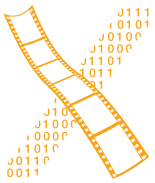

Chair for Computer Science 10 (Media Computing and Human-Computer Interaction)

RWTH AACHEN UNIVERSITY

SoRoCAD: A Tool to Design Shape Changes in Soft Robotics

Bachelor's Thesis
submitted to the
Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department
RWTH Aachen University

by
Kirill Timchenko

Thesis advisor:
Prof. Dr. Jan Borchers

Second examiner:
Prof. Dr. Ulrik Schroeder

Registration date: 20.08.2019
Submission date: 22.10.2019

Contents

List of Figures

Abstract

Due to new materials, fabrication methods, and the integration of 3-D printing into existing as well as new areas of soft robotics, the field of research is in a phase of extensive development. Parts made by 3-D printers find an application in the mechanical aspects for stabilization and stiffening, as well as in the design of flexible silicone joints in the form of molds. These 3-D printed molds enable the even faster generation of cost-effective prototypes and a clear starting point for extending existing joint models. When working with these molds, either new models are designed from the ground up, or existing models are used, which are then processed and altered in full-fledged CAD programs. This process not only requires a high level of experience with 3-D design and much time but also an understanding of the subsequent application and the behavior of the silicone used in the further course. This thesis aims at providing an introduction to the subject of soft robotics, on giving an overview of the software currently used in the field to design and construct molds and afterwards to present a software tool that allows the design of 3-D printed molds by utilizing parameterization and cell-based structures to ease and thus accelerate the current design processes.

Überblick

Aufgrund von neuen Materialien, Fabrikationsmethoden und der Integration des 3-D-Drucks in bestehende sowie neue Bereiche der Softrobotik, befindet sich das Forschungsfeld in einer Phase umfassender Entwicklung. Durch 3-D-Drucker hergestellte Teile finden sowohl in den mechanischen Aspekten zur Stabilisierung und Versteifung, als auch beim Entwurf und der Gestaltung der weichen Silikongelenke eine Applikation in Form von Gussformen. Diese 3-D-gedruckten Gussformen ermöglichen ein noch schnelleres Generieren von kostengünstigen Prototypen sowie einen erleichterten Ansatzpunkt zur Erweiterung bestehender Gelenkmodelle. Bei der Gestaltung dieser Gussformen wird bisher häufig auf bereits vorhandene Modelle zurückgegriffen die dann in vollwertigen CAD Programmen bearbeitet werden oder es werden von Grund auf neue Modelle entworfen. Dies erfordert nicht nur ein hohes Maß an Erfahrung mit 3-D-Gestaltung, sondern neben viel Zeit auch ein Verständnis für die spätere Applikation und das Verhalten des im weiteren Verlauf verwendeten Silikons. Diese Arbeit hat zum Ziel, einen Einstieg in das Thema Softrobotik bereitzustellen, sowie eine Übersicht über die aktuell in diesem Feld zur Gestaltung verwendeten Programme zu geben und anknüpfend daran ein Programm zu präsentieren, dass die Gestaltung von 3-D-gedruckten Gussformen durch Parametrisierung und auf Zellen basierenden Strukturen erleichtern und damit beschleunigen soll. Anschließend verifizieren wir die Bedienbarkeit unseres Tools, um sicherzustellen, dass der Mehrwert gegenüber handelsüblicher 3-D-CAD Software gegeben ist.

Acknowledgements

I want to thank Prof. Jan Borchers and Prof. Ulrik Schroeder for giving me the opportunity to make soft robotics a central part of my thesis and for being my examiners. Furthermore, I want to thank Anke Brocker for being my supervisor and therefore providing essential support in developing the outline and critical parts of my thesis. Finally, I want to thank everyone at RWTH Aachen Media Computing Group for playing a fundamental role throughout my student life and helping me become the person I am today.

Conventions

Throughout this thesis we use the following conventions.

Text conventions

Definitions of technical terms or short excursus are set off in coloured boxes.

EXCURSUS:
Excursus are detailed discussions of a particular point in a book, usually in an appendix, or digressions in a written text.

Definition:
Excursus

Source code and implementation symbols are written in typewriter-style text.

```
myClass
```

The whole thesis is written in American English.

Download links are set off in coloured boxes.

File: myFile[a]

[a] http://hci.rwth-aachen.de/public/folder/file_number.file

Chapter 1

Introduction

Robotics

The research area of soft robotics is a subfield of robotics. Robotics, on a higher level, is an interdisciplinary research field with the goal of the creation of autonomous, human-assisted, or human-controlled robots for use in a large number of areas [1]. The primary focus of soft robotics lies in creating parts for use in those robots, that are using highly compliant materials to achieve a behavior of parts that are mostly found in nature, thus mimicking living organisms [2]. It draws heavily from organisms that are continually adapting to their surroundings to overcome physical limitations and achieve a combination of movement patterns and body properties that are rare to find in a rigid, stiffened, and traditional robot.

Materials in Soft Robotics

To achieve these properties, soft robotics makes use of three main methods [3]. All three methods use materials that, if a physical actuator is applied, react by adjusting their inner structure, and thus changing their shape. The first method makes use of dielectric elastomer actuators (DEAs), which are a subgroup of electroactive polymers. The critical property of electroactive polymers is their possibility to change their shape when electricity is applied. Dielectric elastomer actuators combine this property with an extended elasticity. They consist of an elastomer core that lays between two electrodes. When a current is applied, those electrodes virtually reassemble an electrostatic capacitor and compress the inner core, which then expands in the remaining directions, thus creating the desired shape

change [4]. The second method uses shape memory polymers [5] and shape memory alloys. Both of which change their shape when heating is applied. Shape memory polymers restore to their initial shape when heating is applied, whereas memory alloys are stretching after the application of heat. The polymer's properties come from the usage of specific types of polymers that shrink when heating is applied. The alloys get their stretching properties because they are made from fragile metal wires.

Artificial Muscles

The last method is the central method referenced in this thesis. In this method, the shape-changing properties are achieved by using pneumatic artificial muscles [6]. Those muscles are made from a flexible tube, most commonly made out of silicone. This tube can be filled with either air or a liquid, resulting in a change of inner pressure and thus forcing the flexible body to change its shape. By incorporating specific structural patterns within the tube and by the adjustment of its wall thickness, different behavior is achieved.

Simulation and Development

The results of this incorporation of structural patterns and adjustments of wall thickness heavily rely on simulation as well as trial and error within the fabrication and design process. The Simulation Open Framework Architecture offers an efficient open-source way for modeling the mechanical behavior of the tubes and robots [7, 8]. The simulation of behavior is the central part of research in this area, while the body and the tubes design process itself receives little contribution.

Contribution

In this thesis, we present a software tool that aims at simplifying and accelerating the named design process using macOS. The software will offer a structured design pipeline based on a user-extensible model library, that provides the features set for the creation of soft body models and the adjustment of their shapes. With the support of commonly used 3D file extensions, it is easily integrated into existing design pipelines.

Chapter 2

Related work

This chapter is going to explore related work in the field of soft robotics with a particular focus on software that has the goal of being beneficial to the artificial muscle design process. It is split into the work on more simulation-driven software that restricts the user's design choices but offers a more in-depth insight into the resulting body's behavior and the work on more design-driven approaches that provide 3D modeling capabilities and resources specifically suited to soft robotics.

2.1 Simulation Focused

2.1.1 Voxelize and VoxCAD

Hiller and Lipson [9] created a simulation engine that quantitatively models the statics, dynamics, and nonlinear deformation of heterogeneous soft bodies. The library Voxelize incorporates this engine and the necessary interfaces while the user software VoxCAD[1] provides the frontend user interface that allows the construction, editing, and simulation of objects composed of voxels.

[1] creativemachineslab.com/voxcad.html

Definition: *Voxel*

VOXEL:
A voxel represents a value on a regular grid in three-dimensional space. Voxels themselves do not typically have their coordinates explicitly encoded with their values. Rendering systems can infer the position of a voxel-based upon its position relative to other voxels.

While the simplification of the bodies to voxels brings the benefit of more straightforward computation and a simpler assignment of material properties like viscosity and rigidness to individual cells and parts of the body, they limit the accuracy of the simulation and the possible complexity of used shapes and patterns. In applications where only the higher-level behavior of a created body is of interest, this software offers a valuable compromise to get a rough estimate of the artificial muscle's behavior under different stress conditions.

2.1.2 SOFA Framework

The Simulation Open Framework Architecture[2] (SOFA) [10], in its core, is an in-depth open-source framework providing an interface for a variety of high and low-level physics simulations, initially aimed for usage in medicine. It offers a robust core with the possibility of simulating not only rigid bodies but also soft tissue and multi-material bodies. Through a plugin system, its extensibility is used to integrate it into a variety of other research fields. One of which is its usage for soft robotics. There are several licensed plugins, offering not only the simulation of artificial muscles and soft-body robots but also their control. In contrast to Voxelize and VoxCAD, the simulation is not based on voxels and thus more in-depth, realistic, but also computational heavy.

[2] sofa-framework.org

2.1.3 Game Engines

Lastly, physics engines created for games often incorporate the ability for soft body and, therefore, soft robotics simulations. While many of those engines are closed source and not utilized in this field, there is some ongoing research trying to make use of their functionality concerning soft body simulation in medicine. One of which is the development of a laparoscopic cholecystectomy simulator based on the Unity game engine at the Bournemouth University [11].

Soft Bodies in Medicine

LAPAROSCOPIC CHOLECYSTECTOMY:
Cholecystectomy is the surgical removal of the gallbladder. Laparascopy enables the removal of the gallbladder while making only small incisions instead of one very large one.

Definition: *Laparoscopic Cholecystectomy*

While these game engines, in comparison to the beforementioned framework, are more straightforward and, therefore, more accessible, they still have a steep learning curve and are not suited for fast prototyping or low experienced users. Although the mentioned simulator is not aimed at soft robotics, the insights gained throughout the development process can be stepping stones for similar projects in the field of soft robotics.

2.2 Design Process Oriented

While the previously mentioned tools are fundamentals, offering the mathematical and physical backbone to provide the field of soft robotics with the necessary simulation capabilities, they do not offer entry into the research field. Those tools are mostly used by people who are already confronted with the difficulties of soft robotics design. They are not suited to get an introduction into the field and to learn the behavior of soft bodies and the idea behind their design process. This leads to soft robotics being an abstract idea many researchers outside of engineering and computer science have no access to and, therefore, little and slow inte-

Limitations of Simulation Focused Software

gration into other fields of science takes place.
Thanks to 3D printers being widely available in research nowadays, the manufacturing of the necessary molds got easier and more accessible. Several universities are pushing maker communities and offering public access to their infrastructure so that the general public has more contact with the fabrication process of 3D printed parts. One of those universities is the Harvard University[3] and more specific the Harvard Biodesign Lab[4] as a part of the John A. Paulson School Of Engineering And Applied Sciences[5].

2.2.1 Soft Robotics Toolkit

A Collection of Soft Robotics Resources

The Soft Robotics Toolkit[6] is a collection of resources dedicated to the exploration of soft robotics and soft bodies, initiated by the Harvard Biodesign Lab. It developed into the most extensive resource collection, with over 20 contributing research labs and a large community of users. It includes blueprints and files of premade soft robots, soft bodies, and electrical components, as well as the necessary software for designing and simulating them. The earlier mentioned SOFA framework is also referenced in the toolkit. A significant factor in making this toolkit widely used and suitable for beginners is the included documentary for each 3D model. This documentary not only explains the behavior and construction of named models but also offers an in-depth guide to which adjustments and modifications are possible and how their impact on the behavior of the designated part will be. Using this as a starting point, after gaining insight into the development process of soft robots, users then can proceed to work on from ground on self-designed robots and parts.

The focus in this area of research lies heavily on the technical aspect of soft robotics and on providing finished material for design processes rather than rethinking and optimizing the existing workflow. Commonly used appli-

[3] harvard.edu
[4] biodesign.seas.harvard.edu
[5] seas.harvard.edu
[6] softroboticstoolkit.com

cations try to implement as much functionality as possible concerning 3D design and currently struggle to achieve a non-clustered user interface. This is where SoRoCAD steps in.
SoRoCAD is the short form for "Soft Robotics Computer-aided Design". It aims at providing a slimmed down user interface with an optimized functionality towards the design of soft robotics muscles.

2.3 Application Areas

Besides the central usage of our application in soft robotics and its directly related fields, the application's structure and cell-based design process offer capabilities beyond soft robotics. With the creation of specialized cell designs with specific projects and goals in mind, the final resulting models can be incorporated into existing projects. Although the material choice is limited to a refined selection of appropriate materials in soft robotics, the molds generated from the designed models can be filled with any material with a fitting melting points.

2.3.1 PuPoP

One example of such out of field usage is a possible integration into the "PuPoP" project [12]. This project, in its core, aims at providing an interface worn on the palm that pops several airbags up with predefined primitive shapes to simulate grasping in a virtual or augmented reality environment. It is based around simpler primitive shaped airbags that are stacked onto each other to recreate specific, more complex shapes inspired by real-world objects. These shapes could be incorporated in cell designs, adjusted for integration into SoRoCAD, and through a combination of different materials, extend the capabilities of said palm interfaces.

2.3.2 Self-healing UI

Another project offering a possible integration is ”Self-healing UI” [13]. By utilizing a specialized composite material consisting of a self-healing polymer and carbon nanotubes in combination with materials like fabric and silicon, an interface device with self-healing, sensing, and actuation capability is created. With the addition of a software backend handling the sensing and actuation, this project extends the toolbox of human-computer interaction offering new forms of application.
The in this project created sensors are based on repetitive patterns and shapes that could be transferred into cell designs for SoRoCAD. This could accelerate the fabrication of said user interfaces.

Chapter 3

Development of SoRoCAD

While this project heavily resolves around its implementation and backend functionality, there will be little to no code references within this chapter. If you are interested in the technical aspect of SoRoCAD, have a look at the source files and the corresponding comments within those. The main goal of this chapter is to provide a high-level overview of SoRoCAD's functionality, its benefits, difficulties, and limitations.

3.1 The Platform

The Programming Language

Our software is aimed at running on macOS. While there are several programming languages available that are executable on macOS, we decided to settle with Apple's programming language, Swift. It offers native performance in contrast to languages like Python or Java. Furthermore, it has an easier to read and understand syntax in comparison to Objective-C, the other primary programming language on macOS, while offering an almost identical amount of interfaces necessary for our project and its requirements.
In the last few years, the development towards 3D graphics in mobile phones and other industry branches, as well

as the push into virtual reality and augmented reality, led to the release of a number of interfaces and frameworks focusing on 3D modeling, rendering and exporting. These interfaces will be fundamentals to achieve the necessary performance and ease of use required for our software project. The most significant framework for our project is Apple's SceneKit[1].

SceneKit

SceneKit is a high-level 3D graphics framework allowing the composing of so-called scenes. These scenes can incorporate many different elements, like cameras, lights, and 3D models, and are mostly aimed towards game development. SceneKits capabilities include the animation and interaction of named elements, and even a basic physics engine suitable for rigid body collision and particle generation. The before mentioned elements are encapsulated in objects called nodes. These nodes are essentially representing elements in three-dimensional space. The before mentioned elements are encapsulated in objects called nodes. These nodes are essentially representing elements in three-dimensional space. Depending on the node's type, they can consist of geometry, or other for the scene relevant items like lighting and physics enablers. The main benefit of this encapsulation is the unification of resources, materials, and properties of an item into a single easy to manage object. Furthermore, the underlying low-level system native implementation maximizes performance, not only on macOS but also on Apple's mobile platform iOS[2].

IDE

Because of simplicity and compatibility reasons, we choose to develop in the integrated development environment provided by Apple: Xcode[3]. It offers everything necessary for front and backend development, is free and because of its origin, deeply integrated into macOS. The development takes place on a Macbook Pro (2016) running macOS Mojave (10.14.6). Furthermore, because of the scope of this thesis, we assume a general understanding of macOS as an ecosystem and programming environment within the next chapter.

[1] developer.apple.com/scenekit
[2] apple.com/ios
[3] developer.apple.com/xcode

3.2 Requirements

SoRoCAD, as a tool, aims at providing a variety of features to ease the creation and manipulation of soft robotics parts. This feature set is rooted in requirements for frontend, as well as backend functionality. While the higher level overall requirement is a simple graphical user interface with a clear structured and easy to follow workflow, it is rooted in smaller requirements for different parts of the software.

3.2.1 Frontend

General UI Structure

The main focus of the frontend is usability. Our software should provide a familiar user interface structure with a clear line between different modes and interface elements. We orient ourselves on industry-standard 2D and 3D graphics software, namely Autodesk AutoCAD[4], Adobe Photoshop[5] and Autodesk Fusion360[6]. Within those applications, widely popular interface elements are sidebars, toolbars, and sliders with the possibility of discrete input values. Furthermore, their user interfaces are heavily centered around the content the user is working on, taking up the majority of the applications window. Adjustments to the content is displayed in real-time whenever possible, and selection, displacement, and activation of elements is touchpad optimized and mouse-based. Lastly, the usage of similar labels for widely used functions, properties, and elements are of interest for our software to assure the seamless integration into existing prototyping pipelines. While SoRoCAD will implement the input possibilities for discrete values and engineering precision, its primary focus lies in creating an interactive graphics-oriented prototyping experience using a WYSIWYG approach.

Workflow Orientation

[4] autodesk.com/products/autocad/
[5] adobe.com/products/photoshop.html
[6] autodesk.com/products/fusion-360

Definition: *WYSIWYG*

WYSIWYG:
What You See Is What You Get in computing, implies a user interface that allows the user to view something very similar to the finished result while the document is being created.

3.2.2 Backend

Based on the requirements for the frontend, the requirements for the backend can be derived. To provide WYSIWYG capabilities, the software needs to run responsively and efficiently. To achieve this, the usage of graphical frameworks provided by Apple is crucial since they have the functionality and integration to provide similar performance on any computer running macOS. Furthermore, they have the benefit of being developed and updated regularly, while being supported for a more extended period, in comparison to non-proprietary frameworks, that can easily be non-functional after small macOS updates. Because of this, we will limit the usage of third-party frameworks within SoRoCAD, only making use of such, when its unavoidable.

Third Party Frameworks

File Import, and Export

Since, we are working with potentially large 3D objects; the backend needs the functionality of not only editing 3D models, including displacement, combination, and intersection but doing so in an efficient manner. Proving support for importing and exporting standard 3D file types in minimized time is crucial. SceneKit covers the majority of the mentioned functionality.

3.3 The Blueprint

Based on the established requirements, we now proceed to create an initial blueprint, that will be our primary orientation within the design and programming phase. The blueprint consists of a workflow model and a user interface scetch.

3.3.1 The Workflow Model

Figure 3.1: A sketch of the desired workflow structure with its three distinct steps

Cell Selection

The workflow will be heavily based on small basic 3D models named cells. Those cells have a fixed initial size and are provided in a library within an overview. These cells incorporate different behavior when exposed to their activator, pressure. In a first step, the user settles with an initial cell design the construct should be based around, using information about the behavior of the cell provided within the application. He gets the possibility to adjust its dimensions by scaling the cell in all directions. After adjusting the appearance of the cell, the user proceeds to continue into a second, independent view. Within this view, he has the ability to stack the previously selected cell in all directions. Using this stacking functionality, he can generate a rough outline, of how his final part should look like. Furthermore, he has the ability to select rows, columns, and layers within the model, to modify their precise position, rotation, and scaling, thus furthermore refining the initially generated body.

Body Generation

Afterward, he proceeds into the last view. Within this view, the user can refine individual cells. By selecting isolated cells, having the ability to scale, rotate, and offset them independently, a final version of the desired model is created. This model can be refined further by being able to remove cells or replace individual cells with other cell types from the provided library, thus incorporating several cell

Body Refinement

types and their corresponding behaviors into the final design. Lastly, he can export the model itself for further refinement in other software, or automatically create a mold for 3D printing, based on the given model. The molds wall thickness and dimensions can be adjusted, as well as the exported models file type extension.

3.3.2 The User Interface

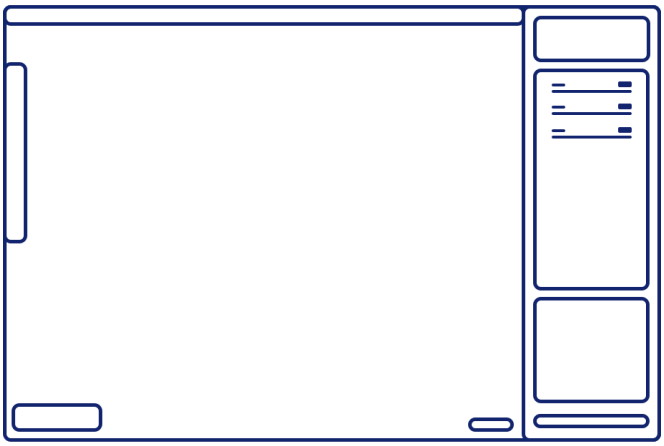

Figure 3.2: A sketch of the planned interface structure with its sidebar (right), content view (center) and toolbar (top)

Higher-level Interface Structure

The workflow model results in three separate views. The first of which handles the cell selection, library overview, and cell adjustment. The second is focused on the body generation and higher-level refinement, while the third one offers small, independent adjustments to individual cells. To keep the interface similar between each view and to work closely with the previously define requirements, we settle with a mode selector centered on top, a large content area displaying the current model, and the corresponding selection and a sidebar anchored to the right side of the applications window. The sidebar contains all necessary interface elements to adjust the parameters provided in the current model. This way, a continuous coherent experience is created despite having different modes. All interface el-

ements are anchored to the side and fixed in size, so that a window resizing leads to a larger content area, instead of linearly scaling up control elements.

3.4 User Interface Elements

Reason for Custom Interface Elements

Although Apple provides a variety of user interface element classes for use in Swift programming, we decide to create our own, to achieve an update agnostic application appearance. Another reason for our self-designed interface elements is the fact that while the provided elements are very refined, their adjustment and customization requires more work than creating new elements from scratch. Lastly, Apple provides and implements several, for our application, unnecessary functionality into each control element. This functionality makes adjustments even more complicated, especially when they are not used within our application and create unused overhead. We create four main interface elements: Boxes, Buttons, Segmented Controls, and Sliders, used consistently in our application. Their main parameters can be adjusted in Xcode.

3.4.1 Boxes

The main fundament of our user interface is boxes. Boxes represent color filled and potentially outlined squares of any size. They are implemented by subclassing the NSView[7] class.

Structure

In their initialization, they require their own layer. This separate layer provides them with the functionality of rounded corners. When the class calls its rendering function, the complete view gets filled with a specific color defined in Xcode. Furthermore, depending on its status, it activates rounded corners with a specific radius and an outline with a specific width and color. These boxes will be used to provide other interface elements with a background, as well as for structural reasons, encapsulating elements for easier

[7] developer.apple.com/documentation/appkit/nsview

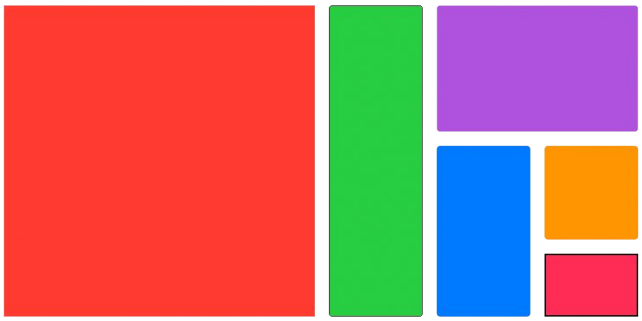

Figure 3.3: A variety of colored boxes with optional rounded corners and borders created by utilizing our custom box class

grouping and hiding. The central usage of this basic class is the sidebar's structure and its background.

3.4.2 Buttons

Interaction

Within our application, buttons are similar to boxes, with the extension of interaction with the user. A button subclasses the NSControl[8] class, which extends the functionality of a box with functions that handle mouse interaction and send information about its state to higher interface elements and application delegates. To make the design responsive, it contains values indicating its push state, a label, and for the desired activation of other functionality, functions to send notifications when it is triggered. While rendering , the button is rendered in multiple layers. The first of which is a color filed box with rounded corners. The second one contains the label, which is transformed from a simple string into NSAttributedString[9] to be drawn onto the base layer. The third contains a slightly darker and smaller box that indicates the buttons state.

Structure

[8] developer.apple.com/documentation/appkit/nscontrol
[9] developer.apple.com/documentation/foundation/nsattributedstring

Figure 3.4: A push-button created by our button class in both activation states, regular (left) and pressed (right)

Lastly, each button can be created as a classic push button, or a state button, which is either on or off. The first will be used for regular buttons, inducing exporting or import of files or triggering the removal of cells. The second type will be used for the cell library overview, which is either visible or hidden.

3.4.3 Segmented Controls

Figure 3.5: A control element based on three segments created by our segmented control class, with the first segment being selected

Structure

This type of control is based on a combination of state type buttons and a box. In its core, it extends the boxes' functionality with a mouse coordinate check, thus allowing for multiple buttons to be encapsulated within the same view.

Interaction

After a user interaction, the mouse position then determines which of the buttons gets selected. The corresponding button then adjusts by changing its color, disabling the other buttons, and sending information to the higher-level objects, inducing an action. All there buttons share the same rounded-corners box, and therefore the same background. This control will be used as a selector for the three view modes, and the three selection modes within the body generation view.

3.4.4 Sliders

Structure

Sliders are the most sophisticated interface element we utilize. They contain a label, a value box, and the slider, consisting of a background and a knob. The label is initialized and handled in the same way it is within a button and a segmented control. The value box is a label drawn onto a NSBezierPath[10]. The slider is a darker, fixed height NSBezierPath, spanning across the majority of the element's width. The knob is a small NSBezierPath that is drawn according to the sliders' current value and the views and sliders' width.

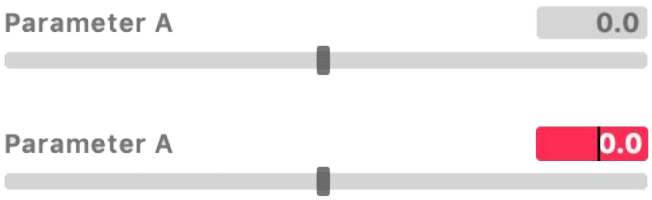

Figure 3.6: A slider based on our slider class, in its regular state (top) and editing state (bottom)

[10] developer.apple.com/documentation/appkit/nsbezierpath

Interaction

To enable user interaction, both mouse position coordinates are used. If the user is clicking within the slider, the mouse position is checked, and a new value is computed based on the maximum and minimum value, as well as the slider's width. The value box then gets updated to display the new value, and higher-level objects receive an action triggered by the change. A similar routine is executed when the mouse is dragged.
The width of the slider and the screens resolution limits the discrete values the slider can select. To overcome this issue, our slider offers the manual input of values triggered by double-clicking the value box. This is achieved, by overlaying the value box with an editable NSTextField[11]. This text field is hidden until the user triggers the input by double-clicking and hides again after a new valid value is entered and confirmed by pressing enter. Lastly, our slider class contains optimizations and functionality, like snapping to integer values or its center value.

Direct Value Input

3.5 The Backend

After creating our custom user interface elements, we proceed with the implementation of the necessary backend functionality later utilized in the user interface and interaction.

3.5.1 Handling 3D Files

As previously noted, SceneKit is the backend fundament in SoRoCAD. Its deep integration and utilization in games and applications for macOS and iOS led to the implementation of native file handling solutions. While a SCNScene[12] can be initiated empty, SceneKit also provides the ability to load a scene from a file. When loading a scene from a file, the file type extension determines the loaded information.

[11] developer.apple.com/documentation/appkit/nstextfield
[12] developer.apple.com/documentation/scenekit/scnscene

File Types

To save and restore whole scenes, including lighting, cameras, and geometry, SceneKits proprietary file extension ".scn" is used. If only geometry with its materials and textures is of interest, thus excluding lighting and cameras, the file type ".dae" is recommended.
Finally, if only the geometry is of interest, without any information on material, widely used file types like ".stl" and ".obj" are supported. Each SCNScene object has corresponding initialization and writing functions we utilize to save and restore individual nodes or entire scenes. In the initial cell library provided with our application, each cell is saved to and loaded from a ".dae" file.

3.5.2 Adjustment of Cell Parameters

SCNNode Objects

The first step in our workflow consists of the selection and adjustment of a cell. In SceneKit, this cell is represented by a SCNNode[13] object. This object , in its core, is encapsulating all relevant information a 3D model has, including its position in the SCNScene, its geometry, rotation, scaling, material, as well as its physical properties for the physics engine and its pivot. The pivot of a SCNNode determines its center, relative to which each geometric operation will be applied. This information crucial when it comes to adjusting properties like scaling and position in our application, thus we need to make sure the pivot is always a the center of each cell. Furthermore, the SCNNode class contains a variety of functions dedicated to providing different information like an object's bounding box, which we utilize to find its center for repositioning purposes.

Benefits of SCNNodes

In the different steps our software incorporates, each cell-specific slider in the sidebar is mapped to multiple or a single function. These functions then adjust the corresponding node parameters and notify the SCNView[14] to update its rendering of the node. In the first step, the cell selection, each slider is directly connected to a function adjusting the nodes scale vector, where a value larger than one makes the node larger, while a value smaller then one does the opposite.

[13] developer.apple.com/documentation/scenekit/scnnode
[14] developer.apple.com/documentation/scenekit/scnview

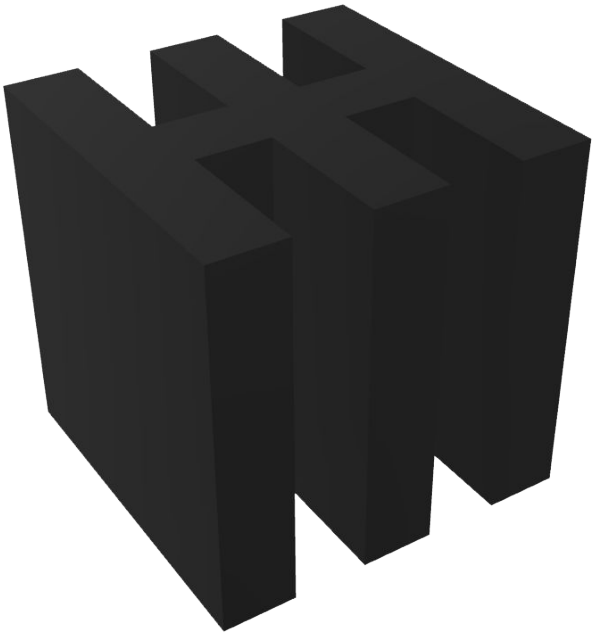

Figure 3.7: An example cell represented by an SCNNode that is included in the application's cell library

Hierarchy in SCNScenes

Lastly, each SCNNode contains the function to remove itself from its parent node and to add another node as its child node. Each SCNScene has a non-removable root node that contains every other node as a child node in an array. This hierarchy is of use when it comes to grouping and joining multiple nodes into a single one.

3.5.3 Generating the Body

In the second step of our workflow, the user proceeds into another view. Within this view, he has the ability to generate a body constructed out of the previously chosen cell. The body is generated by stacking cell objects ontop, beside and in front of another.

Cloning SCNNodes

Using this approach, an issue regarding the properties of the cell arises. The SCNNode class includes a copy and clone function. Both functions essentially clone the cell with its properties, but the copy function does not recursively include child nodes. The resulting problem is that for efficiency reasons, those functions do not clone the nodes geometry and other SceneKit objects attached to it. The clone shares these attached objects with the initial node. This is not desired in our application, since applying geometric transformations to any node, would result in changes to all nodes. To overcome this, we implement our own deep clone function, which not only creates new nodes but duplicates the attached objects and therefore separates the geometries. Using our function, we now have the ability to change the scaling, transformation, and materials of individual nodes.

Generating the Core

The stacking is implemented by first getting the bounding box of a single node. Using the bounding box position and dimension, we can calculate the position of each stacked node. Starting with the first node having its pivot positioned at the origin of the scene, we then continue to create deep copies of the previously chosen node and stacking them until the desired number of rows, columns, and layers is reached. One benefit of this approach is that later changes to the chosen cell geometry in the first step, are instantly applied to the generated body. The user, therefore, has a clear separation of steps in the workflow and dodges the problem of readjusting the body in different steps according to small changes made.

Geometric Transformations

After the initial body generation is finished, the software proceeds with applying desired geometric transformations to the nodes. SoRoCAD offers three selection modes: rows, columns, and layers. The first of which selects nodes in the direction of the y-axis, the second one in the direction of the x-axis, and the last one along the z-axis. Since we allow multiple transformations to a single node, the software needs to store all transformation information for each selection mode. For storing this information, we are using NSDictionary[15] objects. These objects, from a usage standpoint, are similar to arrays, but instead of assigning every contained object a numeral index, they make use of keys to retrieve the stored information.

[15] developer.apple.com/documentation/foundation/nsdictionary

Identifying individual Nodes

In the case of SoRoCAD, these keys are the names of the nodes. A node's name is a string value attached to a SCNNode object. In the initial body generation process, each body gets the name "x,y,z", where each variable stands for the stack count along each axis until the specific node in the generated object is reached. This naming is utilized on multiple occasions within the backend, as it provides a convenient way of identifying individual nodes.
While creating the deep copy of every node and placing it according to the desired row, column, and layer count, the software checks multiple NSDictionary objects using the nodes name. These objects are storing information on rotation, scaling, and offset for every row, column, and layer. If no adjustment information is found for a row, column, or layer, no geometric transformation takes place, and the node is skipped. If multiple transformations are found, they get combined according to their property. A node's offset and rotation is additive, while its scaling is multiplicative.

Centering the Body

Finally, after all steps are finished, the body is rendered in the corresponding SCNView. To make sure the scenes camera behaves in a desired way, a last step is necessary to center the generated body in the scene. To achieve this, we first get the scenes root node bounding box and translate every node so that the body's center point aligns with the origin of the scene.

3.5.4 Joining Nodes

While rendering the previously stacked nodes within the SCNView, a fundamental problem in 3D graphics, called Z-fighting, is created.

Definition: *Z-fighting*

Z-FIGHTING:
Z-fighting is a phenomenon in 3D rendering that occurs when two or more objects have a similar or identical distance to the viewing point. It is usually caused by limited precision and round-off errors. It results in artifacts and partly rendered faces, influenced by the viewing angle.

In our application, there is no way to overcome this issue without sacrificing the WYSIWYG aspect of it. To minimize its occurrence, we utilize the SCNNode value called ”renderingOrder”. In a simplified way, this value determines which node gets rendered first and which one gets rendered last. Regarding Z-fighting, we noticed improvements in certain situations by assigning higher values to the outer SCNNodes and lower values to the hidden ones.

Figure 3.8: Example of emerging artifacts from Z-fighting, created by overlapping nodes

Soft Proof

A primary reason this issue arises is that each node gets rendered as a separate object. A simple solution would be to combine the geometry found in the generated body into a single geometry and thus to eliminate any rendering issues. While this is an option, it is not suitable for our application, since the joining of nodes would eradicate the possibility of adjustments to individual nodes. To provide the user with an artifact-free preview of the current models state, we implement a feature named ”Soft Proof”. In its core, it alters the current model so that nodes are slightly

overlapping each other and then proceeds to join their geometries into a single one while displaying the resulting body in a separate view.
Unfortunately, this cannot be implemented using only SceneKit. Geometric operations like union and intersect are not part of SceneKits toolkit, as it is not aimed at in-depth vector-based transformations and granular alternation of SCNNode geometries. While there may be the possibility of rewriting large chunks of code to achieve the for SoRoCAD required features, it would most certainly blow the scope of this thesis.

The Euclid Library

To overcome this barrier, we utilize the under the MIT license published Euclid[16] library. Euclid is a library for creating and manipulating 3D geometry using techniques such as constructive solid geometry to combine or subtract shapes from one another. Euclid provides SoRoCAD with the necessary geometric operators while having a slim footprint and included SceneKit support. We utilize it to enable our "Soft Proof" feature and use it for the creation of molds based on the constructed models.

Definition: *Constructive Solid Geometry*

CONSTRUCTIVE SOLID GEOMETRY:
Constructive Solid Geometry is a technique used in solid modeling. It allows the creation of a complex surface or object by using Boolean operators to combine simpler objects. This leads to potentially visually complex objects resulting from the combination of primitive ones.

Combining Geometry

The "Soft Proof" feature is implemented by first creating a new SCNNode object that will be used for encapsulation. We then proceed to deep copy all nodes in the generated body and add them as a child node to our freshly created object, to avoid destroying the generated body's current state. Using Euclid , each node's geometry then gets converted into a mesh. This newly created mesh then is transformed according to the transformation of the node and stored in an array. After every node is converted, SoRoCAD continues by applying a union operation to pairs of meshes, starting with the first meshes in the array. It continues to do so until all meshes are combined. The resulting mesh now represents the combined geometry of

[16] github.com/nicklockwood/Euclid

all nodes found in the SCNScene. To display this mesh, we utilize Euclids SceneKit support. It offers a function to initialize a SCNGeometry[17] from a mesh. After creating the geometry, we can continue by creating a new SCNNode and attaching the SCNGeometry object to it. As the last step, the created node's geometry gets assigned material and can be displayed by being added as a child to a root-node of a new SCNScene.

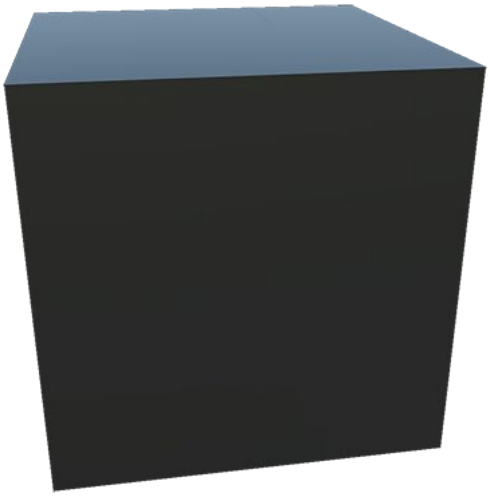

Figure 3.9: The previously presented body after having its geometry merged using the constructive solid geometry functionality of the utilized Euclid library

3.5.5 Creating a Mold

Based on the previously introduced pipeline, we can now proceed with the automated creation of a mold. In our context, a mold is the negative of the scaled bounding box containing the negative of the generated and refined body. To implement the automated generation of the mold, we use Euclid's mesh, join, and subtract functions. SoRoCAD starts off by deep copying every cell found in the fi-

Merging Meshes

[17] developer.apple.com/documentation/scenekit/scngeometry

nal model. Using the mesh conversion function, we again transfer each node's geometry into a mesh. Each mesh then gets applied to the corresponding geometric transformation to fix its position and rotation since the pure conversion of a node's geometry only places the unrotated mesh in the coordinate origin.

Retrieving the Artifical Muscle Mesh

We then proceed to create a new mesh. This mesh is initialized using a cube constructor provided in the Euclid framework. Its dimensions are defined by the bounding box dimension of the combined cells mesh or the final model and can be retrieved using either Euclid or SceneKit. In this step, we are using variable scaling factors to increase the cube's size and thus adjusting the wall thickness of the final artificial muscle. The default value used is 1.2 for all sides, leading to the walls having an additional thickness of 20% of the overall model. We then proceed to use the subtract function provided in Euclid to subtract the combined models mesh from the created scaled bounding box mesh. After this step, we end up with a mesh that looks precisely the way the final artificial muscle will look like.

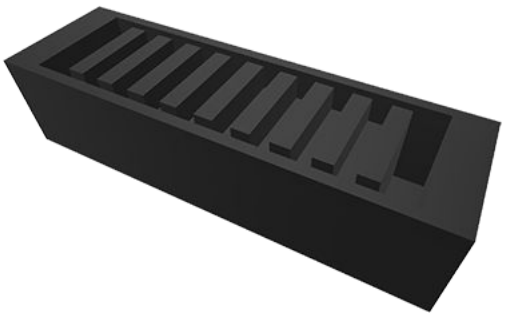

Figure 3.10: The cross-section of an example mold that is encapsuling a model created by stacking the previously referenced example cell four times

Getting the Mold's Mesh

To finally retrieve the mold instead of just a model of the artificial muscle, we repeat the process once more. We create a new cube mesh based on the now artificial muscle model. We then proceed to scale the cube up until the desired mold wall thickness is reached and then subtract the artificial muscle model. In this step, we scale the height and reposition the meshes before subtracting because the final

mold consists of two separate parts to encapsulate and provide easy access for pouring in the silicone.
The resulting mesh is converted into a SCNGeometry, and then a new SCNNode is created based on said geometry. This node is then added to the root-node of an empty SCNScene and exported using the previously mentioned methods.

3.6 The User Interface

Xcode's Interface Builder

After implementing the necessary backend functionality, we now proceed to construct the user interface using the interface builder integrated into Xcode, with the addition of our custom interface elements. Each element was programmed in a configurable way and with support for the interface builder in mind. The previously presented blueprint will be our orientation throughout this process. Before proceeding, we need to extend the SCNViews functionality and establish general rules for our interface.

3.6.1 Fundamentals and Color Palette

General Design Orientation

With the goal of our user interface to share similarities with widely used applications, we decided to stick with a simplistic and flat design approach. We are using minimal but sufficient contrasts between elements while excluding the use of computational heavy and potential distracting animations. Apple provides the majority of the colors found in our color palette as a reference point for individual user interface elements. Furthermore, we are limiting the use of different high contrast colors and settle with a color identified by "NSColor.systemPink[18]" as our central signal color. Only the visual indication of the selection of nodes, the applications icon, and the most central interface elements are sharing this color. All color values can be found within the source files.

[18] developer.apple.com/documentation/appkit/nscolor/2879261-systempink

Figure 3.11: The final application icon and used colors

3.6.2 Extending SCNViews

Limits of SCNViews

A SCNView, in its core, is the frontend interface element that allows the user to view and control SCNScenes. While it offers a large number of functions and objects to call and control, it relies heavily on extensions to make it suitable for specific applications. In our case, one missing core functionality is the selection of nodes. We are rewriting the SCNView's mouse event handling functions to handle the selection of individual and groups of nodes. We start by adding variables to save the currently selected row, column, and layer number.

Node Selection Handling

Furthermore, we attach an object reference of the type SCNNode. This object reference will later be used to reference the node the user has selected. Within the mouse event handling functions, we use a function called "hitTest[19]". This function returns every object in the way of a user's mouse interaction in the embedded SCNScene. Using the returned objects, we then check if the object is a valid and selectable part of the scene and proceed by coloring it in our signal color and referencing it using our selected node object. Using our naming scheme, we can use the node's name to determine the corresponding row, column, and layer number. Furthermore, we can send an action, notifying other parts of SoRoCAD that the selection has changed.

[19] developer.apple.com/documentation/scenekit/scnscenerenderer/1522929-hittest

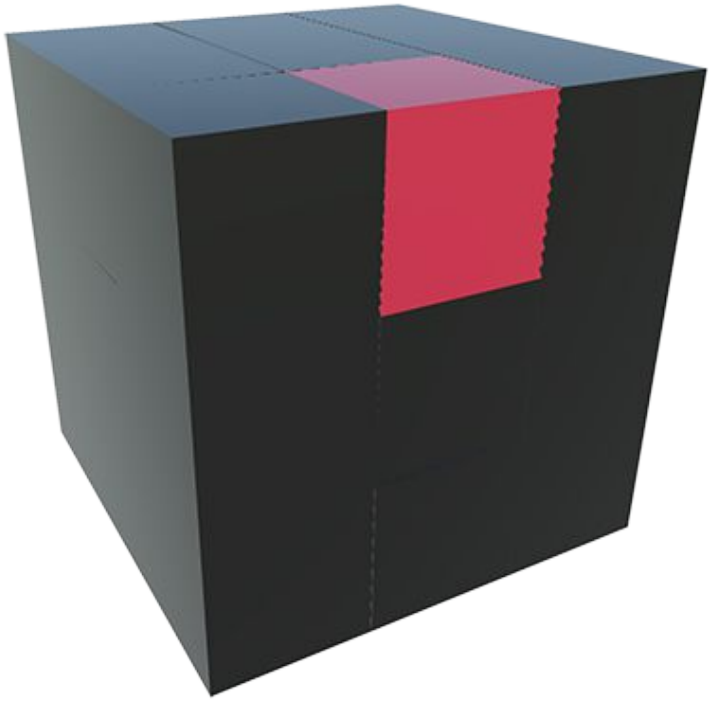

Figure 3.12: A single cell (represented by a SCNNode) selected in our custom SCNView extension highlighted using our signaling color

3.6.3 The Resulting Interface

After incorporating our design decisions and working closely with the initial blueprint, we were able to implement the majority of the planned functionality, aesthetics, and behavior. We separated the three design steps into individual views with adjusted user interface elements.

Model Orientation Indicator

For a better 3D space orientation, we decided to add a custom element to each view, indicating the camera's current position in relation to the currently displayed model. Because this decision was founded from a design perspective in a late step of development, it is added in the upper left corner of each of our extended SCNView instances. With each user interaction with the 3D model, the indicator adjusts its position accordingly.

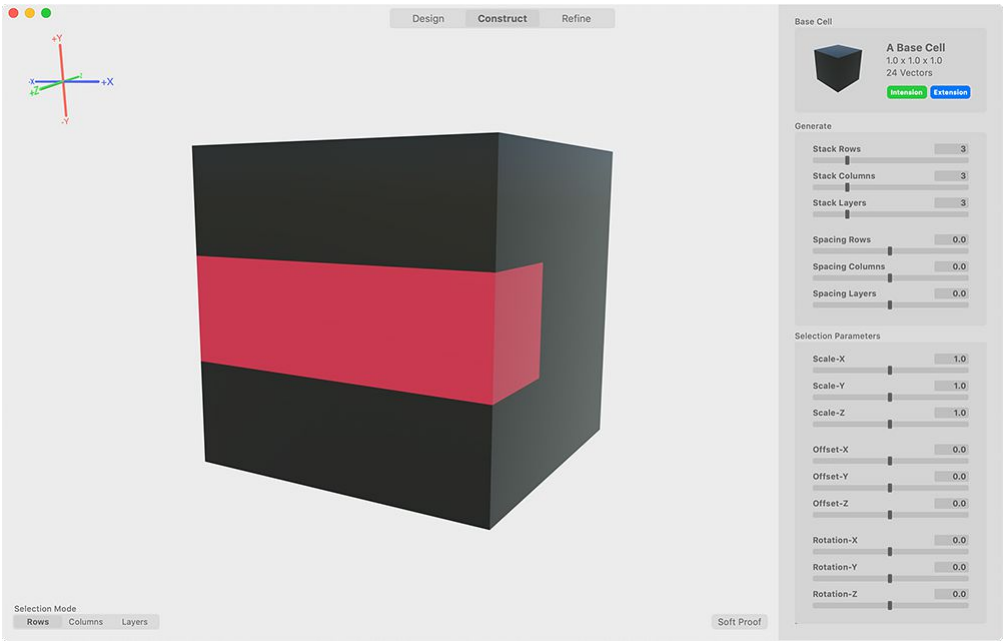

Figure 3.13: The final user interface's "Construct" view with the orientation indicator (upper left), the selection mode control (lower left) and its sidebar (right) containing sliders for row, column and layer based cell adjustments

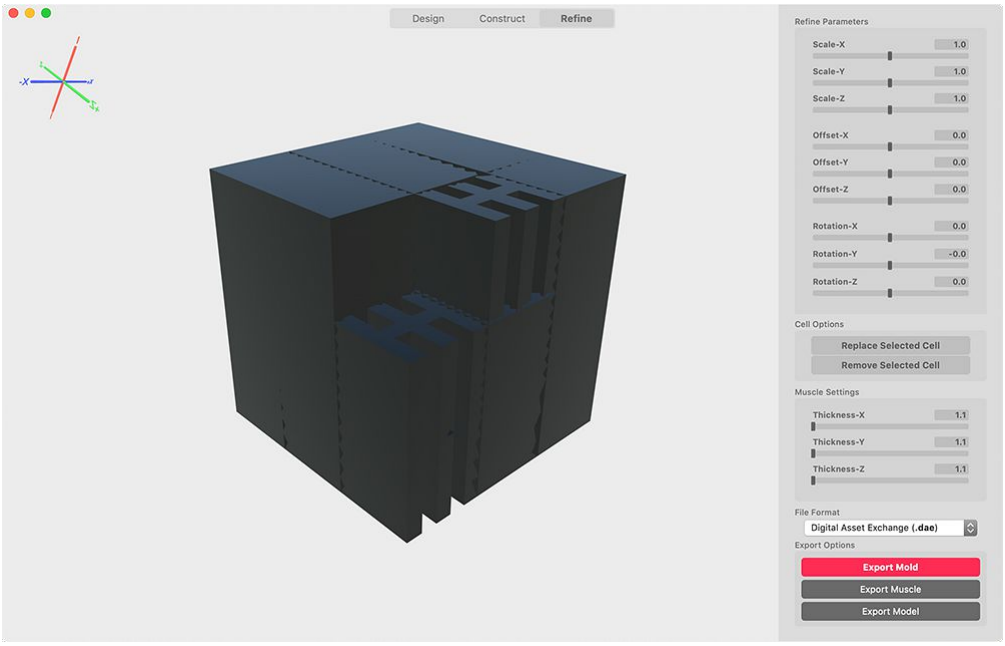

Figure 3.14: The final user interface's "Refine" view with its sidebar (right) containing sliders for individual cell adjustments, buttons for cell removal and cell type replacement as well as export options

Chapter 4

Evaluation

To analyze the intuitiveness and real-world performance of our application, we are conducting a user study. The user study's central goal is the qualitative measurement of our created workflow and user interface. Based on this goal, we are not focusing on the user's understanding of soft robotics or general 3D design, since one of the applications main initial goals was the ease of use for zero-experience users. The central part of our evaluation is the qualitative analysis because of our limited amount of participants.

4.1 Methodology and Study Procedure

We are combining a semi-structured interview with a questionnaire that users will fill out partly before and after completing two 3D modeling tasks. Both tasks are based on the recreation of provided 3D models. The first model is a simple structure to test the user's understanding and experience in 3D modeling, while the recreation of the second model aims at stressing the application's workflow and user interface by requiring a broad palette of combined functionality to be used in a targeted manner.

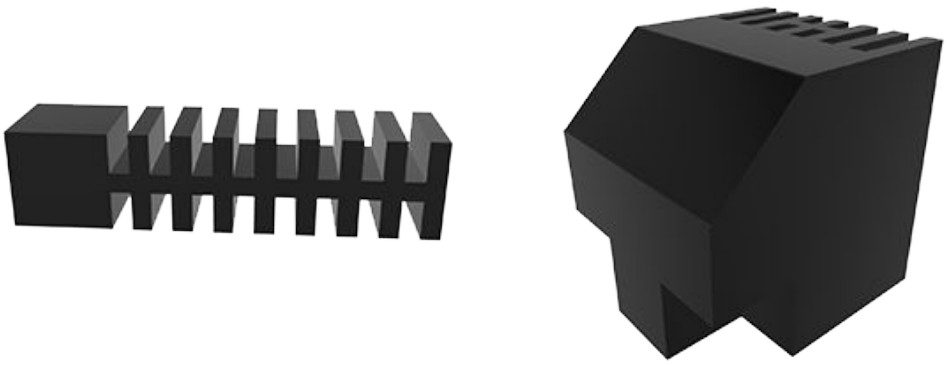

Figure 4.1: Both models used in the user study. The simpler model (left) is based on five cells stacked and displaced horizontaly to overlap with one being replaced with a box cell. The more complex model (right) consists of a 3x3x3 stack of box cells with a rotated row, replaced row as well as rotated and replaced (top right) and removed individual cells (bottom)

Methodology

On the Mac used for the study, two application windows are open at the beginning of a session. The first one is a 3D view of the current model the user needs to recreate, and the second window is our application. At every moment, while working on recreating the model, the user can analyze and view the model again. We then proceed to start a screen recording for each task. While the user works on the given model, no interaction between the investigator and the user takes place. The investigator only steps in if the user, after 2 minutes, is not able to achieve a desired geometrical operation or model outcome. Both models are recreated with a 30-second break between one another.
After both models are recreated, the user gets handed the second part of the questionnaire consisting of statements evaluating the difficulty of each task and whether or not the interface design and application functionality is clear to use and understandable. These questions are utilizing a scale from 1 to 5, where lower numbers indicate a denial of a giving statement while higher numbers indicate approval.

The questionnaire consists of the following statements in the given order:

- Task 1 was difficult
- Task 2 was difficult
- The overall task difficulty was appropriate
- The user interface structure is understandable
- The applications workflow is understandable

We are using MATLAB[1] for the quantitative analysis of our data and MAXQDA[2] for the qualitative analysis. The qualitative analysis is based on coding. The provided user feedback is repeatedly coded under generalized key-points to then provide a higher level as well as a detailed view of the collected suggestions.

Procedure

Before the users take on the tasks, we collect information on their age, gender, eye-health, experience with CAD software, and whether or not they are currently using the macOS digital ecosystem, to get an overview over external factors influencing our data. Afterward, the user gets a 1-2 minute introduction to the general idea behind SoRoCAD and the different steps incorporated in our designated workflow. While the user does get an introduction, nothing is explained to him in detail. The interface elements functionality, the workflow, and the naming of individuals steps and elements are not explained.

User Feedback Discussion

After the user evaluates the application using the scale and the statements, he has the ability to provide additional feedback regarding the user interface and design and the general functionality of SoRoCAD. The provided feedback is then discussed to refine critical points and clear possible misunderstandings. The discussion is recorded. All sessions are held in an isolated, distraction-free, and quiet place.

[1] mathworks.com/products/matlab.html
[2] maxqda.de

4.2 Quantitative Analysis

4.2.1 Participant Information

Our study participants are aged between 22 and 29, averaging around 24.5, with a median of 24. The majority of our participants are male, with a share of 75%. The average female participants' age is 23, while the males average at 25.

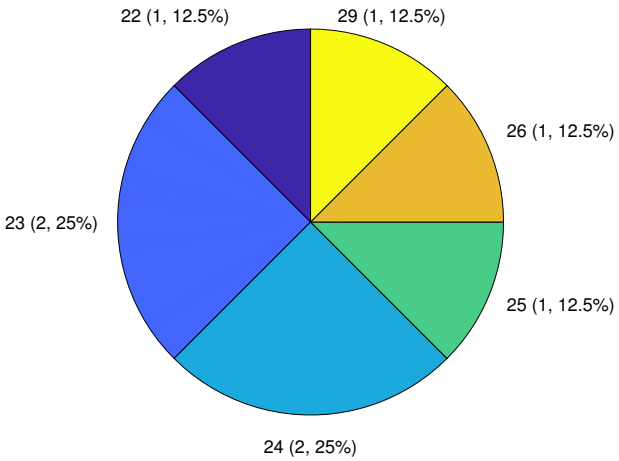

Figure 4.2: Our participant's age distribution

Considering eye-health, 75% of our participants had no visual impairment, and 25% wore glasses or contact lenses. A higher experience with 3D design and CAD, in general, was given in 37.5% of our participants, with 50% of all participants using a macOS computer in their everyday life. The majority (66%) of our participants with a visual impairment were using Macs in their everyday life.

4.2.2 Task Evaluation

Task 1

The completion of task one, remodeling of the simpler model, took 3.2 minutes on average with a median of 3.4 minutes, a variance of 1.54 minutes, and a 95% confidence interval of [2.17,4.24] for μ and [0.82,2.53] for σ under an assumed normal distribution. Participants who said they were experienced with CAD and 3D design took on average 3.52 minutes with a median of 3.5 minutes, while users with no experience took 3.01 minutes on average with a median of 2.35 minutes, thus being 14.5% faster on average.

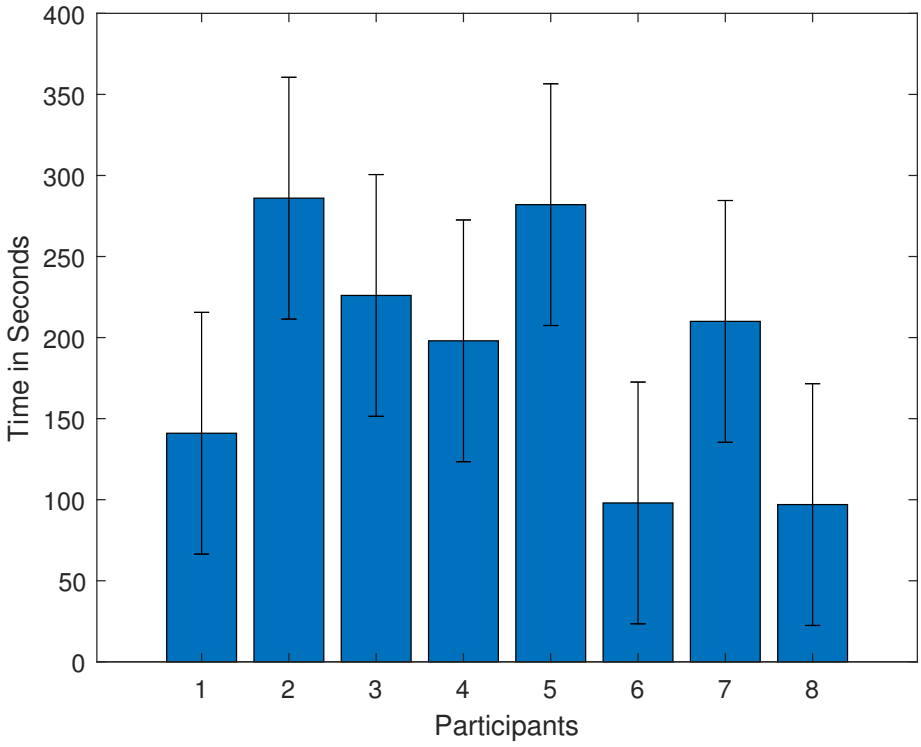

Figure 4.3: Overview over the time it took each user to complete task 1. The values range from 97 seconds to 286 seconds with a standard deviation of 75 seconds

Task 2

The completion of task two, remodeling of the complex structure consisting of different cell types and requiring a broader spectrum of functionality, took 7.98 minutes on average with a median of 5.9 minutes, a variance of 16.89 minutes and a 95% confidence interval of [4.55,11.42] for μ and [2.72,8.36] for σ under an assumed normal distribution. Participants who said they were experienced with CAD and 3D design took on average 11.39 minutes with a median of 13.02 minutes, while users with no experience took 5.93 minutes on average with a median of 5.48 minutes, thus being approximately 48% faster on average.

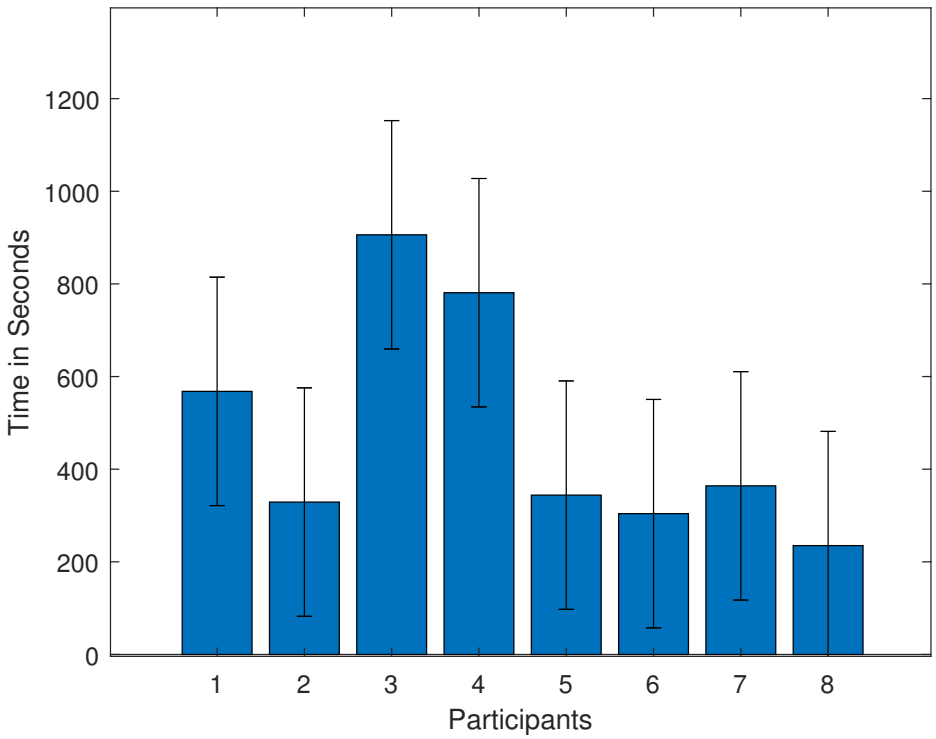

Figure 4.4: Overview over the time it took each user to complete task 2. The values range from 235 seconds to 906 seconds with a standard deviation of 247 seconds

4.2.3 Questionnaire Results

Statement 1: "Task 1 was difficult"

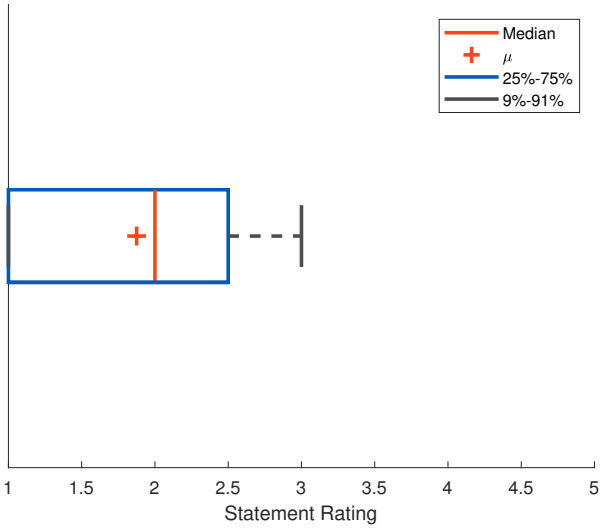

Figure 4.5: The box plot for the statement "Task 1 was difficult". All users rated this task 3 from 5 or lower, indicating a low perceived difficulty with the mean (here μ) and median hovering around 2 points

The overall perception of the difficulty of task one was low. Our goal was to create an initial task that does not overwhelm the participants and only provide an entry into the application's use case. Given the provided feedback and metrics, this goal was met. Although the differences in completion time between different subgroups of our participants were given, no correlation could be found on the perception of difficulty.

Statement 2: "Task 2 was difficult"

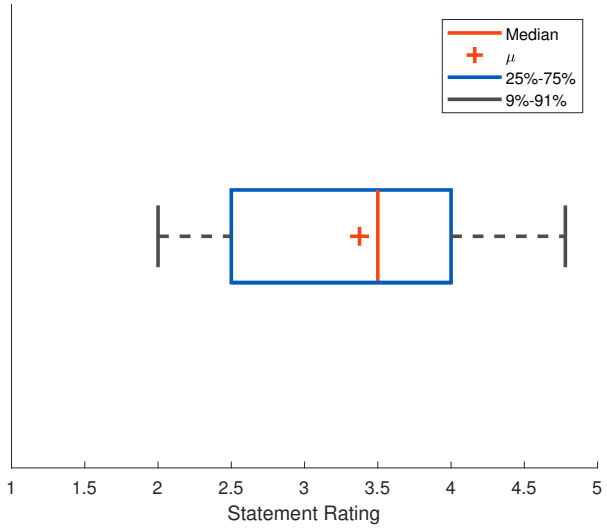

Figure 4.6: The box plot for the statement "Task 2 was difficult". The median and mean (μ) hover around 3.5 indicating a higher perceived difficulty overall, with higher variance in individual ratings ranging from 2 to 5 points

The results from our task 2 data show a similar pattern. The second task was perceived as more demanding, on average, with a broader range of perceived difficulty. Although the highest rating on perceived difficulty was by a non-experienced user, the participants who stated that they are experienced, perceived this task as more difficult on average, in contrast to non-experienced participants.
Furthermore, no user subgroup (based on gender, experience, eye-health, or Mac usage) percieved the overall task difficulty extremely different than the other groups. Lastly, the majority of users rated the application as clearly structured, while the overall workflow had a higher variance in ratings. Especially the experienced users rated low on the overall workflow, stating in the later interviews that the workflow was too different in comparison to what they are used to.

Statement 3: "The overall task difficulty was appropriate"

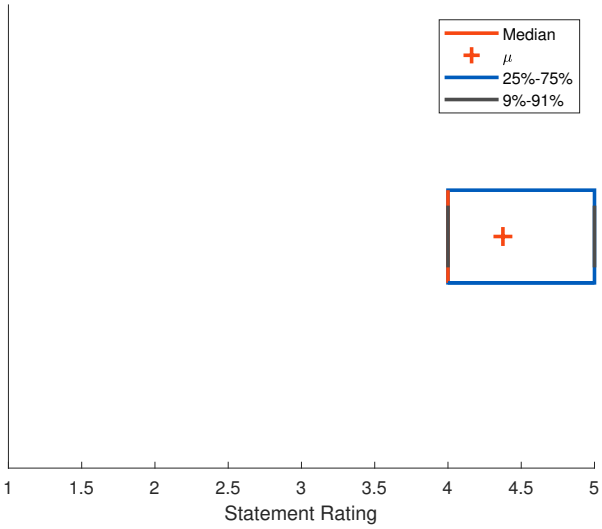

Figure 4.7: The box plot for the statement "The overall task difficulty was appropriate". The mean (μ) hovers around 4.5, and the median is at 4. The values range from 4 to 5 indicating a low variance and supporting the positive reception by the participants

The overall difficulty of the tasks in our user study was perceived as appropriate throughout all participants. In our interviews, all participants stated that they were not overwhelmed at any moment in the study, and not a single one had the perception that the outcome of the task is unachievable using our application.

Statements 4 and 5

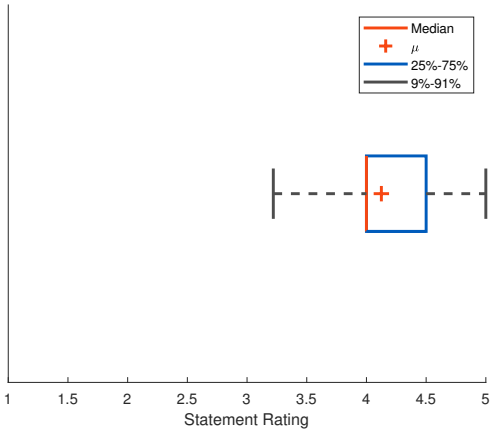

Figure 4.8: The box plot for the statement "The user interface structure is understandable". The mean (μ) hovers around 4.2, and the median is at 4. The values range from 3 to 5 indicating a higher variance and underline the more negative reception by CAD experienced participants

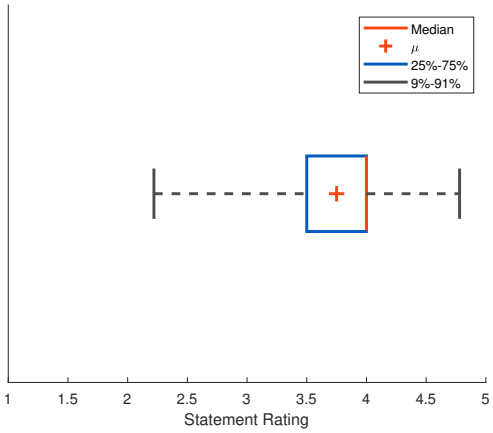

Figure 4.9: The box plot for the statement "The user interface structure is understandable". The mean (μ) hovers around 4.2, and the median is at 4. The values range from 3 to 5 indicating a higher variance and underline the more negative reception by CAD experienced participants

4.3 Qualitative Analysis

We proceed with the evaluation of recommendations for SoRoCAD. We are using coding to group recommendations and critique of a similar type into a cluster to extrude the relevant findings. As a result, using MAXQDA, the inputs were clustered under the keywords ”Functionality”, ”Workflow” and ”Design”. The majority of suggestions were based around the user interface and the design of the application.

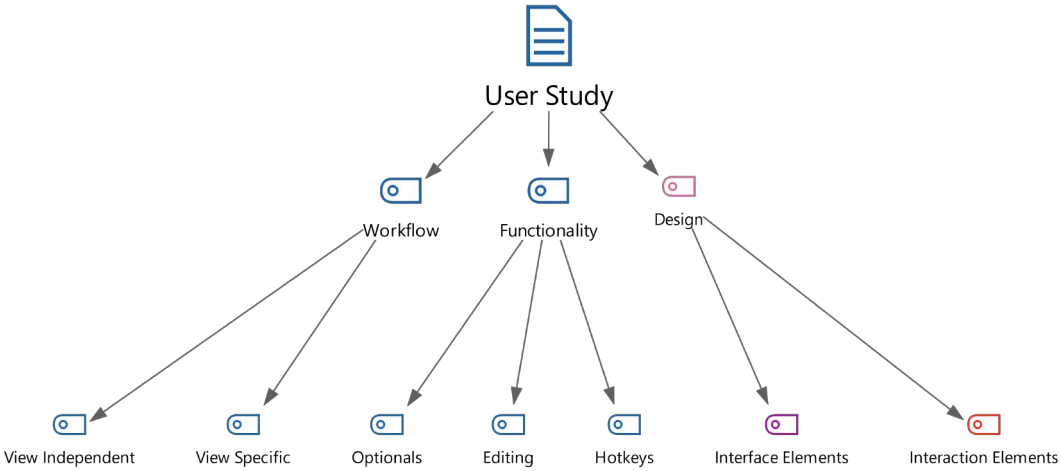

Figure 4.10: The resulting main codes ”Workflow”, ”Functionality” and ”Design”and their sub-codes indicated by arrows

4.3.1 Workflow

Under ”View Independent”, the central critique was the absence of help elements. In the later interviews, 6 out of the 8 participants stated that it was not clear that switching between the steps was desirable, and part of the workflow. Furthermore, the differentiation between the ”Construct” and ”Refine” step was rated as vague by one user. Especially the three CAD experienced power users stated that they would favor a clustered, more complicated interface with a single view and without separated steps.

The only view specific critique stated by three participants was the lack of change saving within the ”Refine” steps. When users decided to go back into the ”Construct” view, their cell replacements and deletion were not applied back again.

4.3.2 Functionality

This code is divided into three sub-codes. The majority of critique is encapsulated in the ”Editing” sub-code. One function that was missed by 7 out of 8 participants was an ”undo” function. The users repeatedly tried to use keyboard input to undo the last applied transformation and were searching for interface elements to trigger such function. Building on this missing function, users expected to have the majority of operations mapped to individual keys and key-combinations. Another major point was the saving of intermediate changes to individual cells and entire rows. All users except participants 1 and 8 expected a more consistent behavior regarding this functionality between the editing in the second and third steps. Lastly, optional settings, like the usage of degrees instead of radians, for the adjustments of a variety of the application's elements, was desired by 5 out of 8 participants.

4.3.3 Design

Within the design code, we grouped and sorted every input regarding the application's user interface that was not primarily based on deep backend functionality. The coding resulted in two major sub-codes, the first of which summarizes the critique of elements that the users do not directly interact with, while the second one includes interactive elements like sliders, buttons, and the 3D viewer.
In our study and the later interviews, 7 out of 8 users stated that the struggle with orientation in 3D space within the application. While the effect of an application of geometrical operations is apparent, the users critiqued the lack of clear visualization of rotation within the model viewer and

wished for a visualization of a coordinate grid. They criticized the existing rotation indicator as being too small and confusing. Another reoccurring point (mentioned by users 2, 3, 5, 6, and 8) of discussion around the 3D editing was the lack of a wireframe like viewing mode to visualize overlapping cells and their interaction with one another better. As an alternative to a wireframe view, three users suggested a color coding of cells with intersecting parts having a bright contrast color or a different transparency level. Lastly, one user percieved the applications signal color as too negative, giving him the impression of behaving wrong within the interface's boundaries, suggesting to replace it with a more neutral color.
The naming scheme of rows, columns, and layers was confusing for the experienced three participants. The resulting inconsistency because coordinates, as well as rows, columns, and layers, were used for labeling, amplified misunderstandings in geometrical orientation.
Sliders were the interface elements that received the most substantial amount of critique. Six participants missed tick marks indicating specific commonly used values for a given interval as well as a magnetize feature for a more accessible selection of specific values of interest. Furthermore, one reoccurring suggestion (users 2,3,6,7,8) was the usage of different units and limits for the individual slider's functions. Four users felt limited by the range of possible values and confused with the unity choice on multiple occasions.

4.4 Summarizing the Findings

We draw the following conclusions based on our user study. Resulting from our quantitative analysis in combination with the discussions, the application's intended use case was mostly met. All participants were able to recreate the provided models using SoRoCAD. Additionally, , all five zero-experience users performed as well or even better than the three experienced users within both tasks.
While two of the participants asked for a more streamlined workflow with a harder separation of steps and viewing modes, the majority of participants stated that they pre-

Central Goal

fer the minimalistic clean interface over a more sophisticated 3D design suite when it comes to creating simple to medium complex models. Furthermore, , the majority of critique towards the user interface is based around sliders and visual indicators for orientation, both of which can be adjusted, reimplemented and fixed without a large amount of effort because of our modular and adjustment oriented implementation. Smaller critique points like inconsistent labeling and confusing interaction behavior require even less effort to overcome.

UI Critique

Backend Critique

The functions that are deeply rooted in the backend are a more significant point of concern. One of which is the implementation of a more convenient change saving functionality to assure that the changes made to a particular model in a given step are saved even after switching modes and interacting with interface elements that are oriented towards the start of the design process.

Chapter 5

Summary and future work

Based on our findings from the evaluation and our efforts and thoughts on the future development of SoRoCAD, we can now summarize the overall project's process and outcome and will try to provide an insight into what is possible and should be aimed at in future contributions.

5.1 Summary and contributions

With SoRoCAD, we created a first base of what soft robotics oriented 3D design software could look like. We were able to implement the majority of intended features in relation to design and function. The SceneKit framework provided by Apple gave us the necessary core features to build a mostly robust application that was able to sustain user testing successfully. The intended workflow was mostly perceived as useful and as accelerating the understanding of 3D modeling. Furthermore, the implemented functionality is already sufficient enough for experimenting with adoption into the under "Application Areas" mentioned projects.
Unfortunately, we were not able to implement all the functionality we intended. One primary functionality that is

missing is the simulation of models within the software itself. While we provide the export functionality and interface for future integration, the current limitations given by SceneKit made it impossible for us to include soft body simulation in the given time. Furthermore, as a conclusion drawn from the user study, a variety of adjustments and additional features on an interaction level is necessary to make SoRoCAD a viable alternative in 3D design.

5.2 Future work

Based on the previously mentioned necessary adjustments, and the initially introduced related work, we can derive three major points of future work around SoRoCAD can be based on.
The first necessary development is the integration of said soft body simulation functionality. There are several simulation frameworks available written in a variety of programming languages. The Swift programming language allows for the integration of C and C++ code through type conversion headers. These could be used to overcome SceneKits limitations and, based on external libraries, incorporate a single cell as well as whole-body simulations.
Another entry point is the numerous adjustments to interface elements that are necessary. The confusion and misunderstandings towards interaction elements, the used signal color and the representation of 3D space in our user study, have shown that SoRoCAD requires further user interface refinement. This could be embedded in studies on the perception of 3D space in a computer environment, to extract a more refined and natural way of user elements for 3D interaction.
Finally, the cell library currently consists of a limited amount of cell designs. Since the central design part in SoRoCAD is based on these cells, a comprehensive extension of the said library, including a complete tagging system describing the behavior of each cell would be beneficial.

Appendix A

Source Code and Resources

Xcode Project Files[1]
Default Cell Library[2]
Application Icon[3]

[1]http://hci.rwth-aachen.de/public/users/krasnoshchokov/SoRoCAD_SourceCode.zip
[2]http://hci.rwth-aachen.de/public/users/krasnoshchokov/SoRoCAD_Library.zip
[3]http://hci.rwth-aachen.de/public/users/krasnoshchokov/SoRoCAD_AppIcon.zip

Bibliography

[1] Saurabh Vaidya, Prashant Ambad, and Santosh Bhosle. Industry 4.0 – a glimpse. *Procedia Manufacturing*, 20:233 – 238, 2018. 2nd International Conference on Materials, Manufacturing and Design Engineering (iCMMD2017), 11-12 December 2017, MIT Aurangabad, Maharashtra, INDIA.

[2] Sangbae Kim, Cecilia Laschi, and Barry Trimmer. Soft robotics: a bioinspired evolution in robotics. *Trends in Biotechnology*, 31(5):287 – 294, 2013.

[3] M. Manti, V. Cacucciolo, and M. Cianchetti. Stiffening in soft robotics: A review of the state of the art. *IEEE Robotics Automation Magazine*, 23(3):93–106, Sep. 2016.

[4] E. Acome, S. K. Mitchell, T. G. Morrissey, M. B. Emmett, C. Benjamin, M. King, M. Radakovitz, and C. Keplinger. Hydraulically amplified self-healing electrostatic actuators with muscle-like performance. *Science*, 359(6371):61–65, 2018.

[5] C. Liu, H. Qin, and P. T. Mather. Review of progress in shape-memory polymers. *J. Mater. Chem.*, 17:1543–1558, 2007.

[6] Daniela Rus and Michael T Tolley. Design, fabrication and control of soft robots. *Nature*, 521(7553):467, 2015.

[7] Frederick Largilliere, Valerian Verona, Eulalie Coevoet, Mario Sanz-Lopez, Jeremie Dequidt, and Christian Duriez. Real-time control of soft-robots using asynchronous finite element modeling. *ICRA*, 2015.

[8] Christian Duriez. Control of elastic soft robots based on real-time finite element method. *ICRA*, 2013.

[9] Jonathan Hiller and Hod Lipson. Dynamic simulation of soft multimaterial 3d-printed objects. *Soft Robotics*, 1(1):88–101, 2014.

[10] François Faure, Christian Duriez, Hervé Delingette, Jérémie Allard, Benjamin Gilles, Stéphanie Marchesseau, Hugo Talbot, Hadrien Courtecuisse, Guillaume Bousquet, Igor Peterlik, and Stéphane Cotin. *SOFA: A Multi-Model Framework for Interactive Physical Simulation*, pages 283–321. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[11] Jinglu Zhang, Yao Lyu, Yukun Wang, Yinyu Nie, Xiaosong Yang, Jianjun Zhang, and Jian Chang. Development of laparoscopic cholecystectomy simulator based on unity game engine. In *Proceedings of the 15th ACM SIGGRAPH European Conference on Visual Media Production*, CVMP '18, pages 4:1–4:9, New York, NY, USA, 2018. ACM.

[12] Shan-Yuan Teng, Tzu-Sheng Kuo, Chi Wang, Chi-huan Chiang, Da-Yuan Huang, Liwei Chan, and Bing-Yu Chen. Pupop: Pop-up prop on palm for virtual reality. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, UIST '18, pages 5–17, New York, NY, USA, 2018. ACM.

[13] Koya Narumi, Fang Qin, Siyuan Liu, Huai-Yu Cheng, Jianzhe Gu, Yoshihiro Kawahara, Mohammad Islam, and Lining Yao. Self-healing ui: mechanically and electrically self-healing materials for sensing and actuation interfaces. In *Proceedings of the 32Nd Annual ACM Symposium on User Interface Software and Technology*, UIST '19, pages 293–306, New York, NY, USA, 2019. ACM.

Index

Typeset October 21, 2019