starting

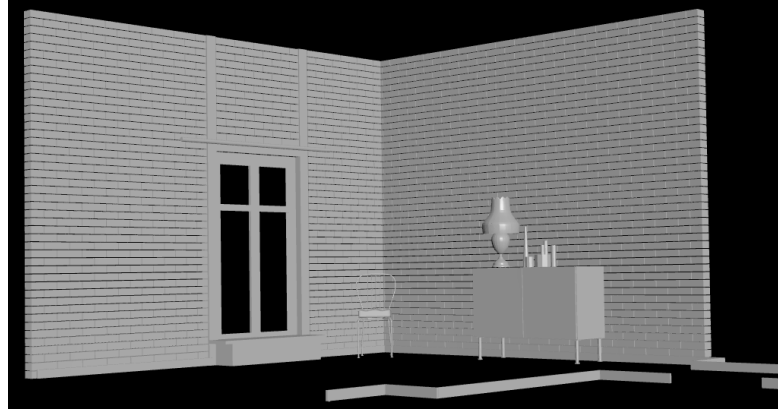
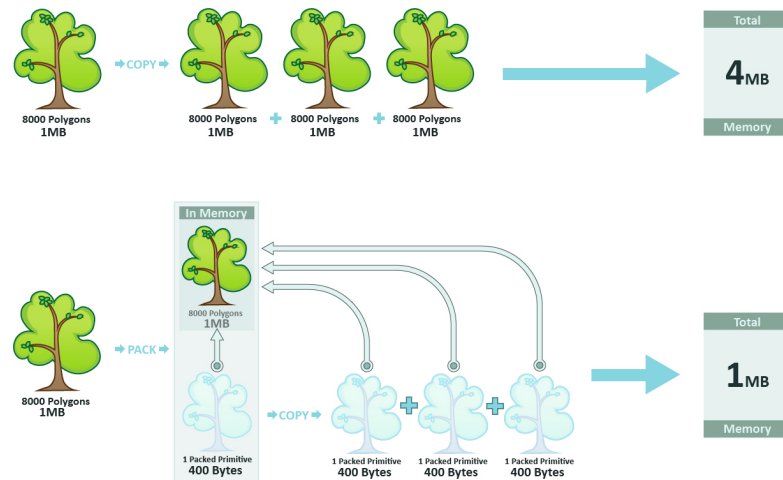


FIGURE 6.5: Procedural brick wall inside the given reference model of the set

with a line with points for each brick, then copying the line for every row of bricks and in the end copying a brick to every point on every line. Using expressions and references to parameters the bricks and their spacing were adjusted to fit each other, leaving a small gap for the mortar. The pattern and size of the bricks were chosen based on standard brick sizes and evaluating reference images. With the help of the reference set and the brick wall setup the necessary walls were then created and placed accordingly.

FIGURE 6.6: This illustration shows how packed primitives work inside Houdini. In case of UEB010 the trees shown here can be replaced with bricks. <sup>4</sup>

To keep the geometry simple, the bricks consisted of a single box with just 6 polygons and were instanced to every point using Houdini's packed geometry functionality. This means instead of copying the complete geometry for every brick, it just creates a reference to the original brick and therefore Houdini only has to keep the brick in memory once. To add complexity for the actual rendering later, the reference brick geometry was then exported to create a range of different high quality variations to use after the sim.

<sup>4</sup> source: <http://www.sidefx.com/docs/houdini/model/packed> - access date: 2017-19-08



The plaster was a very basic model of just a box in front of each wall with the respective dimensions to hide the brick wall behind and fitting the surrounding wooden beams and the floor.

The mortar was created using the bounding box of each brick wall, extending it outside by a small factor and then using a boolean operation to subtract the original brick wall from the extended bounding box. This left over the mortar required to exactly fill the gaps between the bricks.

### 6.2.3 Fracturing

Another step when preparing geometry for destruction is fracturing. If something has to explode, be destroyed or in general break into pieces, it needs to be fractured before the simulation [Gress, 2014, p.401]. At least that's the case for Houdini, except when using special exceptions like dynamic fracturing. There are other applications that handle fracturing differently and work with dynamic fracturing per default. This is more restricting as it does not allow for the same level of in depth control.

In Houdini there are different tools and approaches to fracturing. The more common and established one is voronoi fracturing. A more advanced, more customizable and especially through improvements in recent releases promoted approach is boolean shattering.

Voronoi shattering is based on the voronoi diagram. A voronoi diagram can be drawn for a specified set of points in a 2 or 3 dimensional space. Each point represents the center of a region or so called cell. The space is then partitioned in a way that every point in a cell is closer to the center of its cell than to the center of any of the surrounding cells. This means the boundaries of a cell lie exactly halfway between 2 center points, corner points on the boundary have the exact same distance to at least 3 surrounding cell centers [Fisher, 2004].

This diagram is "drawn" for randomly scattered points on the surface of the geometry. The geo is then divided into individual pieces based on each voronoi cell. Since the look of the diagram depends massively on the count and layout of the points, the look of the fractures can be controlled easily by manipulating the points before fracturing the geometry. For example the points can be scattered throughout the geometry instead of just the surface or the density of points can be higher in some areas than in others, to create a variance of size between the individual pieces. Advantages of this approach are its performance, reliability and the resulting convex shape for each individual piece.

The boolean shattering approach is based on boolean operations on geometry. The initial geometry is fractured by a second "cutting" geometry using booleans. This way



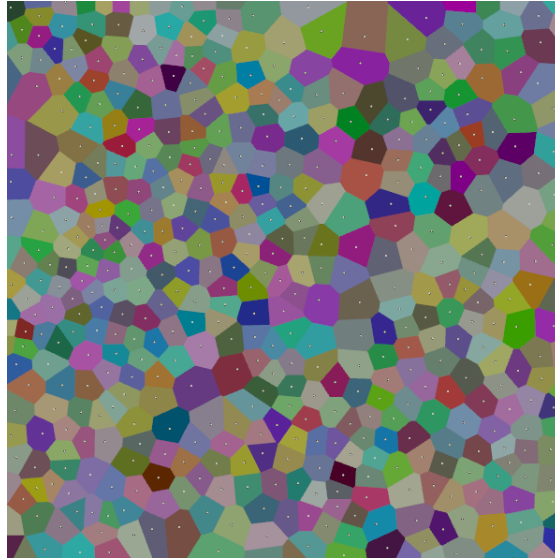


FIGURE 6.7: Voronoi diagramm

an object can be shattered in a realistic and art directable way using an arbitrary amount of cutting surfaces of any detail level[SideFX, 2017b]. For hero objects or when there are very specific fractures required, that cannot be replicated using voronoi, boolean shattering is definitely the superior approach.

At the same time there are some disadvantages leading to the further use of voronoi fracturing, despite its limited results. Booleans on geometry, although improved in recent software updates, are still unreliable and sometimes unpredictable. Using it can lead to weird issues regarding topology and general "cleanliness" of the geometry. It also requires a certain baseline in resolution and shape to create desirable results, which is often unpredictable. But probably the biggest disadvantage are the resulting concave shapes often appearing when aiming for a realistic fracture through booleans.

For UEB010 both approaches were used to create the required pieces for the destruction of the wall.

At first the bricks were not supposed to be fractured and broken apart. The whole simulation would be started of and directed by the motion of the bricks as the main unrealistic element of the shot, so the bricks could have been kept intact without causing any issues regarding believability. After a few simulations and while the motivation to achieve photorealism grew bigger, it became clear that it would improve the look of the simulation drastically, if the remaining bricks at the border of the wall would break apart and get affected as well. This would help in creating a more realistic, detailed and interesting shape for the leftovers of the wall.

Because there was direct control over the exact position and properties of the fracture required and it would only be a single fracture through the whole wall, the boolean

approach was used in this case. To create the fracturing geometry the group of active bricks was separated, converted to a coherent volume and then expanded to the approximate middle of the first row of remaining bricks. Then the volume was converted back to polygons and noise was added to the surface to create randomness and detail for the resulting fractured edge.

The plaster and mortar would need an extensive amount of varied but similar pieces with convex shapes. Therefore voronoi fracturing was chosen, as it provided the better performance, more procedurality and in general faster iterating to get the desired look. In order to create a more realistic look and avoid the artificial voronoi pattern some advanced techniques were used to adjust its results.

First instead of fracturing everything in one step, it was done in a layered approach. The first layer provided the ground fractures and set the possible maximum size for a piece. Then for the following layers only randomly selected pieces would get fractured further into a random amount of pieces, while a small percentage kept it's larger size. By this approach a wide range of different piece sizes and an overall randomness were achieved.

Another technique to improve the look of voronoi fracturing is called pre-fracture transform. Standard voronoi fracturing generally creates straight, unrealistic fractures. A way to solve this issue is to add noise to a surface before fracturing, then fracture the geo and afterwards transform each point back to its original position before fracturing. This helps breaking up the voronoi pattern and leads to more interesting and detailed shapes. This approach is limited in a way, because it can often lead to overlapping or broken geometry, especially when heavy noise is applied. There is a fine balance between breaking an unacceptable amount of geometry and achieving an interesting look. This issue didn't effect UEB010 as much, because when the wall is still unbroken, the prefactured geometry was used [Gress, 2014, p.422].

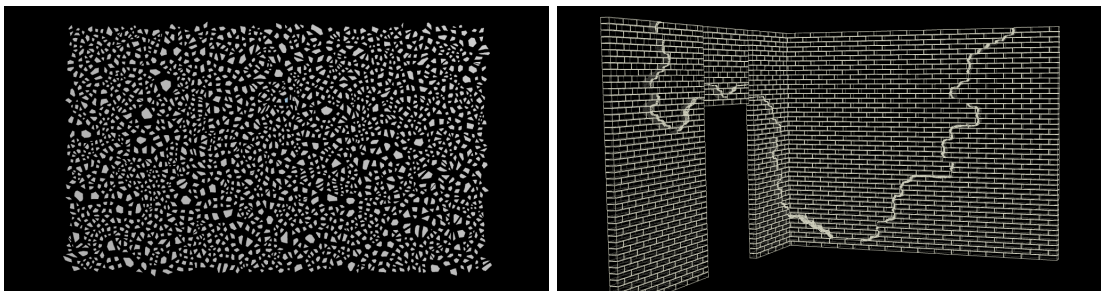


FIGURE 6.8: Fractured plaster(left) and bricks(right)

## 6.3 Simulation

After everything, from the scene, over the models, up to fracturing, has been prepared, it is time to start working on the actual simulation. Based on the R&D phase, previous experience and the desired look the general simulation approach should be determined regarding solvers, forces, constraints, collision and dependencies.

For UEB010 the requirements of the shot were established as the following:

The bricks are the main, leading element behind the destruction. They are influenced by supernatural, magic forces and affect every other element of the sim. The plaster and mortar are connected to the bricks and each other. When the bricks start moving, the connections for the plaster and mortar start to break and to a lesser extent they are affected by the same supernatural forces.

Through the destruction debris and fine dust is created, opposite to previous elements this should behave in a realistic way and not be affected by the supernatural forces. Instead it should move based on gravity and the movement of the other elements.

The exact approach to fulfill these requirements and create the destruction fx of UEB010 is described in the following sections.

### 6.3.1 Divide & Conquer

An important step when beginning to work on the fx is a proper division of the single elements into individual simulation steps similar to typical divide & conquer approaches in computer science. Although it would be possible to solve all effects in one big simulation, it is highly impractical, as it would need an enormous amount of computational resources and time. In the time needed to simulate everything at once, it would probably be possible to simulate multiple iterations for every single element when broken down. In addition the setup needed would be unnecessarily complex and confusing to work with.

The resulting best practice is to divide the simulation into logical steps, depending on simulation type and dependency on each other. The obvious division is based on simulation types. For example, simulating RBDs separately from fluids. It is important to follow the necessary order for the respective shot. For most destruction fx the RBDs are the leading element of the sim, but there are other cases for example especially powerful explosion, where it could make sense to simulate the fluids of the explosion first, affecting the RBDs in a second pass.

Further it is possible to break down the steps even smaller, when separating the different objects of the same simulation type. This can be done, when the motion of one object affects another object, but the same is not true in reverse, for example when there are massive mass or size differences.

For UEB010 there were 3 simulation types, RBDs, particles and smoke sims, further divided into a total of 6 simulation steps.

The first and most efficient sim was the RBD sim of just the bricks. Depending on the results of that simulation first the plaster and then the mortar was simulated, resulting in 3 individual RBD simulations. This way iterations on the leading brick sim only took minutes and allowed for lots of testing and tweaking. Once setup correctly the plaster and mortar then just needed to be run a few times with minor tweaks, as their movement and behavior was mainly influenced by the bricks. This saved hours of simulation time, because those secondary sims were a multitude more demanding regarding computation time.

Based on the results of the RBD sims, particle debris was then simulated separately. Because this was just a particle sim it was also very efficient and allowed fast iterating until the desired results for the debris was found. Then the particle sim was used as a source for the dust/smoke simulation, separated into 2 areas, inside the room and outside. This was done to easily create different conditions for smoke behavior outside and also save sim time, halving the initial simulated area. A huge advantage of this approach was also that the movement of the heavy dust sim could easily be iterated on through fast particle simulations.

### 6.3.2 Rigid Bodies

As described in the last section the rigid bodies were divided into 3 different simulations, bricks, plaster and mortar.

The first one just contained the bricks as active RBD objects and the set and a ground-plane as static, inactive collision geometry. A variety of forces were used to create the supernatural destruction and anti-gravity look of the simulation. The constraints used were 2 different kinds of simple glue constraints.

The second sim consisted of the plaster as an active RBD object, the set and a ground-plane as collision and most importantly the previously calculated brick sim as an animated static RBD object. This meant the movement of the bricks was not calculated again but used as an object for the plaster to interact with. Again a variety of forces were used to create a similar supernatural movement for the plaster. This time a mix of glue and hard constraints were used, to constrain the plaster to itself and to the bricks.

The mortar sim was setup very similar to the plaster sim, the only change was additionally to the result of the brick sim, the previous plaster sim was included as an animated static object as well. First planned as an important part of the destruction the mortar took a less prominent roll after the first few tests, because of its similar look to the plaster, when broken apart and also due to particles, dust and textures of the bricks partly taking over the mortars purpose.

For all sims the general physical properties like mass, friction and bounce where set consistently and based on previous researched real world values of comparable materials. Especially mass has an important impact on collision behavior. While bounce and friction affect the overall look and "energy" of the sim.

### **6.3.2.1 Forces**

The movements of the RBD sim were heavily art directed and a wide range of forces were used to control every object precisely.

The first general force starting the destruction is a subtle anti gravity force supported by a negative attractor force causing the rigid bodies to be pushed away from the room. After a few seconds a more distinct curve force takes over, that pulls everything outwards in a vortex like pattern. In addition, custom vector fields and various noise forces introduce general turbulence and randomness. The anti gravity forces increase and all the pieces fly up and spread out all over the sky. After the overall desired motion was achieved, various smaller forces were introduced, constrained to smaller, specific areas or objects, to control individual parts of the simulation until the desired specific look was achieved.

To create a more realistic, less artificial look, all forces were varied with subtle differences between each individual piece and in addition overall noise was added to most forces. Another important step that needed a lot of adjusting and fine tuning, but helped immensely in getting natural movements is fading forces in and out over a range of frames. Especially for the leading forces that operate in a consecutive order, smooth transitions are essential to avoid obviously artificial movements.

### **6.3.2.2 Constraints**

Constraints in Houdini define connections between individual pieces. They are created after fracturing to hold the geometry together, have properties like strength and can break dynamically if certain criteria are met, depending on the type of constraint.

For UEB010 a wide range of constraints were created holding the wall together and influencing the look of the whole destruction sim.

One constraint every piece was part of was an activation constraint. To control which parts of the wall would be active and be affected by the sim, which would stay the way they are and to dynamically control both groups, an unbreakable glue constraint was created between each piece of the wall. To begin the sim, this constraint would be gradually removed for the intended active parts of the bricks. For the secondary plaster and mortar sims, the movement of nearby bricks was used to determine if the activation constraint should be broken.

Most other constraints were used to connect the plaster and mortar to the bricks for the secondary simulations. For those custom force thresholds were introduced at which a constraint would break. Then different groups of strong or weak constraints were divided up randomly. Additionally to create interesting breaking patterns the constraints were stronger, the closer the pieces of plaster were to the middle of a brick. This way plaster over the gap between bricks would break faster than the plaster directly over the brick.

Same as with the forces a general overall randomness was created for all constraints by varying their strength or breaking threshold slightly between each piece or brick. This variance and minor differences between pieces is an import step in creating believable fx.

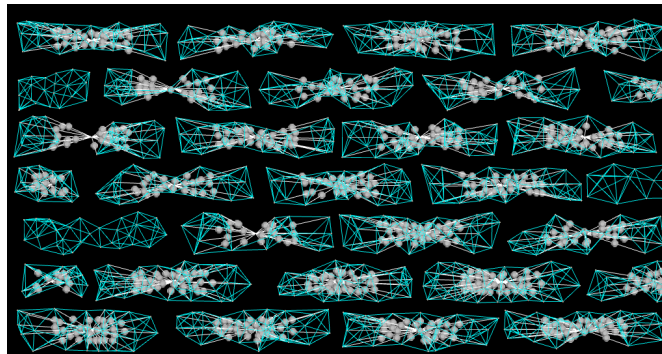


FIGURE 6.9: Constraints for the plaster simulation. The teal lines are glue constraints between plaster pieces, the grey spheres are hard constraints between plaster and bricks.

### 6.3.2.3 Replacing with high quality assets

When all RBD simulations are final and ready for rendering it is now time to replace the simple proxy geometry with the HQ assets to get the details necessary for the final rendering.

The bricks in UEB010 relied completely on displacement and normal maps to create surface detail. Nevertheless, the proxy bricks needed to be replaced with the textured asset, because of UV mapping and minor size differences. This was achieved by simply replacing the original bricks in the procedural wall setup with the new bricks. To get

rid of the size differences the brick asset was deformed to match the size of the proxy geometry.

The plaster was replaced with a high quality version as well. To get more realistic fractured edges and add detail to the overall look of the plaster, the original geometry was first subdivided to increase resolution. Then a setup was built to add different noises on top of the edge of plaster pieces but leave out the flat side facing the room. Additionally bevels were added to get a more interesting reflection behavior. Although the effect was minimal in the final shot, because of the distance to the camera, it helped to increase the overall realism of the shot.

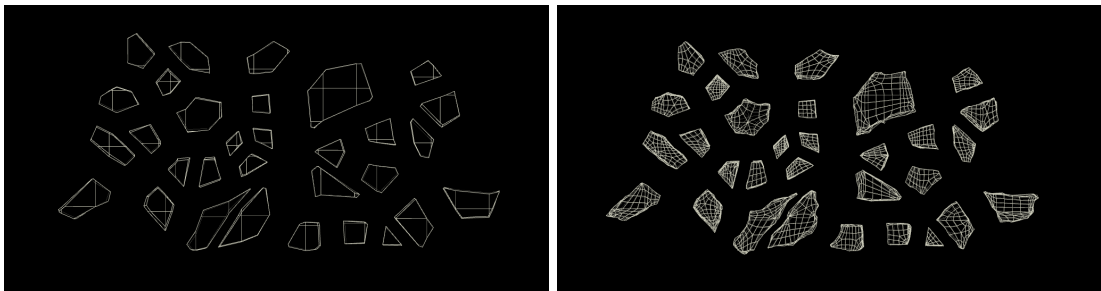


FIGURE 6.10: Exchanging sim proxies(left) with HQ geometry(right) for rendering

#### 6.3.2.4 Transfer to maya

After simulation, some post-processing and importing the HQ assets, the geometry was exported to Maya for rendering in V-Ray. This step in the pipeline lead to a massive amount of issues and consumed a lot of time unnecessarily.

The export from Houdini was done as transformed packed primitives, organized as individual pieces via a path attribute and saved in Ogawa as described in Chapter 5. This created a relative small file size and allowed organized and precise access for each piece in other applications. So far everything worked as expected.

Big issues came up when trying to import the created Alembic in Maya 2017 via the native importer. The import took an unreasonable amount of time and when finished lead to abysmal performance in the Maya interface, while the Maya log did not output any information relevant to the issue. Even after testing all available export settings in Houdini the problem still persisted, but only for Maya, whereas every other 3d application and even Nuke could open the alembic problem-free and with acceptable performance.

Further research revealed that the problem is quite common and related to the outdated alembic import functionality of Maya. It can be avoided by using third party software to handle the alembic import into Maya and most professional studios get around it by using their own alembic tool set. Those solution were not viable for UEB010 as neither

the software nor the manpower and knowledge required to implement the custom alembic import was available.

Eventually an acceptable solution was found by using the V-Ray proxy functionality of V-Ray. It allowed importing alembics as V-Ray proxies and all functions associated with them [Chaos Group, 2017]. This led to reasonable load times and interface performance. Although some issues still remained like long scene startup times when rendering and software crashes when selecting or operating on large sets of pieces.

### 6.3.3 Particle dust

After finishing the RBD sim, the result was used to create particle debris to add another layer to the destruction.

Although the particles are only a secondary effect and only appear subtly in the final image, they are a great supporting tool to improve believability and realism in the shot. In addition they can be used as the main element to source and direct the smoke simulation and therefore have a great impact on the final look of the shot.

#### 6.3.3.1 General behavior

As with most other setups creating variance and natural randomness among the movement of particles is an important step to achieve realistic motion. An easy way to realize this was by varying the mass and/or scale across all particles. This influenced the effect of each force on the particle and therefore created varying particle movements. To get a natural spread of randomness it was essential to avoid a linear distribution of mass and instead focus on many light, small particles and fewer bigger, heavy ones.

The complete RBD sim, the set and a groundplane were used as collision geometry to have dust collecting on the floor and on roughly flat surfaces, like some of the fractured edge and also the furniture.

To save simulation time, the particles were only sourced in activated areas of the sim. This was controlled by nearby moving active RBD geometry. An auto sleep function turned particles inactive, after there is no or only little movement in a certain timeframe, for example when lying on the floor for a second. Other particles falling down or moving geometry could activate them again if necessary.



### 6.3.3.2 Particle source

To create realistic particle debris the source of the particles needs to be realistic and based on the actual destruction happening in the shot. This can be done by calculating the distance between points on the surface of each individual RBD piece. A reference frame is set, pointing to a frame before any destruction happens. Then a solver checks the distance of the points on all pieces at each frame. As soon as the distance changes compared to the reference frame, destruction is determined and a debris source point is created on the affected piece.

The created debris source points have certain attributes and further move with their respective piece. This allows for some additional customization to control how particle debris is sourced. Generally the source points are deleted after a certain age threshold to stop the debris from continuously emitting even long after the destruction is over. Additionally the velocity of the source points can be calculated to use as a starting velocity for the debris particles.

For UEB010 the age threshold as well as the velocity was used to control the particle source and varied depending on particle and time. Because the velocity was pointing outwards of the room based on the RBD motion, it was mixed with a more prominent custom created velocity pointing inside the room and layered with noise to increase randomness. This combined with an animated and varying birth rate allowed for realistic debris sourcing.

### 6.3.3.3 Forces

For the particle debris mostly realistic forces were used to create believable and natural debris behavior, in contrast to the supernatural RBD forces and movements.

The main force driving the particles was a gravity force. To get the look of really small, floating dust particles it was decreased a lot compared to real life gravity. On top of the gravity, a turbulent noise force was added to particles moving faster than a certain threshold to increase the overall randomness of the debris.

Another essential force for most particle sims that was used for UEB010 as well is the drag force. It applies a damping to the overall velocity of all particles and generally has a stabilizing effect on the simulation, while keeping particles calm by taking out energy every frame.

Finally to recreate the behavior of fine dust and debris some clumping was added via attraction forces of particles to each other, again varying in strength and shaping.

#### 6.3.3.4 Post processing

Some minor post processing was done to the final result of the particle debris sim to prepare it for rendering in Mantra. While doing test renderings the particle scale was adjusted to the size wanted for the final look, leading to values below a pixel for most particles. The velocity had to be adjusted as well to get exactly the motion blur required for the shot.

To increase the overall particle count and also add detail and randomness, each particle was then replaced with a group of particles of varying spread and number.

The particles were then rendered as simple points in Mantra, with a basic material and the correct holdouts for compositing already applied.

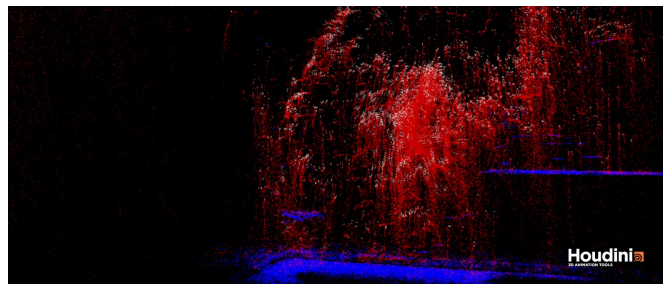


FIGURE 6.11: The result of the particle simulation: red particles are active, blue ones are sleeping and white ones are source particles

#### 6.3.4 Smoke dust

The smoke simulation was the final layer for the destruction fx of UEB010. The smoke look was supposed to be a mix between realistic movement and at the same time build the transition to the supernatural other side.

To ensure a reasonable resolution, save sim time and file space the smoke dust was divided into two separate simulations, one inside the room and one for the outside area. This way the inside sim could be simulated with the necessary higher resolution while keeping the dimensions small. On the other hand the dimensions of the outside sim could be made much bigger, but with a smaller resolution, due to the distance to the camera. Beside those dimension and resolution differences every other part of the simulations was identical between the foreground and background simulation.

##### 6.3.4.1 Smoke source

One of the most important controls over the simulation of a fluid is its source. The source of a volume adds densities and velocities, often as a starting condition into a fluid

simulation. It is essential for the behavior, look and final result of a sim and the more time is spent on refining a detailed and interesting source, the less time has to be spent tweaking the actual simulation.

The source for the smoke sim was created entirely by using the previously simulated particle debris.

Before building a fluid source from the particles was possible, some preprocessing needed to be done. The first step was isolating the most active particles by checking their velocity, age and other particle attributes. Only recently created particles with a high velocity should create smoke, to prevent debris from low impact or inactive parts of the destruction to create an unrealistic amount of smoke.

After selecting the viable particles, they are expanded by adding their own position at the previous frame. Then by connecting each particle pair and resampling the emerging polyline a continuous line of points representing each particles movement over the last frame is created. This is essential to avoid stepping or sub-sampling errors in the smoke source that appear when high velocities are present.

As a last pre-processing step the density and scale are adjusted based on the age and lifespan of the particles. The younger a particle and therefore the more recent the destruction it was birthed from is, the bigger and more dense the particle is made to create a bigger amount of smoke from it.

There is a prebuilt fluid source tool in Houdini that creates a source volume made exactly for use cases like this with additional functionality to further control the source created. Unfortunately it uses the inefficient volume data type instead of the more performant VDBs. Because the source resolution for UEB010 needed to be very high and several iterations needed to be done, the cost of using the fluid source tool would have been excessive.

Instead the source volume was created by copying the attribute values of the particles into a VDB, based on their particle scale, which is more efficient, but lacked most of the control functionality of the fluid source tool. The performance advantage of the VDB workflow was big enough to justify the lost functionality and because the particles were already highly pre-processed there was still enough detail in the source created by them.

Another field that can be sourced into a fluid besides density or velocity is divergence. Divergence specifies areas where rapid expansion inside the fluid is happening. This can be used to break up otherwise uninteresting smoke sims and generally add some energy and explosiveness into the sim. The original fluid source multiplied with some noise, leaving only small patches of density data was used to drive the divergence field of the fluid sim for UEB010.

#### 6.3.4.2 Microsolvers & general behavior

The general simulation work, calculating velocity and other fields or moving density around is done by the pyro solver. Some control and customization options are part of its functionality. To get a more in depth control over the fluid sim it is necessary to use microsolvers.

For UEB010 the three most common types of microsolvers were used, disturbance to create small scale detail, turbulence to add large scale movements and dissipate to recreate a natural density falloff.

Disturbance creates small amounts of change in small localized areas. This leads to a more interesting, more detailed look especially at the outside edges of a fluid, without changing the overall shape or motion of the sim. A useful technique to increase the amount of detail added, is to add various disturbance microsolvers with varying scale. This way the multiple layers of disturbance detail add up to an overall increase in the realism and look of the sim. To avoid unrealistic changes and detail in calm and inactive parts of the simulation, the amount of disturbance can and should be scaled by either density or velocity of the fluid.

The Dissipate solver controls how much the density diffuses into neighboring voxels over time and how much evaporates every second. This is used to recreate real world behavior and avoid artificial buildup and detail in fluids. In the case of UEB010 the diffusion and evaporation rates were kept low for the first few seconds to get a detailed and defined debris smoke in the confined space of the room. As the room breaks open increasingly and more outside influences affect the volume, the dissipation rates increase and the smoke quickly blends with the outside environment to open up the view outside.

The Turbulence solver creates a global turbulence field to add an overall noise on the velocities in the sim. As other microsolvers are often more sophisticated, regarding small scale detail, this is ideal to introduce subtle large scale details into a sim, representing for example winds and air flows affecting the fluid. Same as for the Dissipation the influence on UEB010 was first kept low, as long as the fluid was confined inside the room. Then as soon as the wall opens up more, the turbulence was blended in more and more.

#### 6.3.4.3 Forces & collision

For most fluid sims forces only play a secondary role behind the more prominent density/velocity sources and microsolvers. But collision objects, especially for fluid sims that are based on RBD destruction, play another important role. Through collision fluids

can be blocked to enter certain areas and velocity can be transferred to the fluid from objects moving through it.

Because collision calculation can be complicated between geometry and fluids, it is easier to convert the collision geometry to a volume before simulating. There are different approaches, and as previously explained the most efficient one is working with VDBs. Simply converting the collision geometry to a VDB, using a reasonable resolution, saves disk space, calculation time and memory during simulation. Most of the time, a resolution noticeably below the resolution of the fluid sim should be enough, except in cases with very intricate and detailed geometry relevant to the collision.

For UEB010 the collision VDBs consisted of the RBD sim, the set and further support geometry created to fill gaps in the wall, not seen through the camera, but affecting the simulation. In addition collision geometry was created for the actress moving through the smoke. This was done using the actual alpha from comp, putting it on an image plane in front of the camera, extruding the alpha to get a rough human shape and converting it to VDBs as well.



FIGURE 6.12: The combined collision geometry used while simulating the dust to create the necessary interactions with the set and the actress.

Ultimately, the forces used to support the primary effects on the fluid sim were simply a weak gravity force keeping the smoke on the ground and a subtle drag to help calm down the whole simulation.

#### 6.3.4.4 Smoke post processing

After the simulation some post processing is necessary to prepare the smoke dust for rendering. The first step lies in choosing which fields to export from the sim, as most fields created for simulation are not needed for the actual rendering. For smoke sims the most important one is usually the density field and depending on if motion blur will be calculated and noise textures will be used, the velocity and the rest fields have to be saved as well.

As stated previously, Houdini calculates fluid sims as Houdini volumes internally and therefore the output of the sim is a volume as well. For UEB010 the two divided smoke sims were converted to VDB and saved separately to work more efficiently.

To add even more detail to smoke without any additional simulation effort a 3d noise texture can be used when rendering to vary density values across the volume. Although not physically correct it increases believability by seemingly adding resolution and detail to the smoke.

### 6.3.5 Optimization & cleanup

During post production, when there are tight deadlines and a high workload it's especially important to work as efficient as possible. Fx work in particular is very time sensitive. Each iteration or small change can take hours and days to implement and even then it's just the first step in a very long pipeline of other tasks that often depend on the fx to continue. This means scene optimization and a speedy workflow are essential when creating fx in a production environment. Although there is no definite way to optimize a scene, there are some common measures that should be considered.

A first step to optimizing sim times is dividing up big sims into several smaller ones, as already discussed in the beginning of the section. The time saved by this is crucial and probably bigger than most other optimization measures.

A lot of time can be wasted by simulating details that won't be visible or noticeable at all. This is true for spatial as well as temporal resolution.

In case of temporal resolution at least two things can be optimized. Most simulations are calculated in substeps, they are the amount of timesteps the solver calculates per frame. While an increase of substeps can improve the look of a simulation considerably, especially when there are high velocities involved, the point of diminishing returns kicks in rapidly. Furthermore the increase of substeps correlates directly with the increase of simulation time. Meaning, doubling the substeps leads to a simulation time twice as long, while the actual improvement in quality probably ends up a fraction of that. In conclusion substeps should always be balanced with their actual benefit close in mind.

The other temporal concern is more obvious but nevertheless not negligible. In a shot with several destruction fx and various different simulations, each sim should be closely simulated for exactly the time it is going to be part of the shot and not longer. And even inside each individual simulation this approach can help by deactivating pieces of an RBD sim or individual particles of a particle sim that have stopped moving or are out of the frame. This way computational time can be saved without having any visible difference on the result of the simulation.

Optimizing spatial resolution saves disk space and more importantly simulation time as well. In case of RBD or particle sims this means only simulating the amount of pieces or particles necessary for a believable effect, in case of fluids it is the amount of voxels. Especially for secondary or background effects, which will end up blurred in the distance anyway, but sometimes also for hero sims, less can be enough and save tremendous amounts of simulation and render time, which can then in turn be spent on more iterations, leading to overall improved results.

Reducing disk space and file sizes is a big part of an efficient workflow. There are various ways to achieve this. When working in Houdini in general, it is important to keep an eye on the geometry attributes. Often they accumulate from operation to operation and are dragged along, even if there is no use for them anymore. To ensure an efficient workflow and especially when saving geometry all unnecessary and outdated attributes should be deleted. For geometry containing many individual entities with attributes, for example particle sims or high res fluid sims, this can make a huge difference and even in other cases it at least helps with overall clarity of the scene.

Another way to minimize file sizes is especially relevant for RBD sims. When saving out the result of a rigid body, instead of the whole geometry it's possible to save out just a point representing each piece, containing all necessary data like position, orientation, velocity and more. On import those points can then be replaced with the original geometry again, leading to exactly the same result as the original simulation output, but with smaller file sizes and distinctly shorter read/write times.

Finally a practical technique to optimize the fx workflow especially at the start of the process is appropriate culling of elements. For example in UEB010 instead of simulating the complete wall for each iteration, only an area of maybe 1 m<sup>2</sup> was selected. To judge the general behavior and also constraints, forces, etc. this was more than enough and lead to quick iterations. For most testing purposes a culled scene will be sufficient and the procedural nature of most fx allow for an easy switch to the full scene as soon as the right settings are dialed in.

## Chapter 7

# Evaluation

After rendering and compositing the simulations created in Chapter 6, this was the final result of UEB010 as shown in the movie:

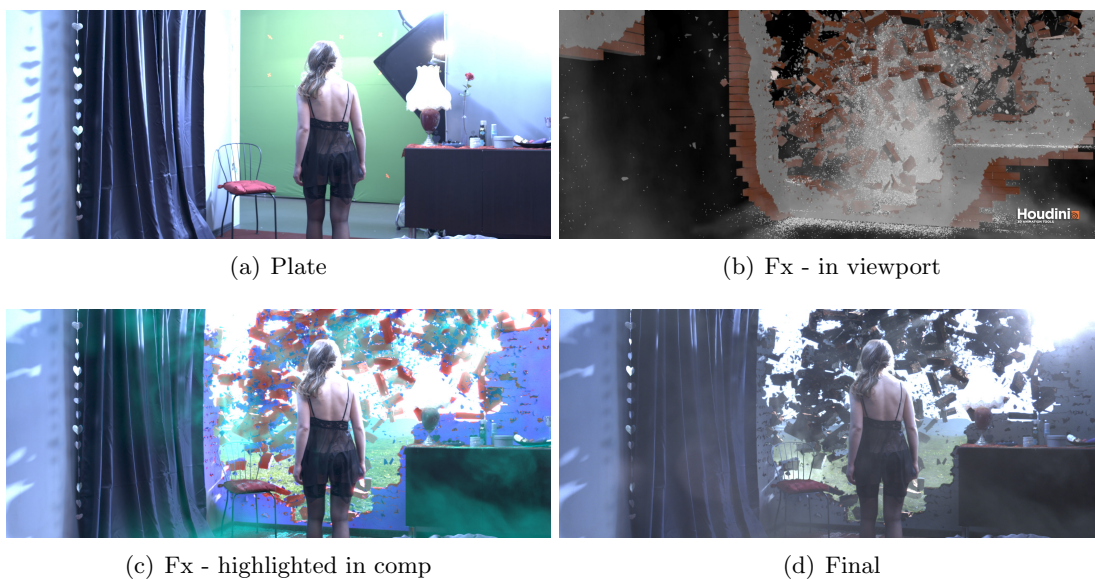


FIGURE 7.1: UEB010 Breakdown

### 7.1 Shot review

The final result shown in the movie fulfilled all requirements set by the team. Supported by an emotional score and excellent editing, the transition achieved the intended impact and showed impressive visuals. Nevertheless, there is also still some room for improvements as some steps of the creation process didn't go as well, as they were planned. This section will give a summary over some difficulties encountered during post-production, how they could have been avoided and possible measures to improve the shot further.



The biggest difficulties and issues which resulted in a loss of quality for the final outcome can be roughly divided into three categories: The first being hardware and pipeline limitations, secondly inefficient workflow and last but not least missing creative direction.

The first category of issues consisted of various limitations regarding the hardware and pipeline available. One of the main limitations for the simulation were the available computers. Although around 30 workstations were available, most of them were intended as render slaves and not sufficiently equipped for simulations because of their limited RAM. Only 11 workstations were powerful enough for complex simulations and even then their RAM limit was reached during fluid simulations. Unfortunately these workstations were almost consistently in use by other artists, which meant, that at most 1 or 2 machines were available for fx work.

In addition the render manager used during production, RenderPal, lacked a functioning Houdini integration, which would allow simulations to be sent to the "farm" of idle PCs and worked on over night. This lead to working on the fx on one computer and manually starting simulations of on a second one. Only individual iterations could be made instead of the actual production practice, where a number of differing simulations are sent to the farm and afterwards the optimal parameters can be picked from the best looking result. It also meant that constant manual supervision was necessary, to start new simulations as soon as the previous was done.

Although all PCs could be used for Mantra rendering and enough licenses were available, there was still no Houdini pipeline setup and the issue with RenderPal was even more significant for 3d renders which require a lot more management to efficiently make use of a pool of computers.

A lot of the hardware restrictions were countered by scene optimizations and setting up simulations as efficiently as possible. In this way it was possible to achieve complex, high quality results regardless of the difficulties. Nevertheless, the hardware and pipeline restrictions made progress a lot slower and added a significant amount of manual simulation and render management work, that could have been avoided if a proper Houdini pipeline had been set up beforehand.

The second category of issues were caused by an inefficient workflow, based on the software used for different tasks of the shot. Because of the missing Houdini pipeline discussed previously and because other 3d artists in the team were more comfortable in using Maya and V-Ray, it was decided early on to use Houdini only for the simulation and then export everything to be rendered with V-Ray in Maya.

As described in Chapter 6 the exchange between applications caused several issues and cost a lot of time to resolve through testing and extensive research. Additionally, new problems arose when trying the same procedure for particles and the fluid sim. Because

there wasn't any time left for elaborate tests and because Mantra as a renderer is optimized for these use cases, the solution was to stay in Houdini and render those elements with Mantra. This was all decided on very short notice and there was almost no time to actually tweak lighting and render settings for the particles and smoke renders.

To address this issue in combination together with the first category: Setting up a solid Houdini & Mantra pipeline would have helped immensely in this case as well. The time spent troubleshooting Alembic, VDB and particle import/export could have been better used for more fx iterations and polishing render settings. Since Mantra is a very capable renderer it may have even been possible to render everything, including the rigid bodies inside of Houdini, which could have improved the quality of the rendering significantly.

Having discussed the biggest technical difficulties of the shot, its most significant issue originated during the conceptional stage. From beginning on the sole purpose of the simulation shot was to create an interesting transition between the brothel room and the dreamscape by using impressive visual effects. Beyond that, the creative vision was seemingly unclear and only few directions were given regarding the final look of the shot.

On one hand this allowed for a lot of creative freedom and possibilities to experiment with different behaviors. On the other hand it became an issue when trying to finalize the first rigid body simulation layers. Countless variations with minimal differences had to be created, before the final version got approved. Spending this amount of time on the comparably less complex RBD simulations meant a significantly shorter amount of time available for the particle and fluid simulations. This led to a distinct loss of quality due to missing time for iterating tweaking and polishing the fluid.

When working at a visual effects facility in a true production environment the issues regarding hardware, pipeline and workflow can be mostly dismissed or at least are massively reduced. But a takeaway from the persisting creative issue is that as an fx artist it is important to ask for a clear direction with defined requirements before working on a shot. Furthermore it is important to regularly seek feedback from supervisors and the client/director. This way the expectations towards a shot are clear from the beginning and upcoming issue can be addressed quickly.

## Chapter 8

# Conclusion

This thesis gave a broad introduction to simulations used in visual effects. A special focus was laid on destruction effects, one of most common simulations appearing in many current movies.

First, simulation was described as physically-based animation and some differences and advantages towards conventional animation were presented including the ability to efficiently calculate realistic or more importantly believable motion for a multitude of individual objects without having to set any keyframes. To get a better understanding for the different elements of destruction effects and their simulation types a classification was given and general use cases were explained.

After a common understanding of simulations in visual effects has been established, the underlying principles behind their calculations were explained, including a reintroduction to the fundamental second law of motion by Newton and its application for different simulation types. This also included informations about the general concepts of numerical simulation with a focus on Euler methods.

The basic properties and algorithms of particle systems were explained and a simple form of particle-plane collision calculation presented. This system was then expanded by rotational components and momentum for rigid body dynamics. Also the more advanced collision detection and response algorithms were introduced, which are used to test for interpenetration, find the exact moment of collision and calculate the appropriate impulse responses.

Finally, an overview over computational fluid dynamics was given, breaking down the incompressible Navier-Stokes equations, separating the grid-based Eulerian model and the particle-based Lagrangian model and ending with a bare-bones, simplified fluid solver.

The second, more practical part of the thesis first described the production environment in which the destruction fx were created. The student production "The Girl Beyond",

and the story and team behind it were introduced and then the specific simulation shot explained. Afterwards, the used software was presented, mainly the 3d application Houdini, which is commonly used for fx work, focusing on its functionalities related to simulations like the dynamic context and the included solvers. Also, some informations about Alembic a popular 3d interchange format were given.

Finally, the implementation of destruction fx for an actual shot were step-by-step explained in depth. First the whole process of pre-production and planning was described, focusing on the importance of a clear shot design, sufficient testing, R&D and starting as early as possible with actual fx work. Then the necessary preparation for an efficient simulation workflow were presented, including how to setup the scene, important considerations for fx modeling, advanced fracturing techniques and most importantly the efficient division of simulations.

For each simulation type the chosen physical properties, forces and behaviors were defined, while for the particles and fluid the importance of an interesting source was emphasized. Additionally, various post-processing techniques used to increase the overall quality of the end result were shown. The implementation then ended with a look into a range of optimizations to ensure an efficient workflow by reducing computational time and file sizes and is rounded off with an evaluation of the significant issues encountered during production and possible improvements for the end result.

In conclusion this thesis isn't intended as a definite guide for destruction simulations, but instead gives a broad overview over the subject area, including the most relevant topics, and offers insight on how to approach simulations for actual use in a production environment.

# Bibliography

- [Baraff 2001] BARAFF, David: Physically based modeling: Rigid body simulation. In: *SIGGRAPH Course Notes, ACM SIGGRAPH* Bd. 2, 2001, p. 2–1
- [Box Office Mojo 2017] BOX OFFICE MOJO: *Worldwide box office results, 1989-Present*. 2017. – URL <http://www.boxofficemojo.com/yearly/?view2=worldwide&view=releasedate&p=.htm>. – access date: 2017-07-25
- [Bridson 2015] BRIDSON, Robert: *Fluid Simulation for Computer Graphics*. 2 Rev ed. Boca Raton : Apple Academic Press Inc., September 2015. – ISBN 978-1-4822-3283-7
- [Bridson and Müller-Fischer 2007] BRIDSON, Robert ; MÜLLER-FISCHER, Matthias: Fluid simulation. In: *ACM SIGGRAPH 2007 courses*, ACM, 2007, p. 1–81
- [Chaos Group 2017] CHAOS GROUP: *V-Ray Proxy - documentation*. 2017. – URL <https://docs.chaosgroup.com/display/VRAY3MAYA/Import+V-Ray+Proxy>. – access date: 2017-08-07
- [Coumans 2015] COUMANS, Erwin: Bullet Physics Simulation. In: *ACM SIGGRAPH 2015 Courses*. New York, NY, USA : ACM, 2015 (SIGGRAPH '15). – URL <http://doi.acm.org/10.1145/2776880.2792704>. – ISBN 978-1-4503-3634-5
- [Fisher 2004] FISHER, John: Visualizing the connection among convex hull, Voronoi diagram and Delaunay triangulation. In: *37th Midwest Instruction and Computing Symposium*, 2004
- [Gress 2014] GRESS, Jon: *[digital] Visual Effects and Compositing*. 01. San Francisco, California : New Riders, 2014. – ISBN 978-0-321-98438-8
- [Museth 2013] MUSETH, Ken: VDB: High-resolution Sparse Volumes with Dynamic Topology. In: *ACM Trans. Graph.* 32 (2013), Juli, Nr. 3, p. 27:1–27:22. – URL <http://doi.acm.org/10.1145/2487228.2487235>. – ISSN 0730-0301
- [Okun 2010] OKUN, Jeffery: *The VES Handbook of Visual Effects: Industry Standard VFX Practices and Procedures*. Burlington, MA : Focal Press, September 2010. – ISBN 978-0-240-81242-7

- [Parent 2012] PARENT, Rick: *Computer Animation: Algorithms and Techniques*. 3 Rev ed. Amsterdam : Elsevier Science & Technology, August 2012. – ISBN 978-0-12-415842-9
- [Reeves 1983] REEVES, W. T.: Particle Systems - a Technique for Modeling a Class of Fuzzy Objects. In: *ACM Trans. Graph.* 2 (1983), April, Nr. 2, p. 91–108
- [SideFX 2017a] SIDEFX: *Alembic files - documentation*. 2017. – URL <http://www.sidefx.com/docs/houdini/io/alembic>. – access date: 2017-07-28
- [SideFX 2017b] SIDEFX: *Boolean - documentation*. 2017. – URL <http://www.sidefx.com/docs/houdini/nodes/sop/boolean>. – access date: 2017-08-04
- [SideFX 2017c] SIDEFX: *Rigid body dynamics - documentation*. 2017. – URL <http://www.sidefx.com/docs/houdini/dyno/rbd>. – access date: 2017-07-27
- [Stam 2015] STAM, Jos: *The Art of Fluid Animation*. Boca Raton, Florida : Taylor & Francis Ltd., 2015. – ISBN 978-1-4987-0020-7
- [Witkin 2001] WITKIN, Andrew: Physically based modeling particle system dynamics. In: *ACM SIGGRAPH Course Notes*, 2001