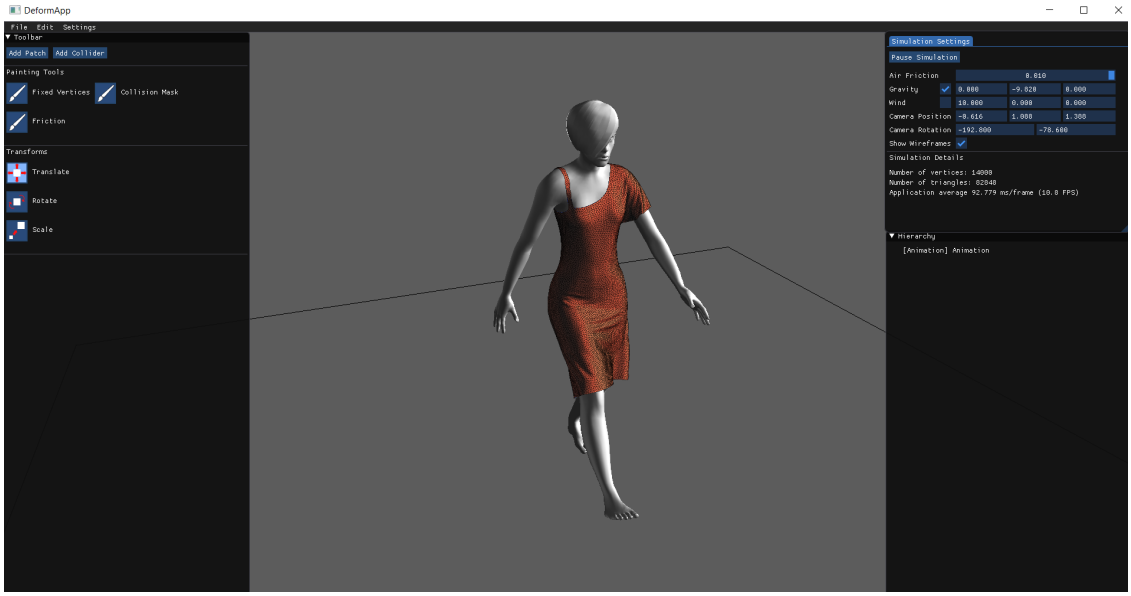

An Interactive 3D Modeling Tool for Creating and Managing Soft Body Animations

Master's Thesis in Interaction Design and Technologies

Razmus Strandell

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

Master's thesis 2020

An Interactive 3D Modeling Tool for Creating and Managing Soft Body Animations

Razmus Strandell

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2020

An Interactive 3D Modeling Tool for Creating and Managing Soft Body Animations
Razmus Strandell

Supervisor: Marco Fratarcangeli, Department of Computer Science and Engineering

Master's Thesis 2020
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The tool developed in this project, shown with a typical scene containing an animated character wearing a dress.

Typeset in LaTeX
Gothenburg, Sweden 2020

An Interactive 3D Modeling Tool for Creating and Managing Soft Body Animations
Razmus Strandell
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Computer animations are commonly deployed in various fields today, such as the movie industry, video game industry and in architecture. This has led to an increase in the demand for 3D modeling tools that can be used to create these animations. The goal of this project is thus to explore what considerations exist when developing interactive 3D modeling software.

This was explored by conducting researching in the area, including existing 3D modeling tools and other related work, and by designing and implementing an interactive 3D modeling tool as per the research through design practice. This tool was built around generating generic soft body animations. The tool was first designed using an iterative design process, including methods ranging from ideation techniques to evaluation methods. The software part was then developed using a mix of popular agile software development methods. The code was developed with focus on ensuring optimal performance, promoting software modularity and minimizing external dependencies.

The end result is a high fidelity software prototype that can be used to create animations of varying complexity. The development of the tool, alongside the conducted research, resulted in a set of guidelines to consider when developing similar tools. These guidelines can be summarized as the following: what external libraries to use, what work process to use, what features should be available, how to design for different user groups and how to make the code modular. Despite the guidelines resulting from the work of this specific tool, they are considered to be applicable to the development of 3D modeling software that handles any kind of soft body animations.

Keywords: 3D modeling, computer animation, soft body animation, user experience

Acknowledgements

I would like to thank my supervisor Marco Fratarcangeli for his constant help and guidance throughout this project. I would further like to thank the other associates at Deform Dynamics for helping out with various technical issues that arose during the project. Finally, I would like to thank the people that participated in the user tests and supported me in other ways.

Razmus Strandell, Gothenburg, June 2020

Contents

List of Figures

1
Introduction

Computer generated imagery (CGI) is the art of applying computer graphics to create graphics for some medium. Today, CGI can be found everywhere; in video games, movies or even commercials. In movies, CGI has been able to achieve close to photo-realistic results for a while now. However, new techniques are still being developed with the aim to produce similar effects at interactive rates, for example to be used in video games and other interactive applications [1]. CGI can be divided into one of the following categories: 3D modeling, computer animation and rendering.

1.1 Background

In 1960, the first film using computer animation was released in the form of a 49 second 2D vector animation [2]. A decade later, in the early 70's, various techniques for doing 3D animations were developed at the University of Utah. By the mid-70's, the first movie using computer animation was released marking the beginning of computer animations in public media [3]. In the early 80's, physics-based animations started to emerge in mainstream media in movies such as Star Trek 2: The Wrath of Khan [4].

Physics-based animations is a sub-field within the computer animations field that focuses on achieving visually plausible physics effects at interactive rates [5]. This includes effects such as gravity and collisions. In general, there are three different kinds of objects to consider in physics-based animations: rigid bodies, soft bodies and fluids [6].

A rigid body is a solid object that is not deformable, which means that the shape of the object can not change. This means that all points on the object are constant in relation to each other despite potential external forces [6]. *A soft body* is a solid object that is deformable, which means that the shape of the object can change. In other words, all points on the object are acting individually and thus the relative distance between two points can change [6]. *A fluid* is a non-solid object, for example a liquid, gas or plasma. In comparison to a soft body, a fluid is not expected to retain its shape [6].

While the simulation of rigid-bodies is easy, soft-bodies are much harder because of the vastly increased number of possible states for the object. Still, physics-based

animations involving soft-bodies are used heavily in video games and movies for special effects. This results in it being an active research area dedicated to different computation methods for approximating them. The rapid evolution of hardware has further promoted this area; computations that were hard to do only a few decades ago, such as animating a moving character, are now considered trivial [7]. This has expanded the use of physics-based animations to many other kinds of applications, such as surgical simulation systems and computer aided design software.

1.2 Research problem

With the increasing need of having plausible animations in different kinds of applications, there is also an increasing need for tools that facilitates their creation. An animation designer should be allowed to create and manage them without requiring much technical knowledge. Currently there are many different applications fitting that description, each with its own set of features and shortcomings.

Deform Dynamics have developed a software development kit (*SDK*) for creating plausible soft-body animations based on the Vivace solver developed by Marco Fratarcangeli, Valentina Tibaldo and Fabio Pellacini [8]. This solver produces stable animations at an extremely fast rate, making it suitable for interactive applications. The goal of this project is thus to research how interactive 3D modeling tools should be developed in order to meet those needs.

1.3 Research question

What to consider when designing and implementing an interactive 3D modeling tool for creating and managing generic soft bodies?

The overall goal of this project is to allow for easy creation and managing of plausible soft-body animations. The goal of the thesis work is to design and implement a tool for doing this. Since creating a full-fledged 3D modeling tool is a complex task, usually requiring years and years of work, the expected result is a working prototype that can reproduce the demos that Deform Dynamics previously have created.

1.4 Related Work

There are various software applications existing today that are used for creating and managing animations. Some of these are very generic, allowing users to create a vast amount of different kinds of animations, while some are more specialized in certain kinds of animations such as cloth animations.

Autodesk Maya, or simply *Maya*, is an application for creating 3D graphics and animations [9]. Maya is very generic, allowing the user to create virtually any kind of animation that can be incorporated in a vast amount of different applications.

It provides the user with a large number of tools and features to generate these animations. One of the main advantages of Maya compared to other tools listed here is character animation.

CLO3D is an animation creation tool specifically targeted for visualizing true-to-life 3D garments [10]. The main feature of CLO3D separating it from other tools listed here is the ability to visualize a piece of clothing from cloth patches by "stitching" them together.

Houdini is a 3d animation software that specializes in procedural generation [11]. Houdini requires no external downloads for visual effects etc. It is considered harder to learn for new users than other tools listed here, requiring more technical knowledge amongst other things [11].

3DS MAX is a generic 3d animation software [12]. 3DS MAX requires animations to "bake" before visualising, compared to live feedback received from some of the other tools. There are plenty of plugins available for 3DS MAX, providing various features and assets. It is only available for Windows operating systems.

DeformPlugin is a plugin for Unity which gives projects access to the various features provided by the Dynamo SDK. The plugin specializes on plausible soft-body animations, with much functionality for simulating cloth. It offers a few different features, including stitching patches of cloth together and different kinds of collisions [13].

1.5 Delimitations

The Dynamo SDK which this tool is built upon is targeted at creating animations involving different kinds of soft bodies. One of its main features, however, is the ability to produce plausible physics-based animations with cloth. This means that the resulting tool will be based mostly around creating and managing that kind of soft-body animations. Generating the internal functionality such as the actual animation, collisions and so on is functionality provided by the Dynamo SDK and will thus not need to be considered when creating this tool. Interactive 3D modeling tools can be incredibly complex, giving the user a large amount of possible actions to perform. This tool, however, will focus on some key features while leaving others out such as stitching patches together.

2
Theory

This section describes the research method which is the basis for the project, as well as theories relevant to the design process.

2.1 Research Through Design

Research through design is the practice of developing a design as a means to solve a research problem. Compared to traditional scientific fields, the aim is not to produce falsifiable theories; rather it is about producing general guidelines that can be used solve similar design problems. [14]. The guidelines are derived from the knowledge gained from the process of developing the design.

2.2 Graphical User Interface Design

Graphical user interface design refers to the design of a graphical user interface, commonly referred to as a *GUI*, that will be used to allow user interaction with a software application. There are a few goals, or metrics, that should be considered when designing a graphical interface: the product is effective to use, efficient to use, error free, easy to use and enjoyable [15]. In order to fulfill these criteria, a designer must strive to promote user flow and reduce *excise*.

2.2.1 Excise

Excise is work that the user of a product has to do that doesn't directly relate to their goal. The different kinds of excise can be categorized as either navigational excise, skeuomorphic excise, modal excise and stylistic excise [16].

Navigational excise refers to navigation that doesn't directly relate to the user reaching their intended destination. Examples of this is having to open up a menu to click on an intended option or to switch between different tabs and/or windows to perform some desired actions [16]. *Skeuomorphic excise* is excise resulting from design choices based on mimicking the behavior of a physical object even when they don't serve a purpose. Examples of skeuomorphic excise are "coffee stains" on a virtual paper in a notebook application and a calculator application adapting the look of a physical calculator [16]. *Modal excise* is unnecessary pop-ups that disturb the users flow. These types of pop-ups can be error messages, notifications, confirmation dialogs etc [16]. *Stylistic excise* is excise caused by the user having to do much visual

decoding to find what they are looking for. Examples of this is when the screen is cluttered with many elements that aren't well separated visually [16].

2.2.2 Application Posture

An application posture defines the nature and characteristics of an application [16]. The posture of a product is related to how it presents itself to the user. How much of the users attention will the application require? How does the product's behavior respond to the attention given by the user? To properly define the posture of an application, it is thus necessary to know the context and the environment in which it will be used [16].

Sovereign Posture

A sovereign posture is a posture suited for applications that are designed for being used for long periods at the time [16]. Examples of sovereign posture applications include word processors, spreadsheets and content-creating tools. The following are common characteristics for a sovereign posture application: Sovereign applications generally *target intermediate users*. The aim is to make sure that new users turn into intermediate users fast. Expert users should be given settings and mnemonics to perform advanced features. Sovereign applications also *use a minimalistic visual style* with discrete colors. The reason for this is that users tend to stare at the application for long periods of time, making bright colors annoying in the long run. The long-term use further allows elements such as toolbars and controls to be smaller than normal, as users tend to get an innate sense of where objects are located. Another characteristic is that they *allow rich input from the user*. This means that any aspect of the application that is interacted with frequently should be able to be controlled in various ways. For example, direct manipulation of objects should be allowed. Fine-tuning parameters and having mnemonics and keyboard accelerators is encouraged. Toolbars consisting of many different tools also works. In most cases, having a sovereign posture means that it is a *document-centric full screen application*. The application should by default be launched in full-screen mode, and the GUI should be optimized with that in mind. The main content which the user interacts with should be placed in the center of the screen. Any documents in the application should be full-screen as well. The final characteristic for a sovereign application is that it *provides rich modeless feedback to the user*. Modeless feedback refers to visual feedback that is incorporated into the GUI instead of having pop-ups or other disturbing elements.

Transient Posture

A transient posture is a posture suited for applications that are designed for being used briefly [16]. Examples of transient posture applications include calculator apps, video recording apps and weather apps. The following are common characteristics for a transient posture application: A transient application should use a *simple and clear interface*. This is because they tend to be used for short periods of time, making users become less familiar with it compared to a sovereign application. Controls

and other interface elements should thus be big and pliant while also using bright, saturated colors. Transient applications are also *limited to a single view and provide access to all functions immediately.* It should be intuitive and quick to use. Finally, *user choices are remembered by the application.* This includes position on the screen and window size.

3
Methodology

This chapter lists different commonly used tools and methods relevant to the project. They have been categorized as either interaction design tools, interaction design methods, software development tools or software development methods.

3.1 Interaction Design Methods

This section describes the different design processes and methods relevant to the design phase of the project. Listed methods range from ideation techniques to evaluation methods.

3.1.1 Design Thinking

Design thinking is a methodology for generating solutions to design problems. According to the design thinking model proposed by the Hasso Plattner Institute of Design at Stanford, it consists of the following five stages: empathizing, defining, ideating, prototyping and testing [17].

During the *empathizing stage*, designers gather knowledge of the problems they are trying to solve through empathy. The focus is about getting insight of user needs by setting aside their own assumptions about the domain. The *defining stage* is where user needs and other data gathered in the previous stage is used to define the problem statement. This stage is where things like necessary features and functions are formulated. The *ideating stage* is where the actual concept ideas are generated using the information gathered in both previous stages. The goal of the ideating stage is to generate a concept that can be used to solve the defined problem statement. During the *prototyping stage*, prototypes of the concept are built. The main purpose of these prototypes is to test and evaluate a concept or a design early in the design process. The final stage is the *testing phase*. Here, the prototypes are tested and evaluated [17].

3.1.2 Iterative Design

Iterative design is a design process where designers work iteratively in a possibly endless process [18]. When applied by the design thinking methodology, going through all five stages would count as a single iteration. That means that once the testing phase is finished, designers start over from the defining stage. The goal at that

point is to use the feedback received from the testing stage to define new problems. Moving on to the ideating stage, solutions to the newly emerged problems are developed. The solutions are then adapted to a new prototype, which is then tested and evaluated. Since designs can usually be improved endlessly, a third iteration might begin now starting once again from the defining stage. However, limited resources and deadlines regarding product release usually tends to stop the design process at some point and treat the design as finished. An iteration does not necessarily go from the testing stage to the defining stage all the time. For example, both testing and prototyping could result in new ideas which returns the design process to the ideating stage. In some cases, testing could also lead to the designers getting new valuable insight of their target group. This could result in the next iteration starting from the empathizing stage.

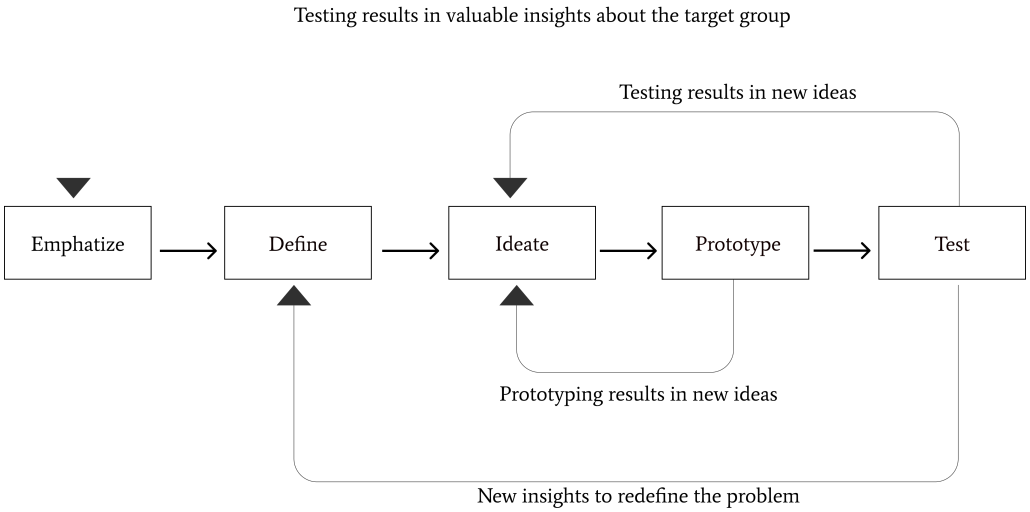

Figure 3.1: The iterative design process with design thinking

3.1.3 Competitor Analysis

Competitor analysis is a method which is based on analysing certain behavior of a competitor [19]. A competitor doesn't necessarily refers to someone within the same field; it could simply be that some functionality is shared. There are several reasons for doing a competitive analysis. A main one is that there is often much to be learned from successful products, as usually lots of resources have been used to develop their design. Another is to get some insight in what is currently working well, and what is not. This can provide an opportunity to give the product an edge over competitors. It is important to remember that a design solution should never be copied without any motivation. The purpose is to find inspiration and understand why something is designed in a particular way so that it can be used to favor your own product. Competitor products might also work as a reference when developing your own features. A full competitive analysis analyzes a number of different competitors to get an as broad perspective as possible.

3.1.4 Sketching

Sketching is a common first step of the ideating stage of a design process [20]. The goal here is to make many different sketches with just pen and paper, each sketch being a possible design solution solving the previously defined problem statement [20]. Sketching is a very cost efficient method for visualizing any kind of design, allowing designers to find design flaws early on as well as possibly contribute to generating new, innovative design solutions. In terms of GUIs, sketching is often done to generate multiple ideas for how the GUI can be designed.

3.1.5 Paper Prototyping

Paper prototyping is a quick, cheap way to test and evaluate the functionality of an application early in the design process [21]. The basic principle of paper prototyping is that you do not spend much time developing the paper prototypes. When evaluating a paper prototype, the user will perform their actions by interacting with the paper prototype interface. A facilitator will then respond appropriately to the users actions by adding overlays, switching to a new page etc. in order to simulate the experience of the real product. The focus of a paper prototype lies on the navigation and interactions rather than look-and-feel.

3.1.6 Wireframing

Wireframing is a way of visualizing a design without having the aesthetic aspect figured out [22]. A wireframe displays what elements will be located at different parts of a graphical user interface and how much space they will take. It also demonstrates how navigation and interactions will be done, as well as showing what functionalities will be available. The upside of wireframing is that it is easy to alter based on user feedback, hence it allows for user feedback early on in the design process before too much resources has been put into making a design. The absence of color and visual elements also makes sure that focus remains on layout and functionality, all of which are key to a good user experience. A wireframe can be thought of as a blueprint for a graphical user interface design. Wireframes can be either digital or done on paper.

3.1.7 Mockuping

Mockuping is when you make mockups of a design. A mockup is a visual representation of a design that looks similar to how a finished product would look [23]. That means that it has colors, actual icons, corporate logos and other important details that contributes to the overall user experience. A mockup does not have to be interactive, which means that it does not necessarily have actual functionality such as buttons triggering actions etc. There exist plenty of software for creating mockups that can be made interactive, allowing mockups to be turned into working digital prototypes very easily.

3.1.8 Cognitive Walkthrough

Cognitive walkthrough is a method for examining the usability of a product [24]. A cognitive walkthrough is task oriented; the user testing the product will be given a list of tasks to perform, with an observer present to observe the users action. The user is generally an expert user, meaning a user with previous experience from using that product or similar products . For each step of the process, a set of questions will be answered by the observer:

- Will the user try and achieve the right outcome?
- Will the user notice that the correct action is available to them?
- Will the user associate the correct action with the outcome they expect to achieve?
- If the correct action is performed; will the user see that progress is being made towards their intended outcome?

After a cognitive walkthrough (or several) has been performed, all of the observers notes are summarized. The feedback received is then used to improve the design in the relevant places. A cognitive walkthrough does not require a fully functional product; a prototype of any kind is good enough. This allows the method to be used early in the design process, allowing designers to catch issues while big changes are easy to apply.

3.1.9 Think-Aloud Protocol

The think-aloud protocol is another method for testing the usability of a product. Similar to a cognitive walkthrough, it is a task oriented method where the user is given a set of predetermined tasks to accomplish [25]. While performing these tasks, the user is asked to explain what they are doing, thinking and feeling. An observer will observe and take notes, as well as making sure that the user remembers to explain what they are doing. The reason for conducting a think-aloud is to give the observer some insight in what the application makes the user think; for example what parts of an application is good, confusing etc. The users participating in a think-aloud session are generally beginner users, as it may give more valuable insights when a user is not familiar with the product beforehand.

3.1.10 Heuristic evaluation

A heuristic evaluation is a method for finding usability issues in a GUI [26]. The users participating in a heuristic evaluation are generally expert users. Heuristic evaluations are performed in two evaluation phases. In the first evaluation phase, the evaluators explore the system freely in order to find some concrete elements that they wish to evaluate. In the second evaluation phase, the identified elements are evaluated based on a few predetermined heuristics. While the heuristics are generally tailored for your specific product, a good guideline is to base them off the following 10 heuristics proposed by Jacob Nielsen [27]:

- System status is visible to the user
- The system follows real-world conventions
- Give the user control and freedom

- The system is consistent and follows platform conventions
- The system prevents errors
- Useful and necessary functions is visible or easily retrievable by the user
- The system is flexible and efficient to use
- The system uses as an aesthetic and minimalistic design
- The system helps users recognize, diagnose and recover from errors
- The system provides documentation where needed

While a heuristic evaluation is performed, a facilitator records eventual problems and flaws. After the evaluation is completed, a list of problems are summarized. These will then be used as a basis for design improvements.

3.2 Wireframing Tools

This section lists some useful tools that were considered for this project. The tools are used for making digital wireframes and mockups of a design.

Figma is a GUI design application specialized at creating wireframes and mockups with little effort [28]. It allows for elements to be interactable and to trigger actions by navigating them to a new screen visualizing the updated state of a system. Another handy feature is that it allows for storing libraries of components that can serve as a design guideline. It comes as both a desktop application for Windows and Mac, and a web application available on all modern operating systems.

Adobe XD is a design tool for creating wireframes and mockups [29]. Designs created in Adobe XD can be made into interactive click-through prototypes. Adobe XD comes as a desktop application available for Windows and Mac.

3.3 Software Development Methods

This section describes some popular methods that were considered for the software development stages of the project. The methods listed include both software development processes and software design techniques.

3.3.1 Agile Software Development

Agile software development is a term referring to a set of frameworks and methods used for software development [30]. They are all based on the following 12 principles derived from the Manifesto For Agile Software Development [31]:

- The highest priority is satisfying the customer with early and continuous delivery of valuable software
- Welcome changing requirements, even late in the process
- Deliver working software frequently
- Business people and developers must work together
- Build projects around motivated individuals
- Face-to-face communication is the best way to convey information

- The primary measure of progress is working software
- Sustainable development is promoted
- Good design and technical excellence is given continuous attention
- Simplicity is essential
- Teams are self-organizing
- The team reflect on their effectiveness regularly

Some advantages of agile software development include flexibility in regard to changes as well as early and frequent delivery of working software.

3.3.2 Scrum

Scrum is a lightweight agile software development method that like all agile methods uses an iterative and incremental process [32]. The basic principle is that the entire process is divided into a number of sprints [33]. While the length of a sprint can vary, all sprints must be the same length. The length of a sprint is usually between 1-4 weeks. Each sprint goes through four stages: sprint planning, daily scrum, sprint review and sprint retrospective.

Sprint Planning

The sprint planning stage is the first stage of a sprint and is where the planning of the upcoming sprint is done [34]. Here, the team discuss and decide what items from the product backlog can completed in the next sprint. The result of a sprint planning is a sprint backlog, which includes the work needed to complete the items from the product backlog. Finally, a sprint goal is set which is the goal that is to be achieved through the implementation of the items from the product backlog.

Daily Scrum

The daily Scrum is a short meeting of maximum five minutes where each member of the software development team reflects about their previous days work [35]. Each member also plans what they should finish during the current day in regards to reaching the sprint goal. Any potential hinders for doing some work should also be brought up and dealt with.

Sprint Review

At the end of each sprint there is usually a sprint review [36]. At the sprint review, the completed and non-completed work is reviewed. The completed work is also demonstrated to the stakeholders if possible.

Sprint Retrospective

The sprint retrospective is what happens between the sprint review of one sprint, and the sprint planning of another [37]. Here the team discuss what went well in the sprint, what could be improved for the next sprint and how to achieve better results for the next sprint.

3.3.3 Feature-Driven Development

Feature-Driven Development is another agile software development method that also uses an iterative and incremental work process. It's a feature-focused method where the goal is to provide a feature every 2-10 day [38].

The first stage is where you *develop an overall model*. Here, the project scope and context is defined. The second stage is *building a list of features* using the overall model from the previous stage. Any features that can't be completed within two weeks should be broken down into smaller features. The third stage is the *plan-by-feature-stage* where the complexity for the different features is established and ownership of each feature is assigned to members of the development team. Using the complexity of each feature, the order of development is decided. The fourth stage is the *design-by-feature-stage*. Here, it is decided which feature will be designed and built. Feature priorities will also be defined here. The fifth and final stage is the *build-by-feature-stage* where the feature is actually implemented. The part of the GUI supporting the feature is also built here. Once the feature has been implemented, it is tested before being approved and pushed to the main build [38].

3.3.4 Kanban

Kanban is an agile software development method that revolves around three basic principles: visualizing your work, limiting the amount of work in progress and enhancing flow [39]. The visualization of work is done with a Kanban board. Kanban boards vary between the contexts they are used, but the general purpose is to have different categories such as planned items, items in progress, developed items and so on. The different items, or features, are then moved from one category to the next as they are done with that stage.

Limiting the amount of work in progress is done by setting a point limit of work that is allowed to be in progress at the same time. Each task is then assigned a certain amount of points based on the approximated workload it brings. The total amount of points from tasks in progress can never exceed the set point limit. These limits may need to be fine-tuned throughout the process. Enhancing flow is done by moving the highest item from the list to the in-development stage after another item is finished. While Kanban is an incremental method, it is not iterative like other agile methods are. It doesn't use any time-boxes or sprints either; instead, Kanban focuses on delivering new features and software as they are completed.

3.3.5 Modular Programming

Modular programming is a code design technique used in software development. The main principle behind this technique is to separate functionality of an application into different, independent modules [40]. This allows for individual modules to be swapped out without affecting the rest of the software. This decreases the coupling of the application, which promotes software maintainability, reusability and extensibility.

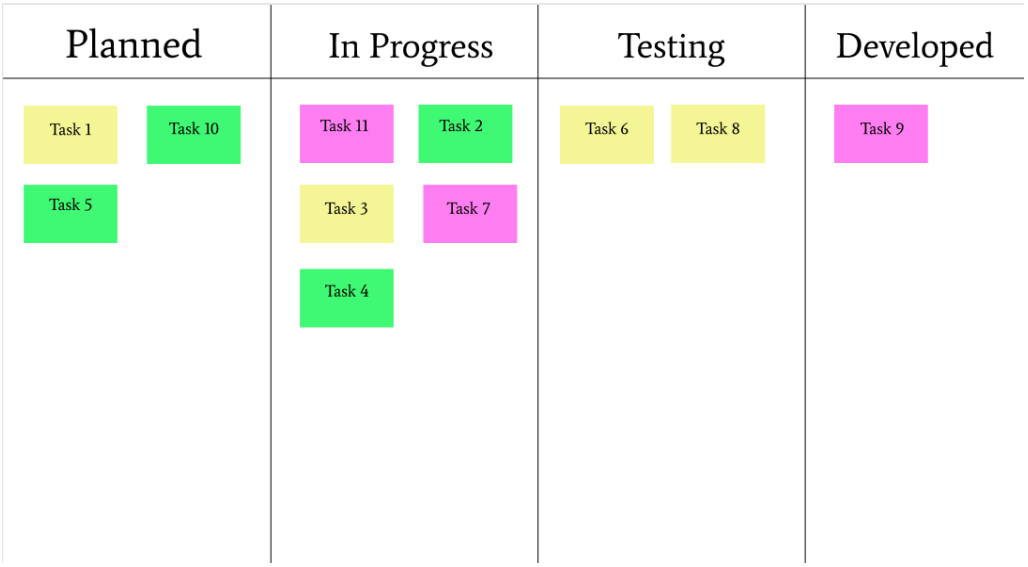

Figure 3.2: An example Kanban board

3.3.6 Interface-Based Programming

Interface-based programming is an architectural pattern used to promote modular programming in object-oriented programming languages where a native module system does not exist [41]. The main principle behind this pattern is to create an interface for each module. The interface functions as guidelines for what functionality must exist in a module. Extending an application by implementing the necessary functions specified by an interface ensures that an application will not break.

3.3.7 Module Pattern

The module pattern is a structural and creational pattern used to achieve modular programming in programming languages that doesn't support it by default [42]. The implementation of this pattern varies with the programming language; in Python it is built in and each .py file is already a module, in C++ it is achieved with having each module as a separate header file, and in Java it can be done using Singleton and static methods.

3.4 Software Development Tools

This section describes the Dynamo SDK that the tool will be built around as well as some useful tools for software development. Examples of tools include graphical user interface frameworks, programming languages and 3D rendering libraries.

3.4.1 DynamoSDK

Dynamo SDK is a software development kit that specializes in generating real-time animations with soft-bodies. It provides functionality for creating cloth patches, adding different objects, handling collisions, transforming objects, sewing patches

together and serializing a scene. The underlying soft-body animations are computed using the Vivace solver [8]. The SDK merely loads and handles the animations loaded in the scene; it is thus not in charge of any rendering. The SDK is written for C++.

3.4.2 Dear IMGUI

Dear IMGUI is an open source GUI library for C++ [43]. It is specifically designed for allowing programmers to create user interfaces for content-creating tools as well as visualization- and debug tools. According to their official site, it is particularly suited for game engines, real-time 3D applications and full screen applications amongst others. While the official version only supports C++, many unofficial bindings to various programming languages exist such as Python, Java and PHP.

3.4.3 Qt

Qt is another open source software used to create GUIs and cross-platform applications [44]. Qt is written in C++ but offers official bindings to Python. Multiple unofficial bindings exist for different languages such as PHP and Java. Aside from allowing the creation of GUIs, it also offers support for database access, thread management and network support.

3.4.4 OpenGL

OpenGL is an open source, language-independent, cross-platform API for 2d- and 3d rendering of vector graphics [45]. The API allows interaction with the computer's graphics processing unit (GPU) in order to perform hardware-accelerated rendering. It is commonly used in all sorts of applications involving computer graphics, such as computer games and 3d-modelling software. In order to incorporate OpenGL in an application, a context creating tool as well as an OpenGL loader library is required.

3.4.5 OpenGL Context Creating Tools

Creating a context for OpenGL is a complicated process that varies between operating systems. As a result of this, automatic context creating has become a common feature in various development libraries [45].

GLFW is a lightweight library that allows creating top-level windows (e.g. no GUI) with OpenGL contexts. It also provides means to register user input from hardware such as mouse, keyboard and joysticks [46].

SDL is another library that can be used to provide contexts for OpenGL, Direct3D and Vulkan. In addition to providing contexts, it further facilitates the managing of threads, shared object loading and networking [47].

3.4.6 OpenGL Loading Library

After creating a context for OpenGL, one must load all the OpenGL functions to initialize OpenGL [48]. This includes both core functions and possible extended functions. While it is possible to do this manually, it is a complicated process that requires using operating system dependent functions. Hence, it is recommended to use an OpenGL loader library to perform this step automatically.

gl3W is an OpenGL loader library which loads pointer to the core OpenGL functions at runtime, allowing the functionality of the OpenGL API to be accessed by an application [48]. gl3W only support the latest OpenGL (which is 4.6 by the time of writing) and requires Python to be generated.

glad is another OpenGL loader that loads OpenGL functions [48]. When generating glad through their website, there are a few options including OpenGL version and specified extensions which allows a user to tailor it to their needs.

GLEW is an OpenGL loader that loads all OpenGL core functions and all extension functions [48].

3.4.7 The OpenGL Utility Library

The OpenGL Utility Library, or GLU, is a library that offers more high-level drawing functions compared to the more primitive ones supported by native OpenGL. Example functionality includes rendering geometry spheres and cylinders, as well as functionality to transform between space- and window-coordinate systems. The library has not been updated since 1998, and it contains functions relying on deprecated code. However, it is still distributed alongside the base OpenGL package.

3.4.8 Open Asset Import Library

Open asset import library, or Assimp, is an open-source library for importing objects in various common 3D-file formats into an application [49]. Recent versions also include a feature for exporting 3D files. Additionally, it offers some tools for mesh post processing. Assimp is particularly suited for use in game engines and real-time rendering applications of any kind. Assimp officially provides an API for C and C++, but bindings exist for multiple languages including Java, Python and C#.

3.4.9 Magnum Engine

Magnum engine is a lightweight C++ graphics library that utilizes OpenGL [50]. It provides an abstraction of the OpenGL API, built-in shaders and a number of available plugins including one for Assimp.

3.4.10 Serialization Library

Cereal is a serialization library written in C++, supporting serialization of various formats such as JSON and XML. It further supports serialization of binary data in addition to human-readable data. The library is header-only, and is thus easy to add to a project since no other dependencies are required.

JSON for Modern C++ is a library for C++ that allows conversion of data to and from JSON. What is prominent for this JSON library is the intuitive syntax, ease of integration in an application as well as memory efficiency.

3.4.11 Transformation Gizmo Library

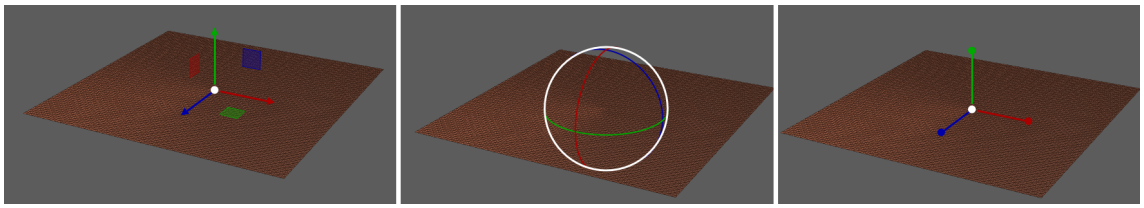

Figure 3.3: An example of how the three types of transformation gizmos can look. From left to right: translation gizmo, rotation gizmo and scaling gizmo.

A transformation gizmo is a widget that allows the transformation of objects through mouse interaction. Commonly deployed in 3D modeling software and game engine editors, the gizmo usually appears at the center of a selected object. There are three kinds of transformation gizmos: translation gizmos, rotation gizmos and scaling gizmos. The following are some popular external libraries for incorporating these gizmos in an application:

Tiny Gizmo is a lightweight library for including transformation gizmos in an application. The library is written in C++ and requires no other dependencies.

ImGuizmo is another lightweight library for including transformation gizmos in an application. Written in C++, the library is built on top of the Dear ImGUI library and thus requires an application to use that library in order to function.

Im3D is also a lightweight library for including transformation gizmos. It further provides functionality for rendering primitives as well as supporting VR applications. It requires no other dependencies in order to work.

3.4.12 Additional Libraries

The VCG Library is a C++ library for manipulating meshes. The most important functionality offered by this library for this project is the ability to easily construct rays and perform ray-triangle intersection tests in order to build a ray-casting func-

tion.

4

Planning

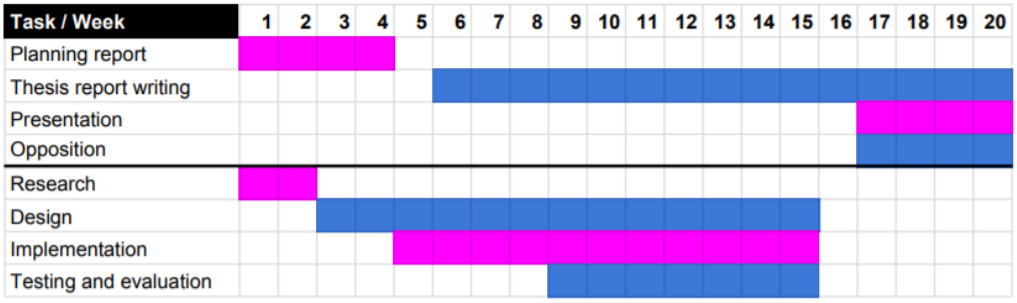

Figure 4.1: The time plan for this project

4.1 Process

The project will be developed using an agile process. Thus, each of the following steps will be executed several times in an iterative fashion.

4.1.1 Research phase

The first step will be the research phase. There are several things that will have to be looked into in this step: The first step will be researching the domain and context of this thesis, including researching related work and literature. The second step will be studying the DynamoSDK, which involves exploring the different functionality offered by the SDK, and how it works. The next step will consist of researching different open-source GUI frameworks and deciding upon a programming language. Both of these things will be done in parallel, as both of them relies on each other; a good GUI framework can motivate a certain programming language, and a good programming language can motivate a certain GUI framework. Finally, the back-end OpenGL will be researched. This includes not only exploring the API itself, but also exploring the different necessary extensions such as an OpenGL loader and an OpenGL context creator which will be used to include the OpenGL functions in the application.

4.1.2 Design phase

The second step will be the design phase. Here, the design is created from the user needs as well as the functionality offered by the DynamoSDK and other software used. The design phases will include the last four stages of design thinking, namely defining, ideation, prototyping and testing. The empathy stage is outside the scope of this project; necessary data about the domain and user needs is provided by the company.

During the ideation stage, a number of different initial GUI designs will be developed using sketching. These will then be compared and possibly merged into a final sketch. A competitor analysis of competing software listed under "Related Works" will be conducted as well to aid this phase. During the prototyping stage, a prototype will be made based on the current design. The kind of prototype will depend on the stage of the process; early it will be a low-fidelity paper prototypes and wireframes. In the middle it will be medium-fidelity wireframes and mockups. Late in the process it will be high fidelity mockups and working software prototypes. During the testing phase, the currently existing prototype will be tested and evaluated. For this purpose, the methods cognitive walkthrough and think-aloud protocol will be used. The testers will range from professional animation designers to hobby users. Feedback received from the testing stage will be used to improve the current design, before starting back from the defining stage.

4.1.3 Implementation phase

The third step will be the implementation phase. This is where the tool is actually implemented. The implementation will be done incrementally on a by-feature-basis, using a combination of the methods Scrum and Feature Driven Development custom tailored for a development team consisting of a single person. The code architecture will be based on modular programming. The interface-based programming pattern and module pattern will be used to achieve modularity and thus also maintainability, reusability and extensibility of the code.

5
Execution and Process

This chapter presents how the execution of this project was conducted. Each of the following section describes one of the stages of the project namely the research phase, the design phase and the software development phase.

5.1 Research Phase

This section will explain how the research phase of the project was conducted. The research phase is split into research of existing 3D modeling software, relevant literature and software development tools.

5.1.1 Researching 3D modeling software

The first step of this project was to research currently existing 3D modeling tools. The tools explored were Autodesk Maya, CLO3D, Houdini, 3DS MAX and the DeformPlugin. At this stage, no rigorous analysis of the tools were performed. Instead, this initial exploration of the tools focused on their general appearance. The tools chosen were based on their reputation within the industry.

5.1.2 Researching relevant literature

This research refers to any literature used during this project. This includes literature about the areas of computer graphics, 3D modeling, interaction design and software development.

5.1.3 Researching software development tools

The first step when researching software development tools was getting familiar with the companies software development kit; the Dynamo SDK. This process consisted of reading documentation for the SDK, checking out various company-made demos that they provided and trying it out myself. This provided valuable insights such as key features, capabilities, limitations as well as how it could be used to create animations. It basically served as the basis for the vision for the entire project. Deform Dynamics further provided a list of what they considered the most typical use scenarios.

Next, different software development tools, such as programming language and external dependencies, were researched. External dependencies researched included

GUI frameworks and libraries for importing objects.

As for programming languages, the two main ones that was considered was C++ and Python. Advantages of C++ was the fact that it is much faster than Python, as it is a compiled language. However, C++ being a compiled language also means that every change in he code would result in the code to be recompiled. Python, being an interpreted language, can completely avoid the compilation time. Python is also a simpler language, allowing faster writing of code [51]. Most of the other software development tools that were researched were in a similar situation with official support for C++ while Python support varied between official, unofficial, experimental and so on. However, the Dynamo SDK was only native supported by C++; in order for it to work with Python, bindings would have to be made. In the end, C++ was chosen as the programming language. This was due to it being decided that faster writing of code and no compilation time provided by Python would not outweigh the official support of researched frameworks, and more importantly the increased speed and performance provided by C++. Since the 3D animations provided by the SDK, which are to be built by the application, are based around being fast and efficient, performance while running the program was determined to be the most important thing in the eyes of the potential users.

The two main GUI frameworks researched were DearIMGUI and Qt. The main difference between the two was the fact that DearIMGUI is more lightweight and easy to manage, while Qt offers more functionality. In terms of compatibility, both libraries were supported by C++ officially, with unofficial bindings allowing both of them to work with Python. Since DearIMGUI seemed to provide all the functionality expected to be in the application, it was thus chosen for its lightweight characteristics.

The OpenGL context creating tool selected was GLFW. This was due to the fact that it is lightweight while still providing everything needed.

As for an OpenGL loading library, gl3W was selected for the application. This was simply because it only loads core functions, thus being faster than the alternatives. As no extension functions were needed, it was deemed the superior choice.

When deciding upon a library for importing objects into the application, the open asset import library, or Assimp, was the only option considered. This was mainly because of the sheer amount of recommendations when researching options, as well as it existed many tutorials for getting started with it in OpenGL.

5.2 Design Phase

When all the decisions had been taken, the next phase was to start working on the design. A number of different sketches representing initial GUI designs were developed. The focus on these sketches were element placements and the navigational structure of the GUI. Features included in the sketches were based on the most typ-

ical use scenarios as well as what functionality was provided by the Dynamo SDK. These sketches were then compared, reworked and combined several times until a single sketch consisting of the best features of each independent sketch existed. This initial sketching was done before any competitor analysis was conducted, in order to minimize their effect on the design. The sketch was then evaluated through a cognitive walkthrough in order to catch some early design flaws and other potential issues. The findings of the cognitive walkthrough were used to improve the design and fix any discovered issues.

Next, a rigorous competitor analysis was performed on the existing 3D modeling tools mentioned in previous sections. This was focused on analysing specific parts of their GUIs such as how various configurable parameters were presented, how the menus were designed and how various GUI elements were placed on the screen. This gave insights about what different tools and panels are commonly used and how they are presented to the user. These findings were used in order to enhance the current design of the 3D modeling tool developed in this project.

The improved design was made into a paper prototype. This paper prototype was used to evaluate the design with two actual, beginner-level users using the think-aloud protocol. Actual beginner-level users refer to persons within the target group that have little to no previous experience working with similar software. The tasks given to the users were tasks such as adding an object to the scene, performing transformations, importing objects, running the simulation and saving the project. Some interested things were noticed from these tests, for example that the different users sometimes had two completely different approaches to the same task. The way transformations were performed could also vary for the same person; a user could attempt to translate an object by dragging it with the mouse, while the same user could later attempt to rotate the same object by altering the rotation-values through the GUI. Furthermore, the lack of explanatory labels on certain GUI elements was noticed to be confusing to beginners.

The feedback received from the think-aloud sessions were used to make new sketches of an improved design. After comparing and combining the new sketches, a final sketch was made which was then made into a new paper prototype. This paper prototype was used to evaluate the design once again. It was done similarly to the first tests; think-aloud protocol with two different users. These users were however intermediate-level users this time, meaning they had some experience working with 3D modeling software. The list of tasks was the same as the one used in the previous usability test in order to get feedback on the same things. During this second iteration of user testing, the tasks were completed much faster with the test users being less confused in general which indicated a good improvement from the last design. Once again, this feedback resulted in new improvements to the current design.

The newly improved design was then turned into a digital wire-frame using the web-based tool Figma. Figma was chosen for this purpose for multiple reasons: first of all, it is web-based. This means that no software is required, and the wire-frame

will be stored online. Storing it online means that it is protected against losing the computer, and that it can be worked on from anywhere. The second reason was that Figma is a tool that I have been using previously, meaning there would be no need to learn a new tool just for this purpose. The resulting wire-frame was then demonstrated at the Deform Dynamics company office to the CEO and one of the other associates there. It was then discussed and evaluated, with rich resulting feedback in terms of current issues, possible fixes and things to add. This feedback included things such as having keyboard shortcuts and visual hints explaining these, increased modal feedback when performing various actions, camera behavior, the choice of words for different GUI elements and the size of different elements such as buttons and icons.

The last planned stage of the design phase, which was evaluating the implemented prototype with real users, was supposed to be done after the software development phase at the end of the project. However, because of the COVID-19 pandemic, this was scrapped in favor of evaluation by the developer and the company supervisor.

5.3 Software Development

With the design being pretty far gone, it was decided that it was time to setup the initial project for the application. This involved setting up a project, downloading all dependencies that had been chosen and setting them up to work with the application. In order to get the GUI provided by the DearIMGUI library to render, a GUI module was developed with the help of their provided example code. After successfully making it start up a window with OpenGL and Dear IMGUI, it was time for the actual software development to start. Focus was on functionality as it would not be affected by any potential changes to the GUI.

Before the actual coding could begin, a list of tasks required to complete the application was constructed, i.e. a product backlog. These tasks were based around features that had been decided to be part of the finished tool, such as adding objects to the scene and actually rendering existing objects. Whether some tasks depended on other tasks was also determined. At the start of each week, a task from the product backlog would be chosen. The task chosen would be based on necessity (for example rendering objects is vital for most other features, hence should be done as soon as possible), complexity (larger tasks are best to do early in a sprint, smaller task later in the sprint) and personal preference. The necessity of an item would also often be affected of the wishes of the company supervisor, and would thus be discussed on the weekly meetings.

5.3.1 Sprint 1 - Implement the Rendering

The goal of the first sprint was to implement rendering of soft-bodies loaded to the scene by the Dynamo SDK. It was decided that it would be made from scratch rather than using a rendering library such as the Magnum Engine. The main motivation behind this choice was the fact that the company wanted to minimize external

dependencies. Another reason was that the process of setting up the Magnum Engine to work with the application did not appear to be trivial. A third reason was that reusing shaders and other rendering-related code that the company had been developing for their demos would be easier without it. To sum it up, it was not considered a big enough time-saver for the project that the downsides of using an external dependency was outweighed. With the modular programming approach in mind, the rendering was implemented as a render module.

Since the SDK provided all the vertex data for the soft-bodies in the loaded scene, this initial work on the render module mainly consisted of generating OpenGL buffers and passing them pointers to the vertex data. Fragment- and vertex-shaders providing a basic lightning model and uniform object colors were also developed in order to be able to render the scene. Furthermore, a shader module was developed for efficient loading of shader code into the application. Using this module, loading a shader from a file would be as simple as passing along the filepath when constructing a new shader object. Doing this further makes it easy to use many different shaders in the same application.

As the rendering of soft-body objects was working before the sprint was over, the functionality to render different collider objects were worked on as well. Since colliders stored by the Dynamo SDK were not associated with any vertex- or index-data, these had to be manually rendered. To facilitate this part of the renderer, the GLU library was added. This library provided a function for rendering a sphere at a given position, gluSphere, which was used for the sphere colliders and also both ends of the capsule colliders. The reason that the GLU library was included despite using deprecated functions was mainly because it was included in the OpenGL package, hence the functions could be used by simply including the GLU.h header file. While it was initially intended to be a temporary solution, it was never a high priority to replace it with a custom function for rendering spheres. In comparison to the sphere colliders, the plane- and box-colliders were rendered by manually constructing vertex data from their positions which were provided by the SDK. By the end of the first sprint, sphere- and box colliders rendered successfully.

As the last task of the sprint, a simple GUI pane with buttons allowing patches and colliders to be created with a single click was also added. The main reason for adding the pane was to test the interplay between Dear IMGUI and the Dynamo SDK. In this version, all objects created with the buttons used predetermined values with no possibility of modifying them from a users perspective.

5.3.2 Sprint 2 - Importing Objects

The goal of the second sprint was to finish the rendering of he rest of the colliders and manage to import objects from .obj files. Rendering plane colliders was done quickly, but rendering capsules experienced some issues. The capsule colliders were rendered, but their orientation and position were not matching their actual orientation and position for reasons unknown at the time. After some bug-searching that

did not result in any solution, the functionality was postponed. Instead, work on the importer module for importing soft-bodies to the scene was prioritized instead.

The importing feature was divided into two tasks: fetching the vertex data from the file containing the object with Assimp for manual rendering, and loading the vertex data to the scene using the Dynamo SDK. The first task, while a rather complicated task, went rather smooth thanks to the various guides and documentation existing for working with Assimp. The second task, while initially looking like an easy one, proved to be harder than expected as the results would be poorly rendered objects with some files causing crash issues. This led to a heavy debugging session.

Figuring out what was causing the bugs was quite the challenge. The vertex data provided by the developed importer seemed correct as the objects would render manually, and loading them into the scene was done following documentation for the SDK. Thus, everything appeared to be handled properly. Luckily, Deform Dynamics provided a demo where they imported a .obj file, which included the source-code. Here it was discovered that they were using another external library called tinyobjloader instead of Assimp. After researching and testing how the two libraries differed, a few interesting points were discovered. The most important discovery was that when using the tinyobjloader library, the vertex count for an imported object would be much smaller than the vertex count for the same object loaded by Assimp. Assuming the different handling of vertex data was the cause, another importer module using the tinyobjloader library was developed. When switching the old importer module for the new one, loading imported objected to the scene successfully worked. Switching between two different importers further displayed the modularity of the source code, and how well it worked in practice thus far.

The last task for this sprint was to add a working file-dialog to pop-up when clicking import on the GUI. As this was not something that was natively supported by the DearIMGUI library, alternative ways of implementation had to be researched. The first attempt was to use an unofficial open-source extension. That solution did not provide satisfying results, however, as the resulting file dialog would look poor from an aesthetic point of view. Eventually a proper file-dialog was made using the Windows 10 SDK, providing the default Windows file dialog.

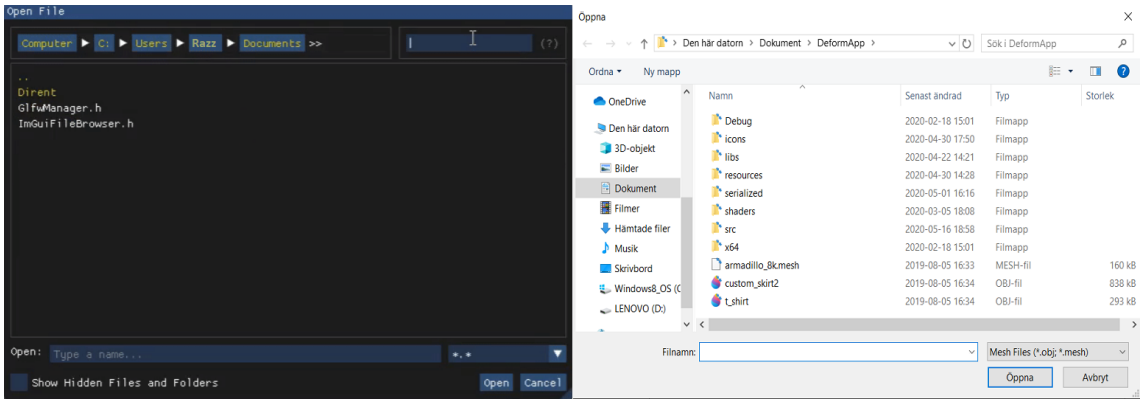

Figure 5.1: This figures compares the two versions of the file dialog. The left side of the picture contains the old one, and the right side contains the new one.

5.3.3 Sprint 3 - The Picking Feature

The main goal of the third sprint was to implement the picking feature. This feature is what would allow a user to drag or stretch a piece of cloth using the mouse cursor. As the SDK provided functions for modifying the mass of a specified particle, the main functionality to be implemented in order for picking the work was a method for finding a specific particle using the mouse cursor. This was done using a classic ray-casting approach, where several ray-triangle intersection tests were performed in order to determine if a particle had been hit. These tests were done between a ray projected from the mouse cursor into the scene, and all triangles in the scene. To facilitate this process, the VCG Library was added to the project. This library allowed for easy construction of rays as well as a function for efficient ray-triangle intersection testing.

Another feature implemented during this sprint was the ability to drag objects with the mouse. In the current state, it was limited to the x- and y-axes of the object. In order to drag an object, it first had to be selected from the hierarchy view of the GUI as selecting objects in the scene through ray-casting had not yet been implemented. This feature also uncovered a bug with the renderer when translating colliders, which was fixed immediately.

In this sprint, the logical model for the objects handled by the application was constructed. This was done by defining the class SceneObjects, with sub-classes representing different types of objects in the scene such as colliders and deformable objects. These were then assigned different attributes such as id, name, position, scale and type. The main reason for creating this model was in order to present objects in the scene to the user in the hierarchy panel. Making the hierarchy panel was also done here. An "Object Details"-view was also added for testing purposes, allowing modifying of basic parameters such as position and scale for a selected object through the GUI.

The final work of this sprint was to implement basic camera movement. The move-

ments implemented were simple x- and z-axis translations as well as the ability to rotate the camera around its own position with the mouse. In order to get this piece of code loosely coupled from the rest of the application, a camera module was developed putting the entire functionality of the camera into a single header file. The camera was implemented at this stage largely for debugging reasons, and thus the way the movement was performed was based on how it would facilitate debugging rather than how it would be preferred for the final application by actual users. The implementation of the mouse-translation further gave rise to a design question regarding how to handle different mouse-based features. "Should the camera be moved only when a camera tool has been selected?", "should the camera only move when an object is not selected?" and "should clicking on an empty part of the screen deselect an object and automatically activate the camera tool?" were some of the questions resulting from this work. Answering these question was not done immediately; instead they were put on the list of tasks to do. The ability to modify the camera values through the GUI was also added at the end of the sprint.

5.3.4 Sprint 4 - The Painting Feature

The main feature of the fourth sprint was to implement the painting feature. This feature would allow a user to paint particles with the cursor using different tools. The different things that was intended to be painted by a user was friction, collision masks and fixed vertices. This feature was divided into the following sub-tasks: alter the value of the particle hit by a ray-cast, rendering "paint" on the affected particles in order to give the user visual feedback, and allowing the brush-size to be modified to hit a larger surface with each click. Since the ray-casting function already existed since the implementation of the picking feature, the first sub-task was trivial to implement.

The first attempt at rendering painted particles was to first store their index in a list, and then use functions provided by the Dynamo SDK to get their render position. Then, a sphere would be rendered at that location using the gluSphere-function provided by the GLUT library. While this technically achieved a decent outcome, there were some concerns. One concern was that the visuals were rather ugly. Another concern, which was the most important one, was that a single object with all particles painted would cause some severe performance issues. This resulted in other approaches to be developed. The next, and final, attempt used a similar albeit a much more effective solution. In this version, selected particle indices were stored in a list like the first version. This time, however, vertex data pointers were manually constructed from the positions of the particles from the list. The pointers were then sent to the renderer which drew them as GL POINTS. With this version, there were no significant performance issues.

The final sub-task was to alter the brush size. This was done using a function provided by the SDK to find particles within a certain radius from a given particle.

Options to switch between the various painting tools, and to modify different param-

eters for each tool, were also added to the GUI. These options included setting the friction amount, the active collision mask and brush radius. Furthermore, buttons for applying a selected paint to all particles and clearing all particles from selected objects were added to the GUI. The different painting tools are selected by clicking different icon buttons from the toolbar. In order to have images for the icon buttons, the ability to import regular image files and bind it to a texture was added to the importer.

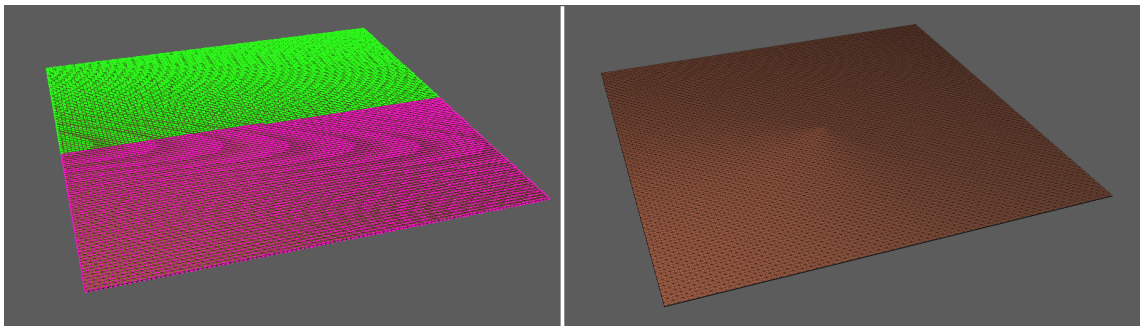

Figure 5.2: The left of this figure shows a deformable patch where one half has been painted with one collision mask, and the other half with a second collision mask. The right side of the figure shows the same patch without any paint applied.

5.3.5 Sprint 5 - Implementing Camera Behavior

The first task to be performed for the fifth sprint was the ability to select objects in the scene through ray-casting. This was also done using the SDK-provided function to find nearby particles, as one of the output parameters was a pointer containing all object IDs. Thus, using a small radius would guarantee that the output object ID would belong to the selected object. This feature seemed to work correctly at first, but it was later discovered that it would not return IDs of collider objects. Hence, it only allowed selecting soft-body objects in the scene.

The second, bigger feature for this sprint was to implement the camera in a way so that it behaves as a user would prefer, instead of in a way so that a developer can perform fast debugging. In order to get an idea of how the end result should be, a competitor analysis was performed. Houdini, Maya3D and 3DS MAX all had their camera behavior examined thoroughly. As it turned out, all of them had the same type of camera movements, albeit with slightly different ways of accessing them. The decision thus became to implement the same camera movement in the application. The desired translation of the camera position was similar to how it was already implemented; the main difference was the controls for performing the translation. The controls were thus updated to be aligned with them, as there was nothing suggesting a deviation from an industry standard was motivated. The biggest change done to the camera behavior was the camera rotation. The initial behavior, which was the camera rotating around its own position, was replaced with camera behavior where the camera rotates around the center-point of its field of view.

The final task of the fifth sprint was to start working on incorporating mesh colliders into the application. This was split into two sub-tasks: adding mesh colliders to the scene and rendering loaded mesh colliders. Adding mesh colliders was trivial after storing vertex data for imported objects. Rendering was implemented, albeit with a huge bug; no positional data affected the rendering, and thus rendered mesh colliders were static. After some discussion with the company it was determined that this issue was a part of the version of the Dynamo SDK that was being used. A fix on their end was thus on the way.

5.3.6 Sprint 6 - Adding Convenience Features

The focus during the sixth sprint was to design and implement some important user actions, such as undo/redo features, copy/paste features and transforming a selection of several objects at once. The design of these features refer both to how these actions should behave, as well as how they should be made accessible to the users.

The first task thus was to design the behavior. In order to make them behave as a user would expect, a competitor analysis on how the features were implemented in existing 3D modeling software was performed. Once again, the analysis showed that there seems to be an indubitable standard for how these features behave in such tools, resulting in that same behaviour being mimicked in the application.

The second task was to actually implement the copy/paste function. Since all objects in the scene was stored in a list, with each object containing all information required to recreate it, copying an object was as simple as creating a new object with same type and parameters and placing it in a list representing a typical clipboard. The paste function worked by adding the object from the clipboard to the list of objects, as well as adding an object of the appropriate type to the dynamo scene.

The final task was to implement the undo/redo feature. This was split into two sub tasks: deciding how it should be implemented and actually implementing it. When deciding how to implement it, there were two options that were considered. The first one, which was the initial thought, was to store each action performed in a stack. The reasoning behind a stack was that they behave on a first-in-last-out basis, which fits the typical behavior of undo/redo features in software. Since actions within the applications are often tied to one or more objects, a list of affected objects would have to be stored with the action. If an action would be adding object x, then if the undo action was performed this would simply be delete object x. The deletion of object x would then be put in a separate stack which the redo feature would fetch from. This approach seemed perfect at first, but came with one big flaw: the reverse action of adding an object, which is deleting an object, is a function missing from the Dynamo SDK.

The alternative solution, which was inspired by an implementation of the undo/redo function in Adobe Photoshop, was to store the state of the entire application with each action performed [52]. This would allow for undoing irreversible actions, but

it would be much more computationally expensive. It would also make the scene having to initialize every time the user performs an undo/redo, which could lead to disrupting user flow. As a meeting with the company supervisor was coming up, it was decided to discuss the matter with him before moving forward with this feature.

During the meeting, a huge turning point appeared. After having investigated an issue the application had been having with the Dynamo SDK, the company had found the source of the problem. As it turns out, the issue was the fact that the SDK was not intended to perform actions such as adding objects, performing transformations on objects and so on, after a scene had already been initialized. However, in order to add and render objects, the scene had to be initialized. The solution to this problem was to redo a huge part of the app so that it did not add objects to the SDK and initialized the scene every time the user performed an action. Instead, it would have to be remade into a more typical 3D modeling tool with an editor containing static objects, and the simulation running after the press of a play/run button. This would allow the objects to be added to the scene handled by the SDK only when pressing the play button, thus every object can be added and transformed before initializing the scene. This also meant that the rendering of objects and most other user actions would have to be handled entirely without functions provided by the SDK up until that point.

While this meant that the work required to be done would increase a lot, it did solve the dilemma regarding the undo/redo feature that was mentioned earlier. Since adding an object no longer would use SDK-functions, the lack of a delete function was no longer an issue. Thus, the method of storing actions in two stacks (one for undo, one for redo) was now the optimal choice.

5.3.7 Sprint 7 - Refactoring the Application

With only a month remaining of the implementation, the remaining four sprints were discussed with the company supervisor. During this discussion, the goals for each of the remaining sprints were formally stated. The main task for the first of these four sprints was to change the application to manually render static placeholders for all objects while in the editor, and then switch to using the Dynamo SDK in order to simulate the animation after pressing play/run. Another part of the same task was refactoring of other functionality in the app to make it work with this new approach. As it was approximated that this would not take up the entire sprint, the rest of the week would be dedicated to begin working on serializing the data in order to save and open project files.

Everything regarding the collider objects was unaffected by the changes to the application, as they did not rely on SDK functions, and thus did not require any further work.

For most of the different types of deformable objects available, rendering static placeholders was nothing more than manually binding buffers with appropriate vertex-

and index-data, before performing a draw-call for each object in the list of scene objects. For many of these objects, the data was already available through the importer. However, this was not the case for deformable patches and volumetric meshes as the Dynamo SDK had previously computed the data for them. Fortunately, the company supervisor provided the source code used by the SDK to generate the data, resulting in the refactoring of the rendering of objects to be done efficiently.

The next task was to refactor other functionalities in the application. These functionalities were the painting particles feature and the ray-cast selection of objects. In order to get this to work, the approach was to recreate the same kind of behaviour that previously had been provided by the SDK. For the ray-casting, this meant looping through all deformable objects in the scene, and performing a ray-triangle-intersection test for each triangle of that object. This also meant that all deformable objects must store their vertices and indices, which was previously not the case. The new changes also meant that each object would have to store their own painted particles, so that they could be passed to the dynamo scene upon running the simulation. After that was added to the scene objects, the refactoring was deemed to be completed.

For the last day of the week, work on serialization of data began. This was done using an external JSON library called "JSON for modern C++". The main reason behind storing the project files as JSON was that the company had asked the serialized files to be human-readable, making JSON a perfect candidate. The JSON file format is also very common and easy to use, making it the obvious choice. Using the library, saving and opening project files was completely working by the end of the week. However, during the weekly meeting with the company supervisor, the choice of library to perform the serialization was not considered optimal. Instead, they encouraged the use of the Cereal library. The main motivation was that it was a library they already used, thus minimizing external dependencies used by their products combined.

5.3.8 Sprint 8 - Various Fixes

Given how the seventh sprint ended, the first task of the eight sprint was to refactor the serialization functions to use the Cereal library. While Cereal was found to be less intuitive than the "JSON for modern C++" library, the refactoring was done in one work day. With the new approach for serialization, each type of object had to implement a serialization function determining how the data of that type of object should be serialized.

The next task of the sprint was to implement the undo/redo feature. It was done as planned two weeks prior, which was to use an undo stack and a redo stack containing the performed actions along with a list of affected objects.

During the first half of the sprint, a memory leak causing the application to crash after being active for around ten minutes was discovered. After some investigation,

this issue was discovered to be caused by new buffers being generated during each frame, without ever being deallocated. This was simply fixed by making sure all buffers were only generated once, and then bound properly each frame.

The final task of the sprint was to allow importing animations from .fbx files. Since the tinyobjectloader library used for importing static deformable objects only supports the .obj file format, Assimp was once again put to use in order to import animations. In addition to importing the vertex- and index-data from a file, a function for computing vertex- and index-data at any given time had to be constructed. That also meant having to compute bone data for animated characters. Being a quite complex task, this was done with some help from the company supervisor. By the end of the week, this feature was fully implemented.

5.3.9 Sprint 9 - Adding Transformation Gizmos

The first task of the 9th sprint was to add transformation gizmos to objects in the scene. In order to do that, research on possible gizmo libraries had to be done. The three libraries considered were tiny gizmo, ImGuizmo and Im3D. After some testing, tiny gizmo was eliminated because it would require the way that objects are rendered in the application to be changed. While both the remaining two seemed simple enough to incorporate into the application, ImGuizmo seemed to be the most simple. Furthermore, the general downside of ImGuizmo seemed to be the fact that it requires the application to use the Dear ImGUI library for the graphical interface. This was of course not an issue as the application already did use that library for the graphical user interface. Im3D also had cheaper looking graphics, which together with the other reasons resulted in ImGuizmo being the chosen library.

Having chosen a library for transformation gizmos, adding the gizmos to selected objects was an easy task. However, thus far rotation of objects had not been a part of the application and was something that had to be added. This was done by adding a unit quaternion to every scene object that represented the orientation.

In order to account for rotations in the application, some functionalities such as ray-casting had to be updated to accommodate the new changes, which became the second task of the sprint. This task also included allowing the orientation of an object to be edited through the GUI. Since the representation of an objects orientation in the GUI was an euler-angle-representation rather than a quaternion representation, a conversion between the two had to be implemented. The fact that a single orientation can be represented as a number of different rotation sequences also made the function a little bit glitchy, albeit working.

The third task was to add functionality which allows a user to drag a timeline on an imported animation in order to show the frame at any given time. Having already implemented a function for retrieving vertices and indices for an animation at any given time, the technical implementation was a trivial task. However, as the feature was not part of the original concept for the application, it had to be decided where

to incorporate this feature in the GUI. In the end, it was placed as a slider under the "Object Details"-tab for the selected animation.

The final task consisted of adding tags to the hierarchy view in order to more easily separate different types of objects to each other. These tags were simple texts specifying the object type, surrounded by square brackets. For example, the tag for a collider object would be "[Collider]".

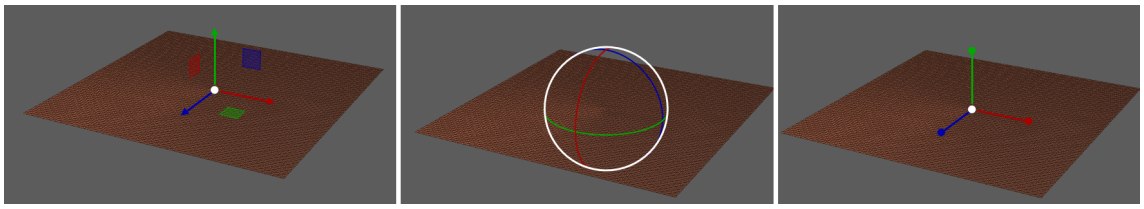

Figure 5.3: The left of this figure shows a deformable patch where one half has been painted with one collision mask, and the other half with a second collision mask. The right side of the figure shows the same patch without any paint applied.

5.3.10 Sprint 10 - The Final Sprint

The main focus for the final sprint of the project was to make sure that the application could be used to reproduce all of the demos that the company had previously developed. The second focus was to clean up to GUI and fix the aesthetics, such as adding proper icons for the tools on the tool bar.

In order to make sure all the demos could be reproduced, the most logical approach was considered to be simply trying to reproduce them. While doing so, some missing features such as anchoring a deformable object to a collider, were discovered. Another missing feature was loading .def files containing serialized data of clothes that are to be added to imported animated characters. Other missing features were mainly altering of different parameters, such as object stiffness and some size parameters for different objects.

Adding different parameters to be altered was a mechanical task, as this functionality was already designed for, but the other features required some design decisions to be made. In the end, the anchoring feature was implemented so that a user gets access to the anchoring tool if an appropriate collider object is selected. When using the anchoring tool, the user is prompted to select an object to anchor to it. If an object is anchored or not is further visualized through the GUI.

The function for loading .def files were provided by the company, which meant that the only thing that had to be implemented was the ability to browse the computer for a file to be added to an already imported animation. Deciding how a user should access it through the GUI also had to be decided. This was eventually decided to be done by adding a button to the object details panel for a selected animation if no file had already been loaded.

The final task of the week consisted of fixing the aesthetics of the GUI, for example aligning everything properly, as well as successfully reproducing and saving most of the demos previously developed by the company. With the demos being successfully reproduced and saved, the prototype for the 3d modeling tool was considered finished.

6
Results

This section will describe the results generated from this project. First, the developed 3D modeling tool will be presented in detail. Then, the answers to the research question of the thesis will be presented.

6.1 The 3D Modeling Tool

The tool built during this project is a hi-fidelity prototype. Most of the key features, such as adding and manipulating different objects, are fully implemented and working. The GUI is constructed in a way to visualize element placements and the resulting user workflow, however certain aesthetic aspects such as colors has not been part of the main focus.

The tool allows a user to create an animation utilizing the Vivace solver using a number of different tools. External objects and animations can be imported into the application from some common file formats. The ability to add patches of cloth and various geometry collider objects is also available to the user with one or two clicks respectively. Objects created this way are added to the scene using parameters, predetermined by the application, that can be changed at any time through the graphical user interface with direct manipulation. Objects can be transformed and configured in a large variety of different ways in order to accomplish a desired animation.

The following sections describe the implemented features of the prototype as well as how they are used by a typical user in order to create an animation.

6.1.1 Start View

The start view, as shown in figure 6.1, is how the application looks like when a user opens up the app. At the top of the application is a menu bar containing some common tabs. On the left side of the application, a toolbar is found. The toolbar offers the user a variety of different tools, such as altering the friction of an object on a particle-basis, fixing vertices i.e. making them ignore external forces and transforming objects. In addition to the tools, two buttons are located at the top of the toolbar: one for adding a deformable patch to the scene, the other to add a collider object to the scene.

On the upper right side of the application, there is a panel with a "Simulation Settings"-tab which is selected by default. The panel contains a "Play Simulation"-button for running a finished simulation, as well as different global parameters that can be configured by the user. These settings include enabling wind and gravity with different values on a per-axis-basis, manipulating different camera parameters and enabling or disabling the rendering of object wireframes. The configuring of parameters is done with direct manipulation.

Below the "Simulation Settings"-panel, there is a "Hierarchy"-panel. This panel shows a hierarchy overview of all objects in the scene, which is naturally empty when starting up the application. The center of the screen shows a rendered, rectangular plane. This plane is used to provide the user with a visual reference of the current camera position and orientation.

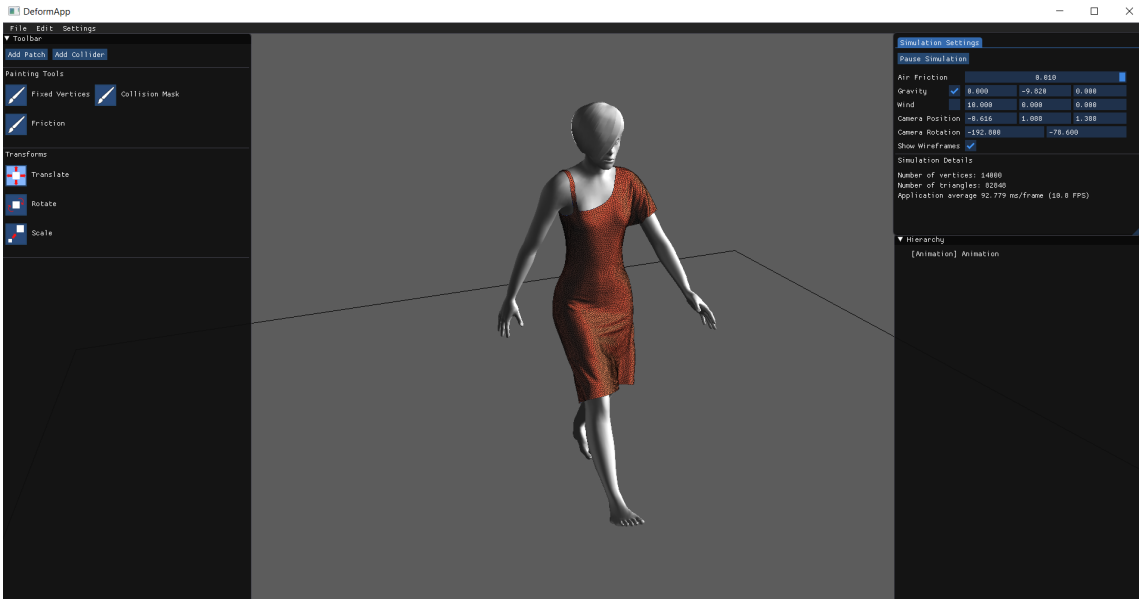

Figure 6.1: The start view of the application, showing a scene containing an imported animated characters wearing a dress.

6.1.2 Object Details View

Figure 6.2 displays how the application looks after a user has added a deformable patch to the scene by clicking the "Add Patch"-button, and then selecting it either by clicking the object on the screen or clicking on its name in the hierarchy. The "Object Details"-tab view shows some configurable parameters for the currently selected object. Example parameters are object size, position, scale and rotation. Similar to the global parameters under the "Simulation Settings"-tab view, configuration is done through direct manipulation. Altering one of the parameters previously mentioned would result in an immediate change to the affected object.

At the center of the selected object, a transformation gizmo can be seen. The transformation gizmo is more specifically a translation gizmo, which is because the

translation tool is the currently active one as can be seen on the left side in figure 6.2. By switching to one of the other transformation tools, e.g. scaling or rotation, the appropriate gizmo will be the one rendered at the objects position. By dragging the controls of the gizmo, the chosen transformation is performed along the selected axis. The name of the object is also highlighted in the hierarchy view on the right side in order to visualize that the object is currently selected.

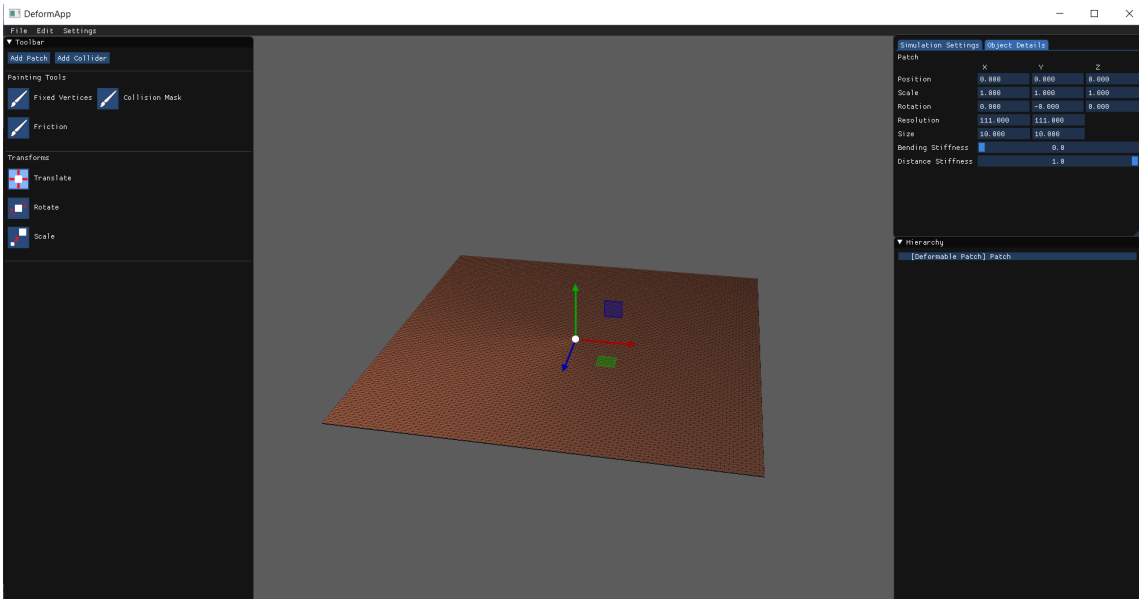

Figure 6.2: The object details view of the selected deformable patch.

6.1.3 Painting Tools

The painting tools are a collection of tool where the user can apply a per-particle configuration to an object by painting with the mouse cursor. Each vertex constructing the object represents a single particle.

When a user selects any of the painting tools, a panel containing settings for that specific tool will appear below the regular toolbar. As can be seen in figure A.10, the settings for the collision mask painting tool include changing brush radius and setting the active mask. The mouse cursor will also change from the default cursor to a white circle representing the brush. The size of the brush mouse cursor changes with the brush radius parameter.

Each different mask is represented by a color, which is shown on the object as the user paints the particles/vertices. The painted particles for each of the painting tools are only visible while that specific tool is currently active. The "Apply To All"- and "Clear All"-buttons will apply or clear the currently selected paint respectively to all particles of the currently selected object.

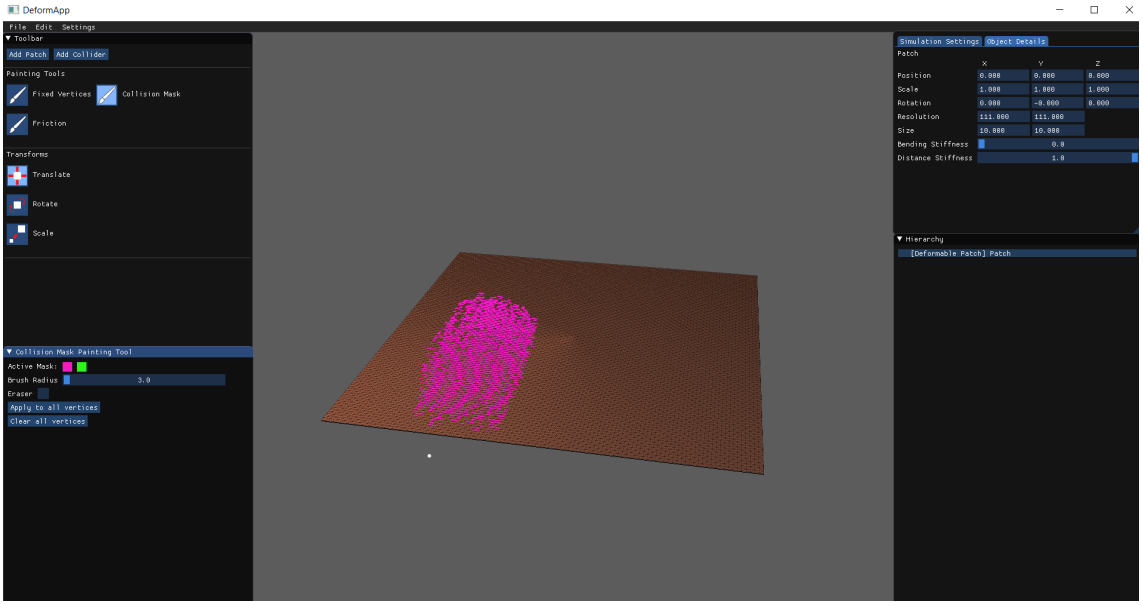

Figure 6.3: The settings for the collision mask painting tools are shown in the bottom left side. Here, a small part of the patch has been painted with the pink collision mask.

6.1.4 Running the Simulation

Once a scene has been set up, the simulation can be run by pressing the "Play Simulation"-button located at the top in the "Simulation Settings" view. Once the application is in play mode, no tools can be used to apply any changes to an object in the scene. If the user wishes to perform any changes to an object, they must first pause the simulation by clicking the "Pause Simulation"-button located at the same place where the play-button used to be in order to get back to the editor-mode. Global settings, such as altering gravity and wind values, can be altered while the simulation is running.

While in play-mode, the camera can be rotated by holding down the left mouse button and dragging the cursor. Translation of the camera along the x- and y-axis of the current field of view can be performed by holding the mouse wheel button and dragging the cursor. Clicking the right mouse button while hovering a deformable object will allow a user to drag the selected particle of the object, as is shown in figure 6.4.

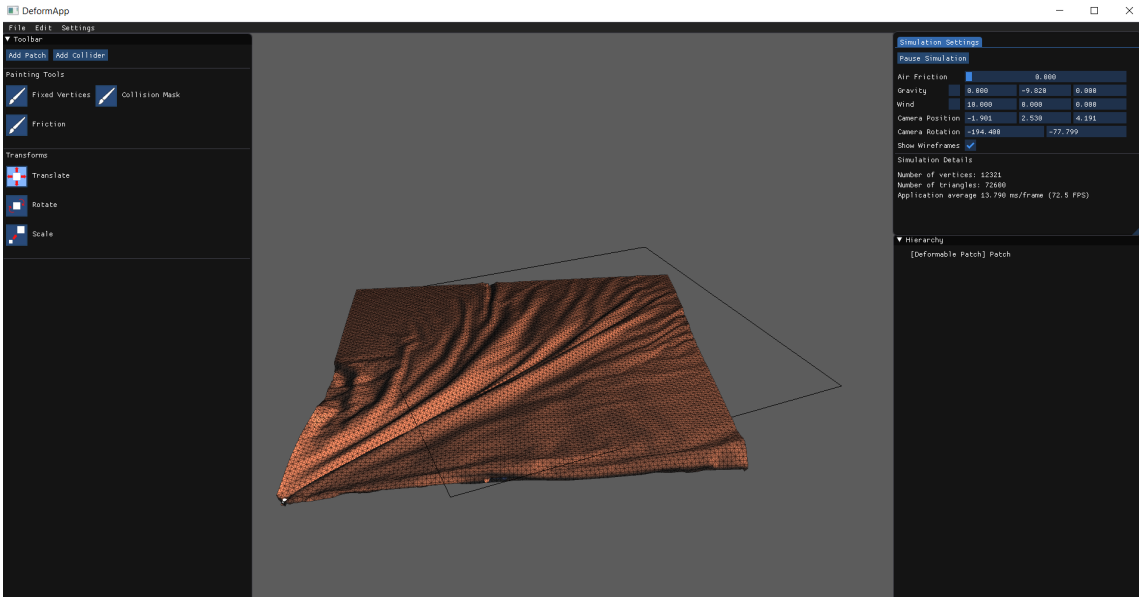

Figure 6.4: This figures shows a deformable patch in a running simulation while a user is using picking to drag it by one of its vertices towards the left. The mouse position of the user is visualized by a white dot.

6.1.5 Adding Collider Objects

If a user presses the "Add Collider"-button in the top right of the toolbar, a small popup window will appear at the location as shown in figure 6.5. The window will prompt the user to choose the type of collider object to add, providing a separate labeled button for each type. The different types of colliders that can be chosen between are axis aligned bounding box, object oriented bounding box, sphere, capsule and plane. Additionally, if a deformable object is selected, the option to add a mesh collider also exists. Upon clicking either of the buttons, the popup window disappears and the selected collider object is added to the center of the scene.

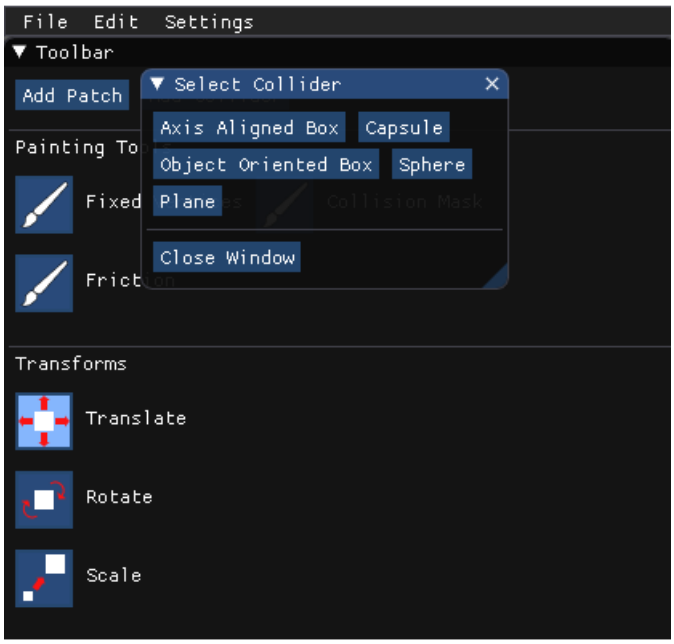

Figure 6.5: This figure shows the popup window for selecting collider type.

6.1.6 Anchoring Object to Collider

Certain collider objects can have a deformable object anchored to them, meaning the vertices of the deformable object that lie inside a collider will be attached to the that collider. Those colliders will have a text listing any object anchored to them under the object details. If no object is anchored to the collider, the anchored object will be listed as "NONE".

To anchor a deformable object to a collider, a user should first place the objects in such a way that the collider and the deformable object are overlapping each other. Then the user should select the collider. If the collider supports anchoring, an "Anchor Object"-tool will be visible on the toolbar. Clicking on the tool will prompt the user to select an object to anchor. Upon selecting a deformable object from the scene, either by clicking it directly or clicking its name from the hierarchy, the tool will be deactivated and the selected object will now be listed under the details view for the collider as shown in figure ffd.

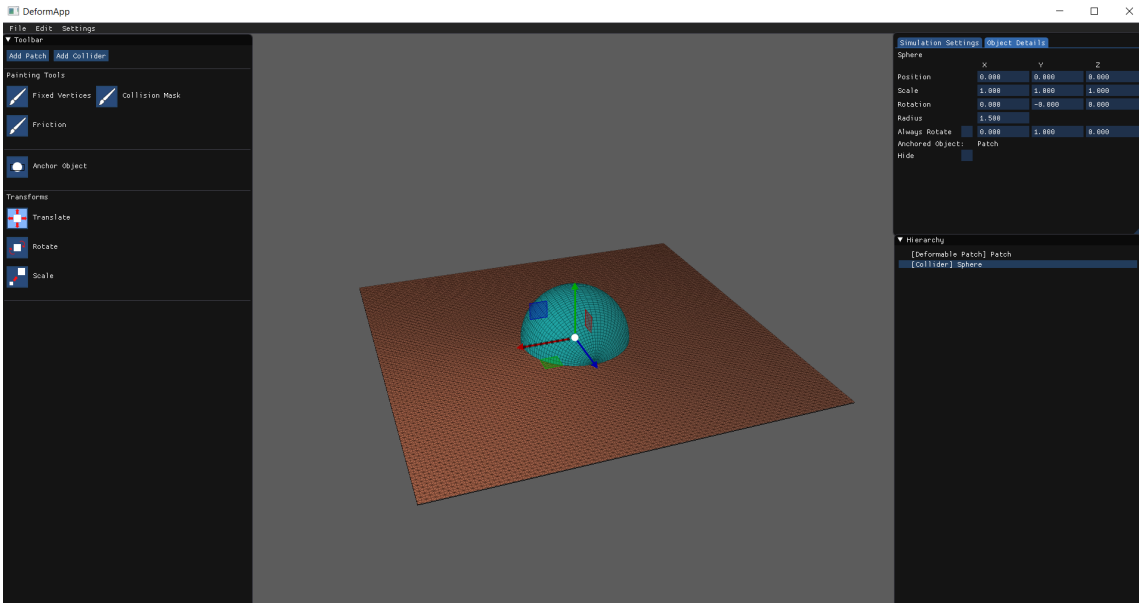

Figure 6.6: The object details view of the selected collider showing the anchored object. The "Anchor Object"-tool is also visible in the toolbar.

6.1.7 File

The "File" menu bar item is the first item located on the menu bar, at the top left of the application. The "File" drop down menu contains options for starting a new project, opening a previously saved project, saving the current project, importing different types of geometry, exporting a simulation and exiting the application as is shown in the left side of figure 6.7. The keyboard shortcut for each menu item is listed in gray next to its regular text.

Two of the menu items will open a sub-menu when clicked. This behavior can be recognized by an arrowhead at the far right of the menu item, pointing in that same direction. For the "Open Recent" item, this sub-menu will list previously opened project files. For the "Import" item, the sub-menu will list the different types of things the user can import to the application. As can be seen in the right side of figure 6.7, these things are deformable objects, animations, volumetric meshes and scenes.

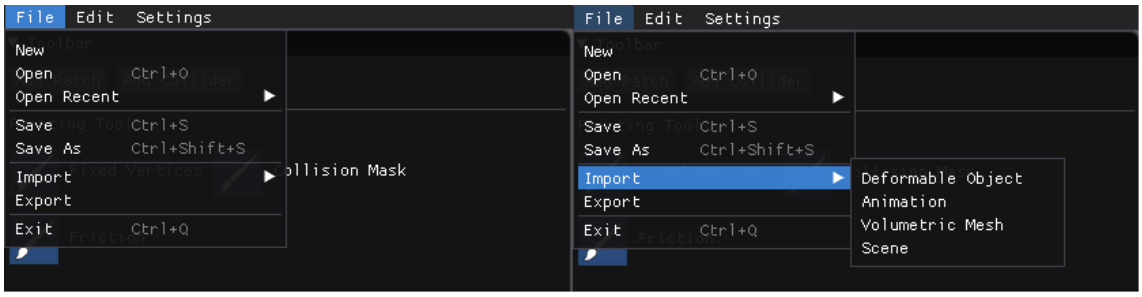

Figure 6.7: The left side of the figure shows the file drop down menu. The right side of the figure shows the sub-menu that opens when pressing the "Import"-menu item.

6.1.8 Import Animation

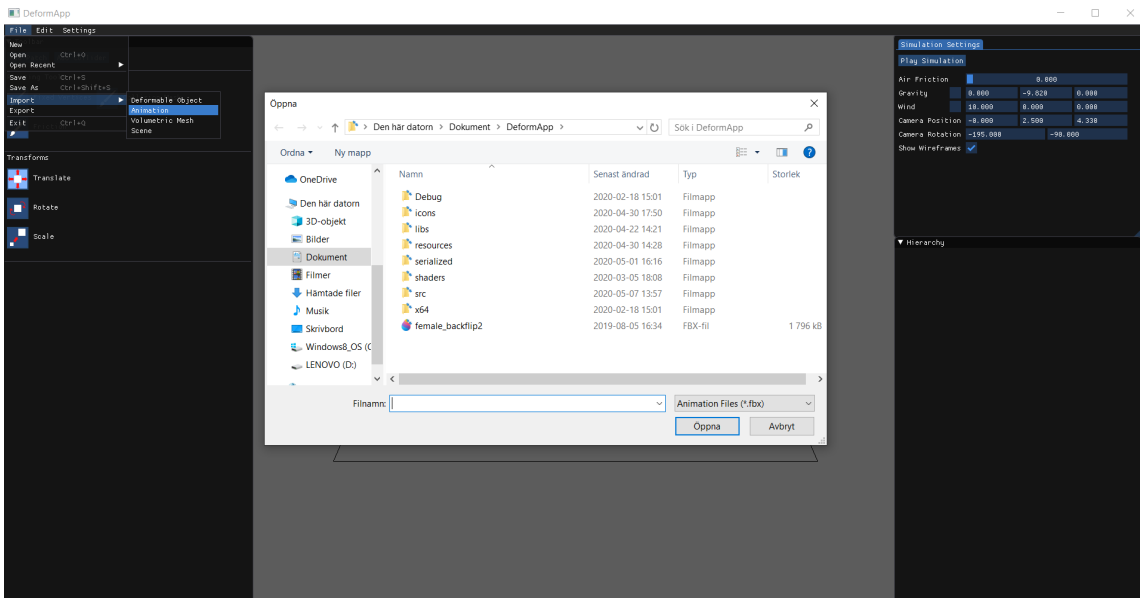

Figure 6.8: The figure shows a windows file open dialog that appears when trying to import an animation.

Importing an animation is done by clicking the "Animation"-menu item from the sub-menu opened when clicking on the "Import"-item from the File menu. Upon clicking the "Animation"-item, a standard windows file dialog will popup prompting the user to select a file. Only files with the appropriate file endings are able to be chosen, in this case .fbx files which is a common file format for animations. This is shown in figure 6.8 Once an animation file has been opened, either by selecting the item and pressing "Open" or by double clicking the item, it will be automatically added to the scene if the file contains valid data. This step is identical to the steps for importing other objects, the difference being the file endings.

When an animation is successfully imported into the application, it will appear as a static object in the center of the scene. The object details view for imported anima-

tions will contain a slider representing the time of the animation in its current state, as can be seen in figure 6.9. Dragging the slider will allow the user to visualize the entire animation, frame by frame.

Another configurable setting specific to animations is the ability to load serialized data, as can also be seen to the left in figure 6.9. This feature allows a user to load a file containing serialized data of clothes that an animated character should use. Similar to the import functions, pressing the "Load Serialized Data"-button will open a windows file dialog prompting the user to load a file of appropriate file ending. A successfully loaded configuration will be listed in place of the button as can be seen to the right in figure 6.9. A trash can icon next to the name of the loaded data file allows the user to remove the loaded data.

In order to play the imported animation, the user must start the entire simulation by pressing the "Play Simulation"-button.

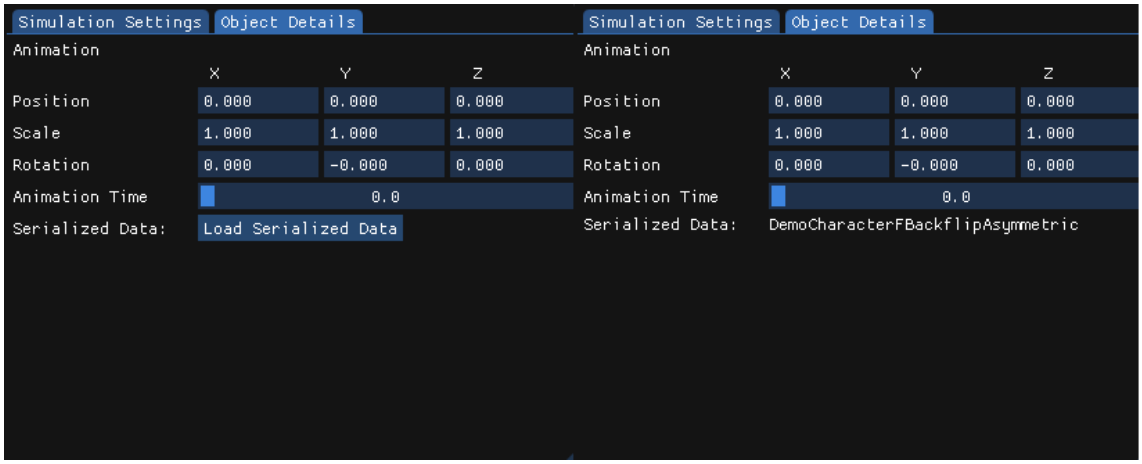

Figure 6.9: The left side of the figure shows the animation details without any serialized data loaded. The right side of the figure shows the same view after serialized data has been loaded.

6.1.9 Edit

The "Edit"-menu bar item opens a drop down menu containing some convenience functions for the user, shown in figure 6.10. These functions are undo which undoes the latest action, redo which redos the last undone action, cut which copies and removes the selected objects, copy that copies selected objects, paste that pastes copied objects and delete that removes selected objects from the scene. The keyboard shortcut for each function is listed next to their name.

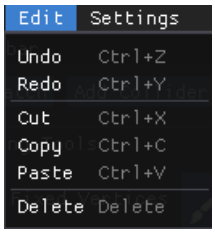

Figure 6.10: The figure shows the drop down menu appearing when selecting the "Edit"-menu item from the top menu bar.

6.2 What to Consider when Designing and Implementing an Interactive 3D Modeling Tool for Creating and Managing Generic Soft Bodies?

The research question was answered using knowledge gathered from the process of this project, including everything from early research to finishing developing the application. When developing the 3D modeling software, the following are the main points that had to be considered:

- What external libraries to use
- What work process to use
- What features should be available
- How to design for different user groups
- How to make the code modular

6.2.1 What External Libraries to Use

The first thing that has to be considered is what external libraries to use in the development of the application. For this project, most of the external libraries were considered for a specific functionality at the start of the task for implementing that functionality. Whenever a functionality could be provided by an external library, it first had to be decided if an external library should be used. This was done by approximating the tradeoff between the pros and the cons of including a library rather than coding the functionality from scratch.

Advantages of External Libraries

An advantage of using an external library instead of making some functionality from scratch is the time saved. This is usually one of the main motivations behind including external libraries. Another advantage is that it promotes software modularity, as libraries are written as separate modules. However, external libraries can be either self-contained or depend on additional dependencies which is something that also has to be considered. The final advantage is that the code has usually been tested, meaning that developers can rely on the functionality provided by the library to be fully working. This can also save time that would otherwise be spent on testing and potential bug fixing of self-implemented functionality [53].

Disadvantages of External Libraries

A disadvantage of using external libraries is that your project will get an additional dependency. This might result in lots of work having to be put into refactoring the application if the external library has to be switched for another one for some reason. Using many external libraries in an application might also cause issues such as dependency conflicts, which might be complicated to fix. Another disadvantage is that popular, open-source libraries might be targeted by hackers thus resulting in a security issue with your application [53].

Guidelines for Choosing a Library

If it is decided that an external library is needed for a certain functionality, these set of guidelines resulting from this project work can be used to help decide which library should be used:

- Use popular libraries, as they are more prone to being of high quality.
- Use open-source libraries. Being open-source makes it easier to validate code in terms of performance and security, and gives the developers more control.
- Use libraries that can perform several of the functions that your application needs. Two birds with one stone.
- Use libraries with as little additional dependencies as possible, as it makes it more reliable in terms of maintainability.

These guidelines are not based around anything specific to the development of the 3D modeling tool, but are more generally applicable to any software development project. For this specific project, the following functionalities were implemented with the use of external libraries and are considered to be particularly useful when developing 3D modeling software:

- Rendering the GUI
- Importing objects from files
- Adding transformation gizmos
- Serialization of data
- Various libraries for various math calculations, including ray-casting.

All of these external libraries were chosen during the execution of the project by following the guidelines mentioned above.

6.2.2 What Work Process to Use

Agile software development is today becoming the most popular way to work with software development [54]. However, there are many different methods available to do so and different aspects of the project might affect which is the optimal choice. These aspects include the size of the development team, how long the project will run for and how often there will be meetings with the stakeholders. The important thing is not to choose a specific method and sticking to it strictly, but rather to allow the work to be flexible and easily adapted to changes as is the main principle of any agile work process.

6.2.3 What Features should be Available

Deciding which features should be in the application is one of the considerations that contributes the most to how the application will turn out. In this project, there were a few factors that heavily contributed to this: functionality offered by the Dynamo SDK, features available in popular 3D-modelling tools, findings from user testing, discoveries during the software implementation phase and the time frame for the project combined with the size of the software development team. The functionality offered by the Dynamo SDK was further based on research into the needs of the user group performed by Deform Dynamics prior to the start of this project. Furthermore, the user tests conducted were also based on what was considered to be the most typical use scenarios for the developed application. This was data provided by Deform Dynamics at the start of the project.

The following is a list of features to be considered when developing a 3D modeling tool:

- Prioritize features that highlights the unique functionality of your application. For example if your app outshines the competition in regards of a specific kind of animation type, prioritize features that aid the creation of those animations.
- Allow a user to add different types of object with ease.
- Allow a user to import objects and animations from commonly used file formats
- Allow a user to perform transformations of objects.
- Allow the user to undo/redo actions.
- Allow the user to copy/paste objects.
- Allow the user to rotate and move the camera around the scene freely with the mouse.
- Allow a user to save and load scenes to/from project files.
- Avoid putting performance-sensitive functionality in the areas of the application where the user spends the most time. An example of this includes not having the simulation always running by default; instead play the simulation with the press of a button, allowing the application to run smoother for the majority of the time.

6.2.4 How to Design for Different User Groups

Throughout the project, various research and usability testing regarding design guidelines was conducted. Previous theory included was targeted towards software applications in general, while the practical exploration of this project was used to determine what aspects could be specifically applicable to designing 3D modeling software. This practical exploration refers to both analysing existing 3D modeling software and the development of a 3D modeling tool.

The resulting points to consider when designing a 3D modeling tool is presented in the list below:

- Use a sovereign posture for the application.
- Allow direct manipulation of objects and configurable settings
- Allow the user to get an overview of the scene, listing all present objects.

- Give the user immediate, modal feedback upon performing an action.
- Allow expert users to perform actions with keyboard shortcuts.
- Minimize excise in order to promote user flow.
- Allow a user to perform the same action in different ways, for example transforming an object with gizmos or by setting parameters through the GUI
- Allow users to change position of panels and other elements in the application.
- Place the most used tools so that they are easily accessible from everywhere within the application.
- Allow a user to drag files from the desktop into the application as an alternative way to import objects.

6.2.5 How to Make the Code Modular

Using a modular programming approach is an important point for making the software easily maintainable, alongside choosing external libraries carefully and using an appropriate agile software development method [55]. With a modular programming approach, each general functionality of the application is handled by a separate module. From the development of the 3D modeling tool in this project, different functionalities needed in the application were discovered as the project went on. This resulted in the following set of recommended modules:

- A GUI module for handling the graphical user interface.
- A render module for rendering objects in the scene.
- A shader module for loading vertex- and fragment-shaders.
- An importer module for importing objects from some file formats.
- A camera module for handling camera parameters.
- A manager module for handling underlying computations such as serialization and computing vertex- and index-data for objects.
- A general module that connects the other modules and starts the application.

7
Discussion

In this chapter, various parts of the project will be discussed. First, the tool built will be discussed in detail. After that, the various stages of the work process will be discussed. This includes the research phase, design phase, and software development phase. Finally, ethical implications arising from this project as well as suggestions for future work in the research area is discussed.

7.1 The Tool

The list of features included in the tool have been based on analysing competing 3D modeling tools, findings from the early user tests as well as the available functionality from the Dynamo SDK which was the basis for the animations produced by the application. The Vivace-solver producing the physics in the simulations is capable of generating plausible soft-body animations at faster rates than competing physics engines, which motivates the area of focus of this tool [8]. Hence, the design has been based around producing this type of animations, compared to competing 3D modeling tools where there is usually a broader set of features.

The tool in its current state manages to create soft body animations with just a few clicks. If a user does not want to rely on external files to be imported, the animations that can be created are mostly simple animations consisting of a few objects with the intention to showcase visually plausible cloth simulations. These types of simulations include flags waving in the wind, cloth being dragged or stretched, patches of cloth colliding with objects of different shapes and simulations with volumetric meshes. If files containing animated characters and full-fledged garments are available, advanced animations simulating the real-life behavior of cloth as a character performs some action can be generated with a few clicks only as well. This is primarily done by utilizing the import features of the application.

The main reason behind allowing a user to add a patch of cloth to the scene with a single click is because this is the most basic kind of simulation when it comes to physics-based animations with cloth. The demos that the company had produced previously in order to showcase their physics engine, were often different simulations with patches of cloth being affected by various external forces ranging from object collisions and forces such as wind and gravity. Being able to reproduce such simulations with ease was thus one of the main goals of the design.

The sewing feature, which would allow a user to stitch patches of cloth together in order to form clothes, is possibly the most important feature that is not part of the resulting prototype. Not having the feature available makes it so that any animation involving garments must have those garments stored as a .obj file prior to using the application. This in turn might force a user to first use another 3D modeling tool in order to create that piece of garment. However, not including that feature was a deliberate delimitation motivated by the time frame for the project.

While some features that were desired in the finished tool were not implemented, such as the sewing feature, a goal of the design was that it should still support adding them in the future. For example, the sewing feature can easily be made available by adding an appropriate icon to the toolbar. By selecting the tool, a panel containing configurable parameters for that specific tool would appear at the same location that the panels for the various painting tools appear in the current prototype. Hence, the design supports not only the current implemented prototype of the application but also the potential finished version of the application.

Generating advanced geometry from scratch, such as a 3D character, was never a part of the intended functionality for the application. While one could argue that it would be nice to be able to do everything needed for an advanced character animation in the same place, this tool is more concerned with being able to produce soft-body animations well. This allows this tool to provide a much cleaner GUI than competing 3D modeling tools, resulting in it being less complicated to use.

Initially, the animations were intended to be active at all times thus allowing a user to manipulate an animation with real-time feedback. This was however changed after some crash issues with using the approach was discovered. The crash issues aside, this part of the software development gave rise to some interesting discoveries. What was found was that adding and manipulating objects while a simulation was not running resulted in a much smoother experience for the user, thus providing them with a better experience overall. This further supports the usefulness of the method of prototyping as an ideation technique [56].

7.2 The Considerations

The considerations, or guidelines, resulting from this project are based on previous research in the area as well as knowledge gained from developing a 3D modeling tool. They have varying levels of concreteness depending on what aspect is considered. For example, the guidelines regarding external libraries do not give concrete external libraries to consider and neither do the guidelines regarding work processes. That is because it was not considered possible to answer those questions with that level of preciseness while still keeping the guidelines applicable to other projects than this one. In comparison, the considerations regarding what features should be available, how to design for different user groups and how to make the code modular all were able to provide more concrete examples of things to consider. The main reason behind that is that they are considerations regarding things that depend on too

many factors. This way, the resulting guidelines are considered to be applicable to the development of any kind of 3D modeling tool.

7.3 Execution and Process

This section will discuss the entire process of this project, going through all the phases and discussing them in detail.

7.3.1 Research Phase

During the research phase, much research was put into exploring other competing 3D modeling tools as well as information required for the software development part. This research included optimal programming language, available external dependencies and the Dynamo SDK. This step gave a lot insights that played a pivotal part in the success of the project. Most importantly, it provided an insight as to how 3D modeling tools are designed, and how they are typically used. It also gave insights to how functions within such programs are expected to behave.

Throughout the project, a need for new external libraries that had not previously been researched occasionally arose. This was either because a new functionality was found to be needed, or that a previously chosen library would fail to live up to what was expected from it. However, the fact that the project was using an agile work process combined with the modularity of the code made it easy enough to either swap one library for another or simply incorporating a new one into the application. This was also possibly due to the choice of writing the software in C++ instead of Python, as there are much more libraries written in C++ and that there would be no limitations arising from a specific library lacking Python bindings. The choice of using C++ was altogether considered a good choice as the compilation time, which was the main argument against using it, was considered negligible in hindsight.

7.3.2 Design Phase

The design phase begun right after the research phase. The decision to not do a competitor analysis of existing 3D modeling tools and explore them thoroughly before sketching on the design was considered a good choice. While doing a competitor analysis earlier might have sped up the early stages of the design phase, reducing bias allowed the design process to yield a more unique design.

Performing cognitive walkthroughs in the early stages of the design phase before testing with real users was a time-efficient method for improving the design. This made the design good enough to perform real user testing on it without wasting too much time. Testing the resulting paper prototypes on real users allowed the design to be more influenced by actual user needs, more specifically beginner- and intermediate-level users where the latter was the main goal. The data given by these user tests was made even more important as real users were excluded from the evaluation stage because of the current pandemic, making this the only time real

user input was received from a non-expert user.

Having the design roughly finished before starting the software development was good as it allowed external dependencies to be chosen based on what features should be available in the finished prototype. At the same time, there were a number of reasons for why the design was not fully finished before starting the implementation.

The main reason is that the process of prototyping has been shown to be a good source for ideation, leading to new design decisions being taken during the process [56]. For example, the "Simulation Settings"- and "Object Details"-tabs have switched places in the implemented prototype from where they originally were in the mockup of the design. While it was initially implemented this way by accident, in the end it was considered to give the application a better flow.

The second reason is that it sometimes can be useful to have different tasks to work on. For example, having both design details to work on as well as implementation would further allow the two tasks to be switched between if something would prevent immediate continuation of either one.

The final reason is that the company requested a demonstration of the first set of features by a certain date, which would mean that it was favorable to start working on them at a fixed time (rather than when the design phase has reached a certain point) if that request was to be met. This further aligned with the personal preference of getting started with the software development early on.

The biggest downside of the design phase was the lack of user evaluation on the finished prototype, which was due to the COVID-19 pandemic. Instead, the usefulness of the prototype was determined by the developer together with the company supervisor. While evaluation conducted by real users would provide more reliable data, the evaluation performed together with the data from user testing on the digital mockup was deemed good enough considering the circumstances.

7.3.3 Software Development

In this section the software development stage will be discussed. The discussion is split into two parts: a discussion of the software development methods used and a discussion of the actual implementation.

Method

The software development process was conducted using a mix between some popular methods for agile software development. However, one major thing differed how the methods were used in this project compared to typical usage: this project was conducted by a single person. The reason this matters is because one of the major differences between some of the popular methods is how communication between the development team is handled. Furthermore, steps where the development team discusses various matters such as daily Scrum meetings in Scrum also become

a bit different when the team is a single person. Assigning tasks to different persons became easy, as all tasks would be performed by the same person. Dividing tasks so that there would be no merge conflicts was also a non-issue as the project guaranteed that no two persons would accidentally overlap each others work. This also resulted in tasks being able to be started late in the week, since there would be no negative consequences of a task being only half-way finished by the end of a week.

The project used sprints that were one week long. This was mainly to be able to present the weeks work to the company during the weekly meetings on Fridays, making each meeting function as the stakeholder demonstration that is typically included in the sprint review. While this worked well in general, demonstrating work was not something that was required each meeting by the stakeholder and hence the sprints could have been for example twice as long with no consequences in that sense.

Not having to do any communication within the team further resulted in the visualization of tasks being more flexible. Instead of a formal medium such as a Kanban board, a simple text file was used listing all different tasks that were not yet completed. This worked well as it was custom tailored

Implementation

The main focuses of the implementation was to ensure the best user experience in terms of performance, promote software modularity in order to make the tool easier to manage and develop and finally to minimize the use of external dependencies.

Performance was ensured by the choice of programming language and by paying attention to internal data structures and algorithms used to achieve certain functionality. It was further achieved by implementing the feature to run the simulation with the click of a button, putting the computations performed in order to visualize the animations at a separate location from where the user performs the work in the application.

The modularity of the code was pursued by having a separate module, or header file, for each of the general parts of the application. Early on in the development, this was followed strictly with ease. A switch of the importer module a few weeks into the project demonstrated the modularity of the software in practice. However, some part of the modular design got lost along the way. More specifically, in the finished prototype, the module responsible for the GUI is now also responsible for functionality more fitting for some sort of manager-module as listed in the module guidelines in the result section. This functionality includes generating vertex- and index-data for deformable patches, serialization of scenes and convenience features such as copy/paste and undo/redo.

One could argue that the reason behind this undesired result is that more responsibility was pushed into an already existing module as the need for new functionalities emerged. By the time that it was clear that a new module should be introduced to offload the GUI module, the amount of work required to fix the problem made

it into a low priority task. The low priority of the task caused it to never be fixed, as there were always more urging matters that could be dealt with. However, one could argue that the reason was due to the need for a manager-module not being discovered during the planning phase, making it more of a a planning error.

Minimizing external dependencies was one of the decisions that was initially considered because the company explicitly requested it. While it was actively pursued, using an external library for any functionality always became a decision between trade off between the pros and cons of using it as mentioned under the "What external libraries to use"-section in the results chapter. For example, an external library for rendering was not included as it was not considered to be a big time saver. However, libraries for the GUI, translation gizmos, importing objects, serialization and so on were included as recreating those functionalities from scratch would be a time consuming task affecting the overall quality of the project negatively. It would be similar to reinventing the wheel. Overall, all external libraries included were dependencies that were assessed to be essential to ensure the success of the project within its time frame.

Sometimes when deciding what features to implement, focus was put on the importance of the feature in a finished product rather than the importance of the feature in regards to the research question of this project. A good example of such features are the undo/redo features and the copy/paste features. While they are necessary functionality in any 3D modeling software, they are not required to be implemented in the prototype in the sense that they are not necessary to produce any animations. Their implementation does not directly contribute to the research question; merely acknowledging that they should be there is enough. Not only was time put into making these features, but after the application had to change into being a tool where you play a simulation with the click of a button they were also refactored in order to work with the new approach. In hindsight, the time spent on all of that would have been better put to use on other things such as spending more time on the GUI aesthetics.

7.4 Ethical Aspects

Making it easier for anyone to make these kinds of animations might remove the need for people of a specific skill set. While this can be a good thing as incorporating animations will be more accessible, it might also cause certain people to lose some market value. Furthermore, increasing the accessibility of available 3D modeling tools might make that market more competitive, possibly resulting in certain products to lose their value.

7.5 Future Work

There are many things that can still be done within the area. This section lists and discusses the most important ones.

7.5.1 Validating the Results

The most important one would most likely be to evaluate the resulting 3D modeling tool with real users. This was something that was initially planned to be a part of the project, but scrapped due to the COVID-19 pandemic. Hence, it is considered to be the highest priority for future work of the project. Another alternative approach would be to validate the resulting findings by conducting a similar project. That project could then potentially build on the findings of this project and explore the area further.

7.5.2 Adding Features

Adding more features to the tool is something that was intended as long as there was enough time remaining of the project. Since the project had a finite time-scope, and that a 3D modeling tool can contain a large amount of different functionality, there are plenty of features that were not included and thus could be implemented in the future. These features include the sewing feature mentioned in previous sections, which would allow a user to sew patches of cloth together to form garments. Other features could be features available in other currently existing 3D modeling tools, such as skinning and rigging [57] [58].

7.5.3 Improving the Aesthetics

Improving the aesthetics is another thing that could be further worked on, in particular the colors of the application. As this was not a part of the main focus of the project, no conclusion can be drawn about that aspect of designing for 3D modeling software. Other visual elements could also be put some more work into, such as buttons and sliders, as they are mostly just using the default appearance that the external GUI library provided.

7.5.4 Technical Limitations

One design aspect of the application is the fact that the simulation is played with the press of a button, similar to how it is done in many existing 3D modeling tools. This approach was used in the application because of technical limitations in regards to the underlying physics engine. If that limitation would not exist, and assuming it would not cause performance issues, always running the simulation in a 3D modeling tool could be explored further in the future.

8

Conclusion

3D modeling is a broad area, with concepts ranging from generating animated character models to replicating various physics-based effects in animations. It is widely deployed in various industries today, such as the game industry, movie industry and architecture, increasing the demand for 3D modeling software.

The purpose of this thesis was to try to answer the following research question:

What to Consider when Designing and Implementing an Interactive 3D Modelling Tool for Creating and Managing Generic Soft Bodies?

In order to so, an interactive 3D modeling tool for creating and managing generic soft-bodies was developed as per the research through design methodology. One of the main features of the tool is the ability to create simulations including realistic looking cloth-physics. The resulting animations can be either simple ones, such as patches of cloth colliding with various geometry shapes, or they can be more complex ones with animated, clothed characters performing some physical actions. The latter kind would require importing the animated character and its garments from a local file. The resulting animation is mimicking the effects of real physics with a plausible result at extremely fast rates.

Before development began, research in the area was thoroughly conducted in order to get a good insight into everything regarding 3D modeling tools. This research included exploring not only currently existing tools, but also relevant literature and previous research done in the area. This research, alongside knowledge gathered from developing the 3D modeling tool, was then used as the basis for answering the research question.

As with most other topics within the field of interaction design, there can never be a single definitive answer to such a question. Instead, the result from this project is a set of guidelines to be considered when developing a similar tool. These guidelines to be considered can be summed up as the following: what external libraries to use, what work process to use, what features should be available, how to design for different user groups and how to make the code modular. While these guidelines resulted from the development of this tool, they can in no way be said to be the only definitive guidelines that may arise when developing a 3D-modelling tool. Instead, they are believed to be a good reference for someone developing their own 3D modeling software. Despite much of the features of the application revolving

around cloth animations, the resulting guidelines are considered to be applicable to 3D modeling tools of all kinds.

Bibliography

[1] T. De Moor. Photorealistic Graphics: The Future Looks Just Like Real Life, August 2018. Retrieved from: https://lab.onebonsai.com/photorealistic-graphics-the-future-looks-just-like-real-life-504f46f87879.

[2] History of computer animation (CGI). Rendering of a planned highway (1961). Retrieved from: https://computeranimationhistory-cgi.jimdofree.com/rendering-of-a-planned-highway-1961/m.

[3] The Computer Graphics Book Of Knowledge. History of Computer Graphics (CG). Retrieved from: https://www.cs.cmu.edu/~ph/nyit/masson/history.htm.

[4] M. Pharr, W. Jakob, and G. Humphreys. *Physically Based Rendering: From Theory to Implementation*. Elsevier Science, 2016.

[5] A. Babadi. What Is Physically-Based Animation?, November 2018. Retrieved from: https://towardsdatascience.com/what-is-physically-based-animation-cd92a7f8d6a4.

[6] A. Bargteil and T. Shinar. An introduction to physics-based animation. pages 1–1, 08 2018.

[7] K. Erleben, J. Sporring, and K. Henriksen. *Physics-based Animation*. Charles River Media graphics. Charles River Media, 2005.

[8] M. Fratarcangeli, V. Tibaldo, and F. Pellacini. Vivace: A practical gauss-seidel method for stable soft body dynamics. *ACM Trans. Graph.*, 35(6):214:1–214:9, November 2016.

[9] Wikipedia. Autodesk Maya, 2020. Retrieved from: https://en.wikipedia.org/wiki/Autodesk_Maya.

[10] Software Suggest. AboutCLO 3D Fashion, 2020. Retrieved from: https://www.softwaresuggest.com/clo-3d-fashion.

[11] J. Petty. What is Houdini What Does It Do?, 2020. Retrieved from: https://conceptartempire.com/what-is-houdini-software/.

[12] Wikipedia. Autodesk 3ds Max, 2020. Retrieved from: https://en.wikipedia.org/wiki/Autodesk_3ds_Max.

[13] Deform Dynamics. Unity plugin, 2020. Retrieved from: https://docs.deformdynamics.com/#unity-plugin.

[14] W. Gaver. What should we expect from research through design? *Conference on Human Factors in Computing Systems - Proceedings*, 05 2012.

[15] Tutorials Point. Interactive System Design, February 2020. Retrieved from: https://www.tutorialspoint.com/human_computer_interface/interactive_system_design.htm.

[16] A. Cooper, R. Reimann, D. Cronin, and C. Noessel. *About Face*. John Wiley & Sons, Inc, 2014.

[17] R. Friis Dam and Y. Siang Teo. 5 Stages in the Design Thinking Process, February 2020. Retrieved from: https://www.interaction-design.org/literature/article/5-stages-in-the-design-thinking-process.

[18] Interaction-Design.org. 5 Stages in the Design Thinking Process, December 2019. Retrieved from: https://www.interaction-design.org/literature/article/design-iteration-brings-powerful-results-so-do-it-again-designer.

[19] M. Isherwood. Stop copying, and start doing competitor UX analysis properly, April 2018. Retrieved from: https://uxplanet.org/stop-copying-and-start-doing-competitor-ux-analysis-properly-bff8dbfc644f.

[20] J. Rojas. Etch A Sketch: How to Use Sketching in User Experience Design, December 2019. Retrieved from: https://www.interaction-design.org/literature/article/etch-a-sketch-how-to-use-sketching-in-user-experience-design.

[21] N. Babich. The Magic of Paper Prototyping, September 2018. Retrieved from: https://uxplanet.org/the-magic-of-paper-prototyping-51693eac6bc3.

[22] P. Yadav. Wireframes in UX Design — What, Why, When and How?, May 2019. Retrieved from: https://blog.prototypr.io/wireframes-in-ux-design-what-why-when-and-how-ff07bb513c89.

[23] A. Micallef. Wireframing, Prototyping, Mockuping – What's the Difference?, February 2018. Retrieved from: https://speckyboy.com/wireframing-prototyping-mockuping-whats-the-difference/.

[24] Interaction-Design.org. How to Conduct a Cognitive Walkthrough, 2018. Retrieved from: https://www.interaction-design.org/literature/article/how-to-conduct-a-cognitive-walkthrough.

[25] B. Martin and B. Hanington. *Universal Methods of Design*. Rockport Publishers, 2012.

[26] E. Wong. Heuristic Evaluation: How to Conduct a Heuristic Evaluation, January 2020. Retrieved from: https://www.interaction-design.org/literature/article/heuristic-evaluation-how-to-conduct-a-heuristic-evaluation.

[27] J. Nielsen. 10 Usability Heuristics for User Interface Design, April 1994. Retrieved from: https://www.nngroup.com/articles/ten-usability-heuristics/.

[28] B. Kopf. The Power of Figma as a Design Tool, 2018. Retrieved from: https://www.toptal.com/designers/ui/figma-design-tool.

[29] Wikipedia. Adobe XD, 2020. Retrieved from: https://en.wikipedia.org/wiki/Adobe_XD.

[30] Agile Alliance. What is Agile?, 2020. Retrieved from: https://www.agilealliance.org/agile101/.

[31] Agile Manifesto. Principles behind the Agile Manifesto, 2001. Retrieved from: https://agilemanifesto.org/principles.html.

[32] Scrum.org. WHAT IS SCRUM?, 2020. Retrieved from: https://www.scrum.org/resources/what-is-scrum.

[33] Scrum.org. What is a Sprint in Scrum?, 2020. Retrieved from: https://www.scrum.org/resources/what-is-a-sprint-in-scrum.
[34] Scrum.org. What is Sprint Planning?, 2020. Retrieved from: https://www.scrum.org/resources/what-is-sprint-planning.
[35] Scrum.org. What is a Daily Scrum?, 2020. Retrieved from: https://www.scrum.org/resources/what-is-a-daily-scrum.
[36] Scrum.org. What is a Sprint Review?, 2020. Retrieved from: https://www.scrum.org/resources/what-is-a-sprint-review.
[37] Scrum.org. What is a Sprint Retrospective?, 2020. Retrieved from: https://www.scrum.org/resources/what-is-a-sprint-retrospective.
[38] Agile Modeling. Feature Driven Development (FDD) and Agile Modeling, 2018. Retrieved from: http://www.agilemodeling.com/essays/fdd.htm.
[39] CollabNet VersionOne. What Is Kanban? An Introduction to Kanban Methodology, 2020. Retrieved from: https://resources.collab.net/agile-101/what-is-kanban.
[40] Wikipedia. Modular programming, 2020. Retrieved from: https://en.wikipedia.org/wiki/Modular_programming.
[41] Wikipedia. Interface-based programming, 2020. Retrieved from: https://en.wikipedia.org/wiki/Interface-based_programming.
[42] Wikipedia. Module pattern, 2020. Retrieved from: https://en.wikipedia.org/wiki/Module_pattern.
[43] O. Cornut. dear imgui, 2020. Retrieved from: https://github.com/ocornut/imgui.
[44] Wikipedia. Qt (software), 2020. Retrieved from: https://en.wikipedia.org/wiki/Qt_(software).
[45] Wikipedia. OpenGL, 2020. Retrieved from: https://en.wikipedia.org/wiki/OpenGL.
[46] GLFW.org. GLFW, 2020. Retrieved from: https://www.glfw.org/.
[47] libsdl.org. About SDL, 2020. Retrieved from: https://www.libsdl.org/.
[48] Khronos.org. OpenGL Loading Library, 2020. Retrieved from: https://www.khronos.org/opengl/wiki/OpenGL_Loading_Library.
[49] Assimp. Open Asset Import Library (assimp), 2020. Retrieved from: https://github.com/assimp/assimp.
[50] magnum engine. magnum engine, 2019. Retrieved from: https://magnum.graphics/.
[51] Laura. Python vs. C++: Let's Compare, January 2020. Retrieved from: https://www.bitdegree.org/tutorials/python-vs-c-plus-plus/.
[52] S. Parent. Inheritance Is The Base Class of Evil, 2013. Retrieved from: https://channel9.msdn.com/Events/GoingNative/2013/Inheritance-Is-The-Base-Class-of-Evil.
[53] A. Papadopoulos. SHOULD DEVELOPERS USE THIRD-PARTY LIBRARIES?, 2018. Retrieved from: https://www.scalablepath.com/blog/third-party-libraries/.
[54] J. Jeremiah. Survey: Is agile the new norm?, 2020. Retrieved from: https://techbeacon.com/app-dev-testing/survey-agile-new-norm.

[55] V. Mirzoyan. The Importance of Modular Programming, May 2020. Retrieved from: https://aist.global/en/importance-of-modular-programming.

[56] R. Friis Dam and Y. Siang Teo. Introduction to the Essential Ideation Techniques which are the Heart of Design Thinking, July 2019. Retrieved from: https://www.interaction-design.org/literature/article/introduction-to-the-essential-ideation-techniques-which-are-the-heart-of-desig

[57] 3D Ace. Skinning, 2020. Retrieved from: https://3d-ace.com/expertise/technical-expertise/skinning.

[58] 3D Ace. Rigging, 2020. Retrieved from: https://3d-ace.com/expertise/technical-expertise/rigging.

A

Images of Different Scenes

This chapter shows some images of different scenes created with the 3D modeling tool developed during this project.

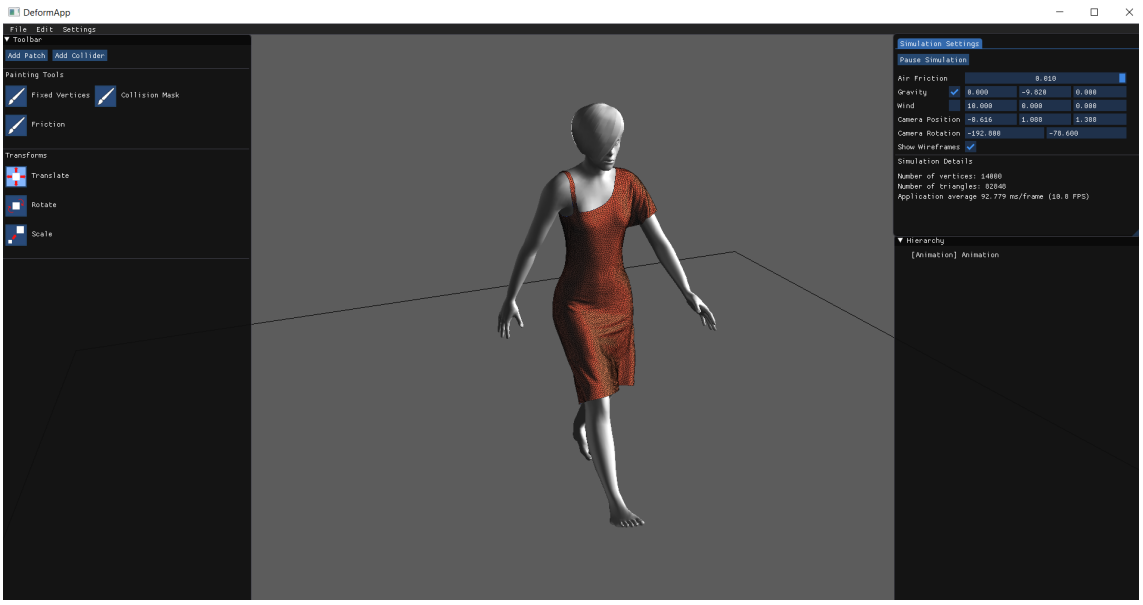

Figure A.1: A scene containing an animated walking character wearing a dress.

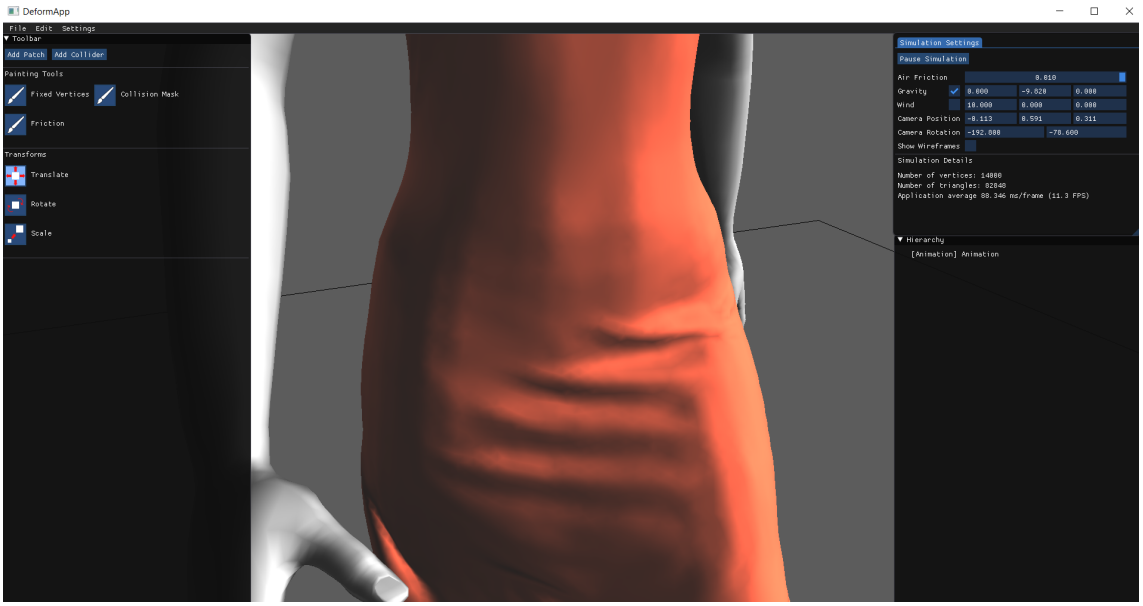

Figure A.2: This figure shows a zoomed in dress on an animated character, displaying the wrinkles in more detail as the garment follows the body.

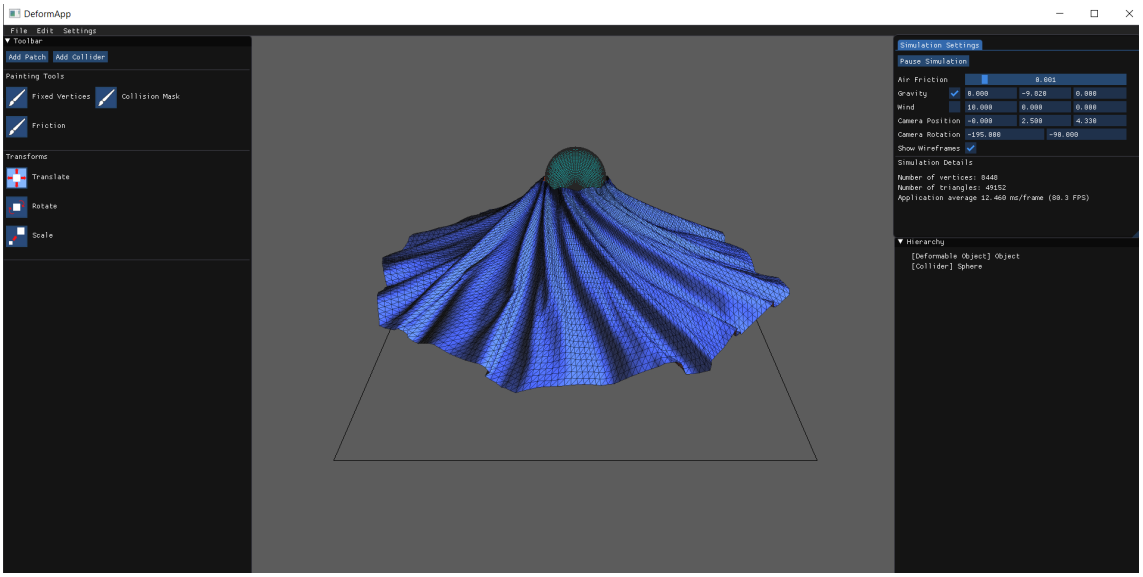

Figure A.3: This figure shows a dress being anchored to a rotating sphere collider.

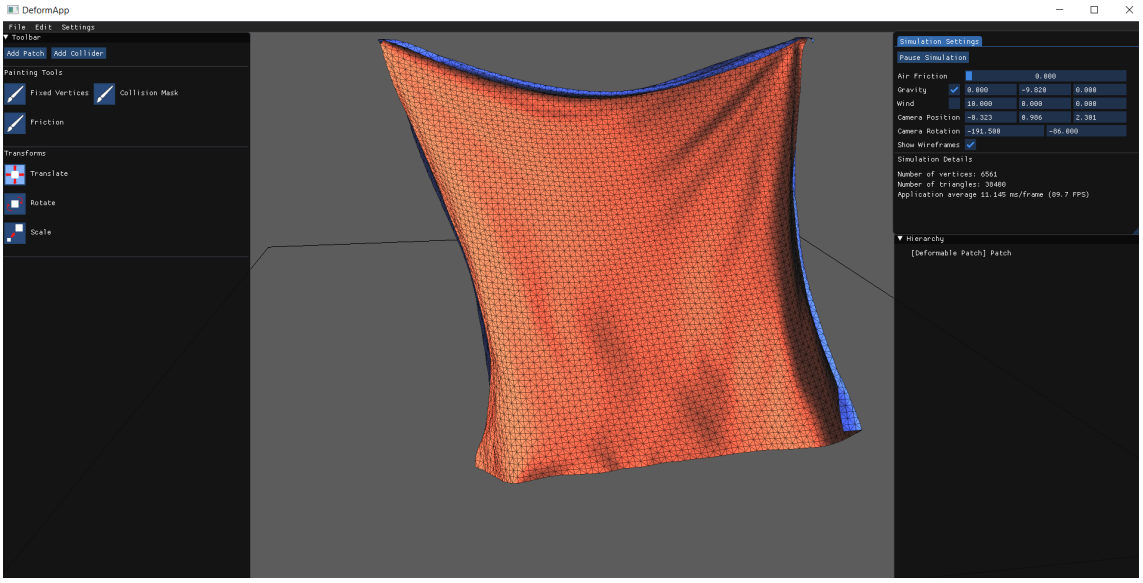

Figure A.4: This figure shows a piece of cloth hanging from its corners while affected by gravity.

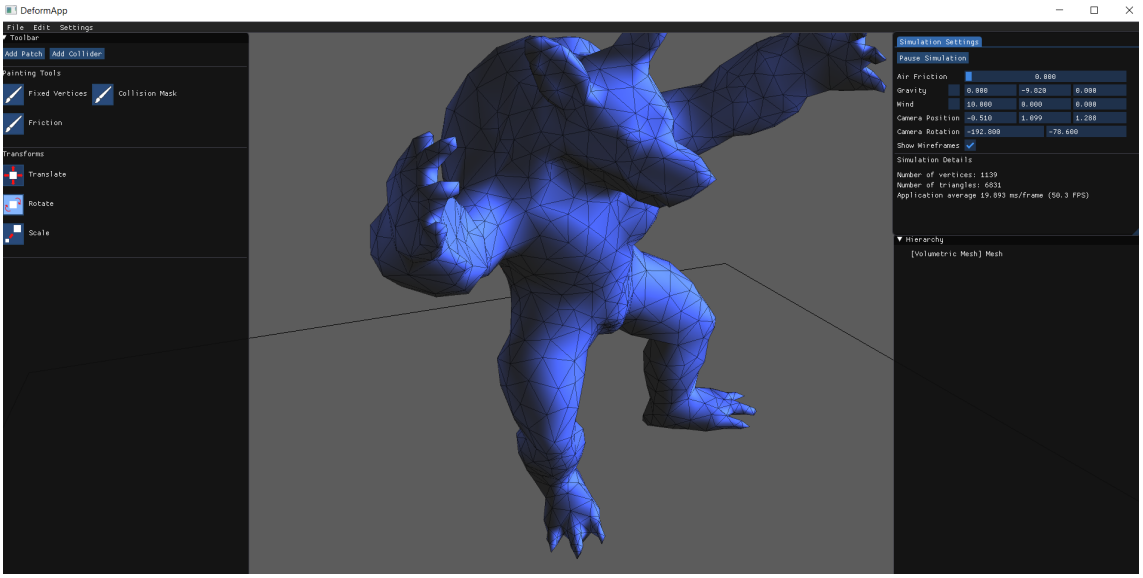

Figure A.5: This figure shows a volumetric mesh representing an armadillo that has been imported into the application.

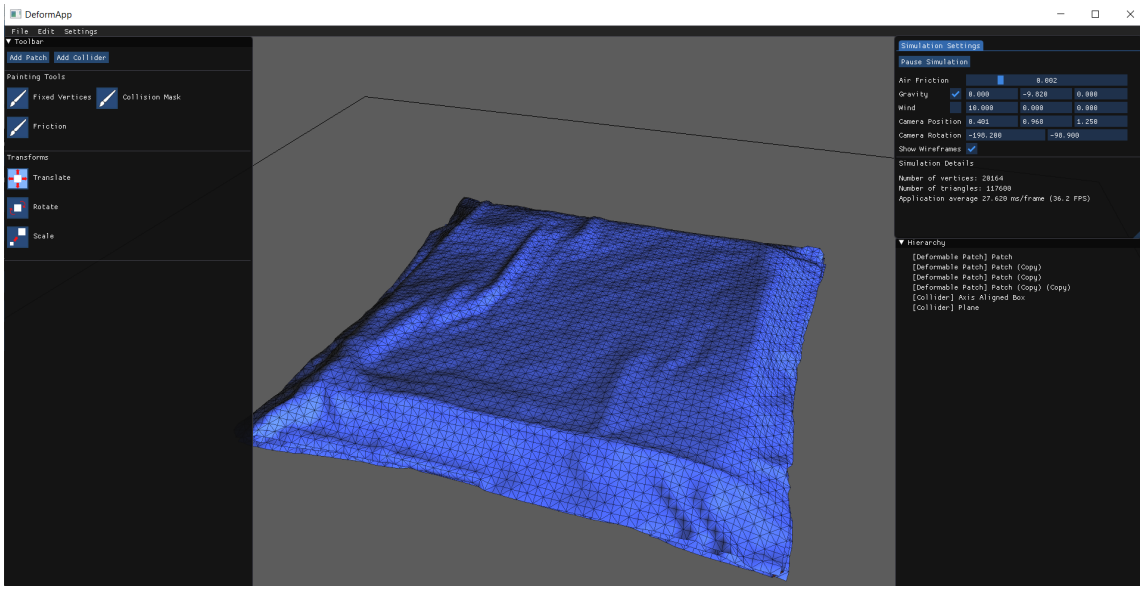

Figure A.6: This figure shows a scene where several patches of cloth are laying on top of a box.

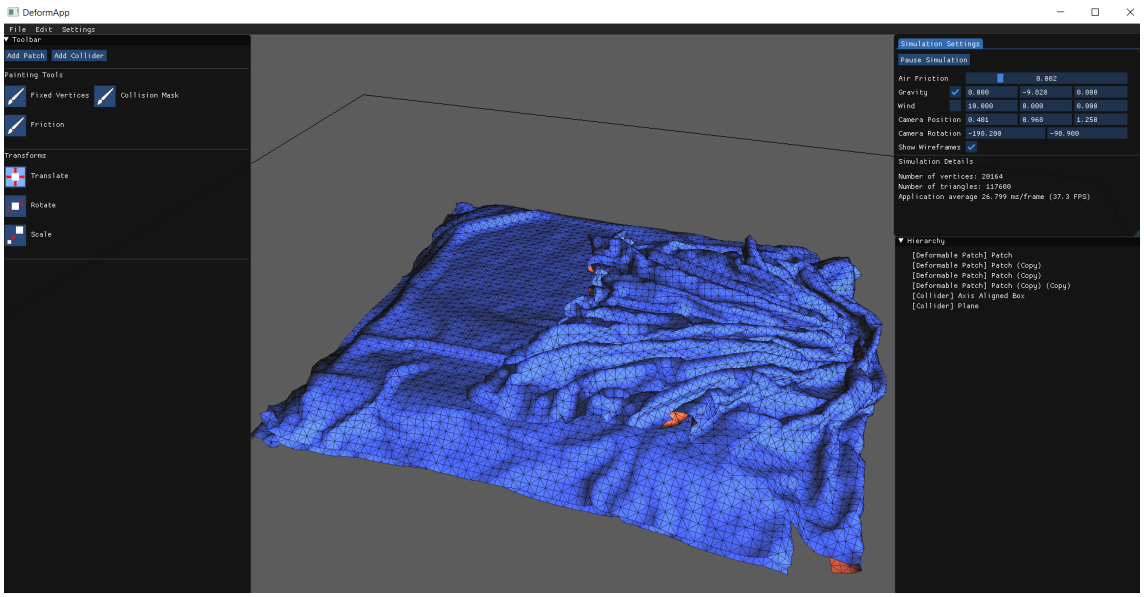

Figure A.7: This figure shows the same scene as the previous figure, except the top patch of cloth has now been dragged to the side of the box.

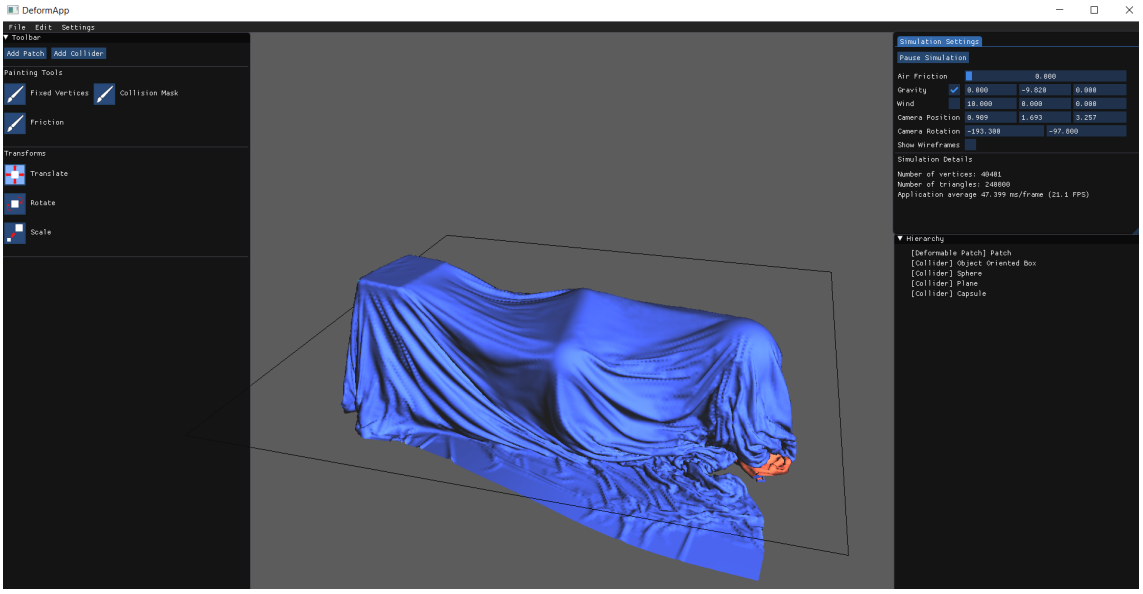

Figure A.8: This figure shows a large patch of cloth on top of a sphere-, box- and capsule-collider that rotate in different ways.

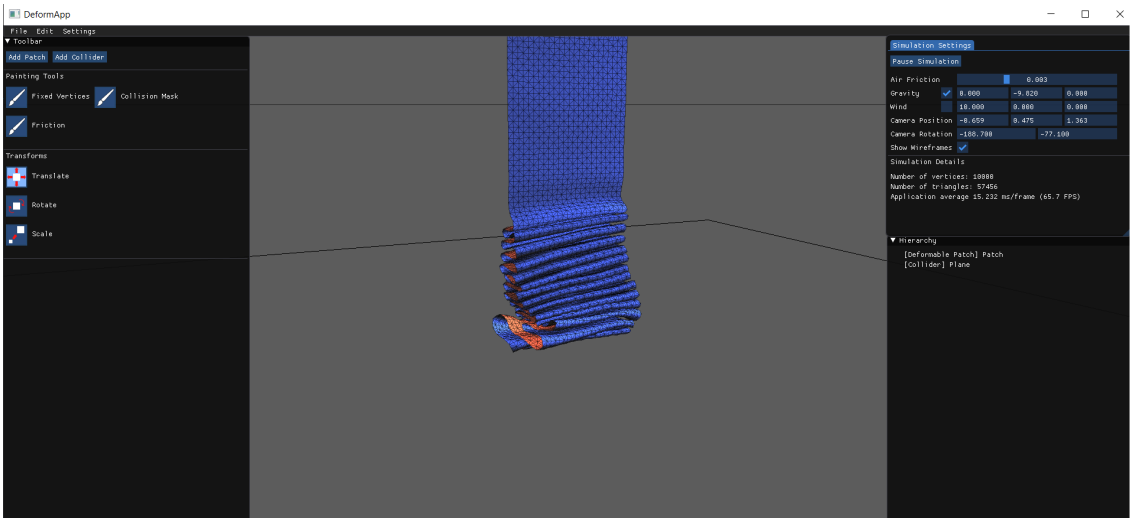

Figure A.9: This figure shows a ribbon falling to the ground.

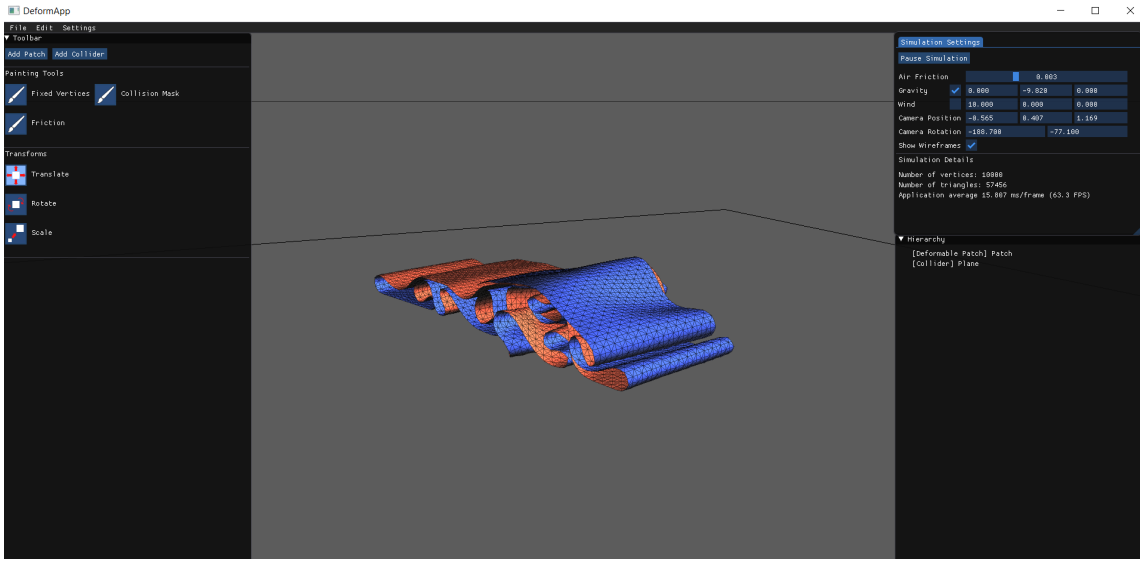

Figure A.10: This figure shows the same scene as the previous figure after the entire ribbon has fallen to the ground.