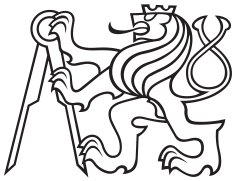
Bachelor's Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering

Department of Computer Graphics and Interaction

2D painting in VR

Robert Papay

Supervisor: Ing. David Sedláček, Ph.D.
Field of study: Open informatics
Subfield: Computer Games and Graphics
May 2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Papay** Jméno: **Robert** Osobní číslo: **492304**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Specializace: **Počítačové hry a grafika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

2D kreslení na papír ve virtuální realitě

Název bakalářské práce anglicky:

2D painting in VR

Pokyny pro vypracování:

Prostudujte možnosti realizace kreslení na papír ve virtuální realitě, a seznamte se Langweilovým modelem Prahy a postupy, jak byl vytvořen [7].
Navrhněte interakční techniky připomínající tvorbu domů Langweilova modelu (kreslení na papír tužkou, perem, štětcem s barevnou tuší, ořezání papíru a lepení částí modelu). Techniky navrhněte jak volné (free hand), tak omezené pomocí vodítek (guide lines, constraints). Uživatel může kreslit na zvětšený papír oproti reálu nebo v měřítku. Předpokládejte, že uživatel bude sedět u reálného stolu, který mu bude vytvářet pasivní haptickou zpětnou vazbu ve virtuálním světě (stůl ve VR uvidím na stejném místě).
Navržené techniky implementujte v herním enginu Unity a otestujte s vybranou skupinou uživatelů (po dohodě s vedoucím páce).
Implementované techniky zakomponujte do již existujícího zážitku nazvaného 'Tvorba Langweilova modelu Prahy' [5] a přizpůsobte je vzhledem tomuto virtuálnímu prostředí. Následně změřte změnu výkonu VR scény (např. pokles snímkové frekvence, zpoždění a prodlevu systému, paměťové nároky) a případně realizujte kroky pro dosažení plynulé (realtime) odezvy.

Seznam doporučené literatury:

1] Jason Jerald. 2015. The VR Book: Human-Centered Design for Virtual Reality. Association for Computing Machinery and Morgan & Claypool, New York, NY, USA.
2] Joseph J. LaViola, Jr. et all. 3D User Interfaces: Theory and Practice, second edition. 2017. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.
3] Steven M. LaValle - Virtual Reality, Cambridge University Press 2016
4] J. Žára, B. Beneš, J. Sochor, P. Felkel, Moderní počítačová grafika, 2. vydání. Computer Press, 2004.
5] Lukáš Pospíšil, Virtuální zážitek „Tvorba Langweilova modelu Prahy“. BP ČVUT FEL 2021.
6] Filip Suchý, Sledování reálných objektů ve virtuální realitě. DP ČVUT FEL 2021.
7] Kateřina Bečková, Miroslav Fokt. Svědectví Langweilova modelu Prahy. Schola Ludus Pragensia 1996.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. David Sedláček, Ph.D. katedra počítačové grafiky a interakce FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **08.02.2022** Termín odevzdání bakalářské práce: **20.05.2022**

Platnost zadání bakalářské práce: **30.09.2023**

Ing. David Sedláček, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Assignment instructions translation

Study the possible realisations of drawing on paper in virtual reality, and get acquainted with Langweil's model of Prague and the methods with which it was created.
Propose interaction techniques resembling the creation of Langweil's model houses (drawing on a paper with a pencil, pen, brush with coloured ink, paper cutting and gluing part of the model). Design both free hand and constrained techniques. The user may draw on a paper enlarged or scaled compared to a real paper. Assume that the user will sit at a real table, which will create passive haptic feedback in the virtual world (the table will be seen at the same position).
Implement the proposed techniques in the Unity game engine and conduct a test with a selected group of users (after an arrangement with the supervisor).
Merge the implemented techniques with an already existing experience called 'Langweil Model of Prague Creation' and adjust them accordingly to the experience's virtual environment. Then measure any changes in performance of the VR scene (e.g. decrease in frames per second, increased delay of the system, memory requirements) and exercise steps to reach smooth (realtime) latency if necessary.

Acknowledgements

I would like to primarily thank Ing. David Sedláček, Ph.D. for his guidance and advice, nonetheless I would also like to express my deepest gratitude towards my family and friends for their invaluable support given to me throughout the entire year of making this thesis.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague, 20. May 2022

Abstract

This thesis focuses on analysing digital painting methods and their possible implementations in terms of virtual reality with the goal of creating a virtual realistic experience reminiscent of creating Langweil's model of Prague. It provides an overview of digital painting, analysis on creating a realistic virtual experience and suggests how painting, cutting and other features could be implemented in virtual reality while keeping the experience immersive. It also provides an overview and test results of an implementation in Unity. This thesis builds on the work of Bc. Lukáš Pospíšil called *VR experience: „Langweil Model of Prague Creation“*.

Keywords: virtual reality, digital painting, Unity, RenderTexture, DCEL

Supervisor: Ing. David Sedláček, Ph.D. Katedra počítačové grafiky a interakce, FEL

Abstrakt

Tato práce se zabývá analýzou a možným implementováním metod digitálního malování v rámci virtuální reality za účelem vytvořit virtuálně realistický zážitek připomínající tvorbu domů Langweilova modelu. Je poskytnut přehled digitálního malování, analýza tvorby realistického virtuálního zážitku a je navržena možná implementace malování, řezání a dalších funkcionalit ve virtuální realitě se snahou udržet zážitek uvěřitelným. Je také poskytnut přehled a výsledky testování implementace v Unity. Tato práce navazuje na práci Bc. Lukáše Pospíšila s názvem *Virtuální zážitek „Tvorba Langweilova modelu Prahy“*.

Klíčová slova: virtuální realita, digitální malování, Unity, RenderTexture, DCEL

Překlad názvu: 2D kreslení na papír ve virtuální realitě

Contents

Figures

Tables

Chapter 1

Introduction

Virtual Reality, or VR in short, is a modern technology that allows users to experience virtual worlds that would prove to be difficult or even impossible to recreate in real life. Recently, museums around the world have started using virtual and augmented reality to enhance existing and create new exhibits in hopes of making them more attractive to visitors. [Stu]

Langweil's model of Prague is one of the most attractive exhibits of the Prague City Museum. [Mus] Quote: *Langweil's model is exclusive evidence of the appearance of Prague's Old Town, Lesser Town, and Prague Castle before the redevelopment of Prague in the late 19th and early 20th century. . . . More than two thousand buildings are depicted in the scale 1:480 . . . The model was created by the employee of the Prague University Library Antonín Langweil (1791–1837) who dedicated all his free time and financial means to his uncommon avocation.*

I am fortunate enough to be part of a project directed by the Prague City Museum, which aims to create an immersive VR experience that would allow museum visitors to sit down at the same table as Langweil once sat at while drawing his masterpiece and try it themselves. This thesis focuses on describing a possible implementation of such an experience and combining it with an environment created by Bc. Lukáš Pospíšil in his bachelor's thesis. [Pos21]

Chapter 2
Analysis

In this chapter, we analyse how digital painting is implemented in non-VR applications and discuss how we could translate their techniques and features into VR. Next, we analyse how to balance realism and usability to keep the experience both immersive and enjoyable. Finally, I present a list of used technologies and functional and nonfunctional requirements of the required implementation.

2.1 Digital painting

Digital painting is a form of art where artists paint using electronic devices instead of traditional tools. In its simplest form, digital art differs from traditional art in many ways, as the works are created, stored and displayed digitally [ŽBS05, Section 2.7], but modern hardware and software allow this gap to be minimal. One of the main advantages over traditional painting is the ability to use tools that would be impossible to use in the real world, such as the ability to undo actions, paint in layers, etc., although availability of these tools depend heavily on the software used (i.e. Microsoft Paint vs. Photoshop).

2.1.1 2D applications

2D applications use a simple concept where a display serves as the canvas and the user uses a tool to paint on it. Artists tend to use a combination of a special pen and a touch-screen (most commonly a specialised tablet, see figure 2.1), but anything that is able to move the cursor can be used. The application then uses the selected brush shape, colour, and other settings to draw where the cursor is pointing to. The canvas might consist of several layers, each layer being independent of each other besides their order.

Figure 2.1: David Revoy using Wacom Cintiq 13HD to draw[1]

[1]Source: https://www.davidrevoy.com/article334/making-of-episode-23

The user then edits only the selected layers, and when the work is exported as an image, the layers are combined into one based on their order. Other features might be implemented, but the ones mentioned concern us the most.

There are many applications made for digital painting using a 2D interface. This includes Adobe products[2], Procreate[3], Clip Studio Paint[4], Krita[5] and many more.

2.1.2 3D applications

3D applications use rendering to display 3D scenes on a display [ŽBS05, Section C], while still making use of previously mentioned tools, techniques and features to paint. They are usually closely related to 3D modelling and can be summed up to 2 categories, based on how they use the 3D space:

- Applications that paint directly into the scene, creating **3D drawings** by building 3D geometry. They are fairly scarce, as it is difficult to control a brush in 3D space using a 2D controller. They are mainly designed for architects. A good example that I could find is Mental Canvas[6].

- Applications that paint onto a rendered 3D object, creating a **2D image** called a *texture* that is then mapped onto the object, allowing the use of forward and inverse mappings [ŽBS05, Chapter 13] to display it. To edit the texture, they send virtual ray (aka *ray cast*) "through the screen" into the scene, take the point on the object where the ray hits and use inverse mapping to write into the corresponding texture elements (aka *texels*). These applications are mainly designed to be part of a 3D modelling pipeline, allowing for an alternate method to create textures for models, although painting in 3D games also mostly falls under this category. Examples include the popular free modelling software Blender[7], the paid but very powerful Substance Painter[8] and a small indie game Passpartout[9].

[2] https://www.adobe.com/creativecloud/illustration.html
[3] https://procreate.art/
[4] https://www.clipstudio.net/en/
[5] https://krita.org/en/
[6] https://mentalcanvas.com/
[7] https://www.blender.org/
[8] https://www.substance3d.com/
[9] https://store.steampowered.com/app/582550/

Figure 2.2: Screenshot from the VR oil-painting simulator Vermillion VR[10]

2.1.3 VR applications

As is the case with 3D applications, VR applications adopt many of the previously mentioned techniques, but the most noticeable difference is the use of physically tracked controllers with 6 degrees of freedom to realise the drawing process. The user wears a headset through which they look at the scene and a set of controllers (some applications may even work without controllers, tracking hands directly instead) to move their virtual hands around. The scene may be static or allow the user to move in the virtual world without moving in the physical world. A common problem is the lack of feedback when touching virtual objects that are not present in the physical world. Analogously to 3D applications, most applications can be summed up to 2 categories:

- Applications that paint directly into the scene, creating **3D drawings** by building 3D geometry. Opposite to 3D applications, VR applications are much more suitable for this category due to their use of VR controllers. As the user usually paints into free space, haptic feedback is not an issue. Examples include Google's Tilt Brush[11], CoolPaintrVR[12] for PlaystationVR, Gravity Sketch[13], SteamVR Home, the main hub for SteamVR[14] and even A-Painter[15], an in-browser VR painting applet.

[10] Source: https://vermillion-vr.com/press-kit/
[11] https://www.tiltbrush.com/
[12] https://coolpaintrvr.com/
[13] https://www.gravitysketch.com/
[14] https://store.steampowered.com/app/250820/
[15] https://aframe.io/a-painter/

Figure 2.3: All the implemented tools, from left to right: pen, big brush, small brush, knife, moving tool, eraser.

- Applications that paint onto a rendered 3D object, creating a **2D image** called a *texture* that is then displayed on the object (see 3D applications for a more detailed description). Applications in this category suffer heavily from the lack of proper haptic feedback, as the virtual surface the user is supposed to be painting on often does not correspond to a physical one. The same applies to the painting tool, since VR controllers are not designed to function as one. Although difficult, overcoming these complications is not impossible through clever engineering. Examples for this category are Vermillion VR[16] (see figure 2.2), a VR oil painting simulator, a VR painting sandbox PaintingVR[17] and a small, yet very well made part of Half-Life: Alyx[18].

  This is the category under which the final application falls.

[16] https://vermillion-vr.com/
[17] https://store.steampowered.com/app/1905940/
[18] https://store.steampowered.com/app/546560/

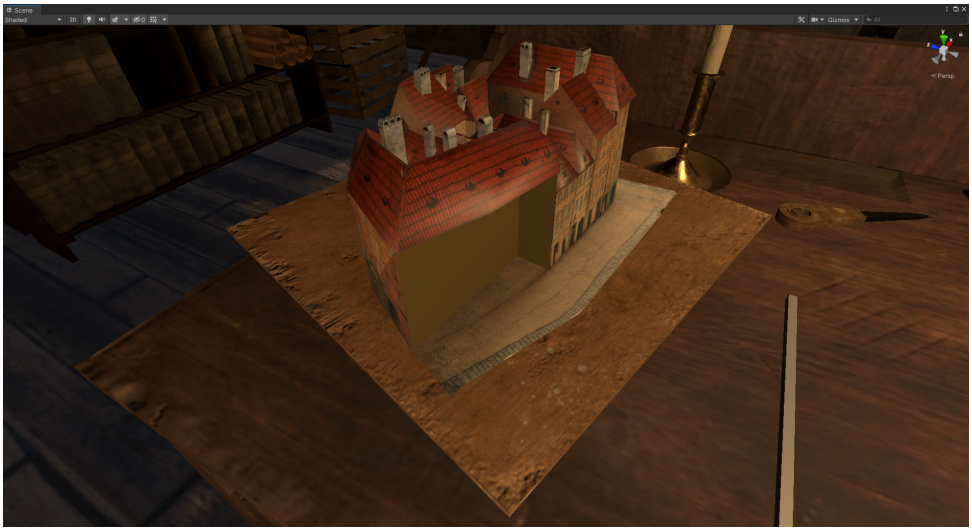


Figure 2.4: Part of Langweil's model of Prague with a missing facade, located on a rotating pedestal.

2.2 Realistic virtual experience

To create a realistic virtual experience that would resemble the creation of Langweil's model of Prague as closely as possible, we need to carefully choose the proper tools and techniques while balancing realism and enjoyment.

Langweil used a combination of line art and black and coloured ink to sketch, draw and paint facades and other parts of the architecture on cardboard and wooden elements. He then cut them out and glued them together to create the model. [BF96] To recreate this, we need a piece of paper, painting tools that paint (3.1), a knife that cuts (3.2) and a way to put the cut out pieces into place (3.5). The final implementation of these tools can be seen in figure 2.3. In the final application, an incomplete model with some missing facades will already be present in the scene (figure 2.4 shows the model that is used in the final implementation) and the user will be given the option to paint, cut out and glue the part they want.

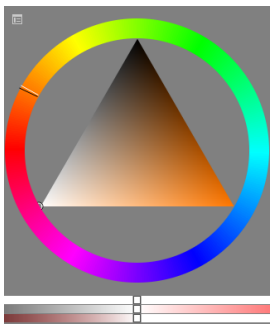


Figure 2.5: Colour wheel in Krita[19]

To create a realistic experience, some features of classic 2D painting applications ought to be omitted:

- An undo action would feel unnatural, yet omitting this feature completely would waste the potential of digital painting, so settling for some sort of an eraser is a good compromise (3.1.7).

[19] Source: Krita

- Colour selection based on a colour wheel, like that which can be seen in figure 2.5, is handy, but creating a limited palette of colours the user is able to choose from makes it much more realistic (3.4). The same applies to choosing the type and size of a tool, where instead of one tool with variable type and size, the user could be given multiple different tools with varying type and size. Although this makes the selection limited, the gained realism may outweigh the lost practicality.

- Layers being a widely used and remarkably useful feature of most modern graphic applications, it would be unwise not to use them. To keep things natural while still keeping some benefits, we may restrict the control of layers in a way that does not allow for layer creation and removal and editing a layer is analogous to using a tool that works only in that layer (3.1.6).

As assigned, the user is expected to sit at a real table that should correspond to the virtual one (3.6). This removes the greatest disadvantage of VR applications with drawing on objects (2.1.3), as it gives us the ability to use haptic feedback provided by the table. The user was also expected to hold an imitation of a real tool that would be tracked and correspond to the virtual one. [Suc21] Due to this, the number of buttons used by the user was kept to a minimum.

2.3 Used technologies

What follows is a list of technologies we decided to use for the implementation:

- Unity [Teca] (requirement) - a game engine that helps creators make 2D and 3D interactive projects by implementing several crucial components such as rendering, lighting, physics, animators, etc., while letting the creators take control by implementing scripts and importing their own assets.

- Universal Rendering Pipeline (*URP* in short) [Tecc] - one of the available rendering pipelines in Unity. It performs a series of operations that take the contents of a Scene and displays them on a screen. Also provides tools to change how some of those operations work.

- SteamVR Unity Plugin [Sof] - a plugin for Unity that provides scripts for an easier start with VR projects in Unity.

- Oculus Rift S[20] and Meta Quest 2 [Met] - VR headsets used when developing and testing the implementation.

[20] No longer available

2.4 Functional and non-functional requirements

Part of the assignment was to implement painting, cutting and other features based on the analysis and design proposals and then merge the implementation with the environment created by Bc. Lukáš Pospíšil. [Pos21] His work also involved implementing painting and cutting, however, it was more of a prototype meant to test out the environment that was expected to be re-implemented [Pos21, Section 7.2], so I did not use that part of his work. The environment he created was already realised as a *Scene* in Unity, though in a different version and using a different rendering pipeline (2.3). This meant I had to upgrade the project, especially materials that had been using shaders no longer available in the current rendering pipeline. The upgrade also changed how the lighting looked. I fixed errors caused by the upgrade, converted shaders to their respective counterparts and removed custom scripts from the game objects. To merge the projects, I exported one of the Scenes and imported it into my project.

In this section, I present a list of requirements the implementation should meet based on the assignment:

2.4.1 Functional requirements

- The program shall allow for painting with several different painting tools that each resemble either a pen or a brush
- The painting tools shall be able to change size (not necessarily at runtime) and color (at runtime)
- The user shall use either a VR controller or a tracked pen as their tool
- Painting, selecting a tool and other similar actions shall happen without the need of buttons or letting go of the controller
- The paper shall have multiple layers for different painting tools and paper texture
- Everything the user paints shall be erasable without disrupting the paper texture or anything else that was painted with a different tool
- The program shall allow the user to zoom/enlarge the paper
- The user shall be able to cut out a piece of the paper
- A part of Langweil's model with missing pieces shall be visible, and the user shall be able to rotate it and fill the missing parts with painted paper cut outs
- The user shall be allowed to remove, edit and put back any cut out
- The program shall have a calibration function to line up the real and virtual tables

2.4.2 Non-functional requirements

- The experience shall resemble creating the Langweil's model of Prague
- The virtual environment shall be similar to Langweil's workplace
- The virtual tools shall resemble Langweil's tools
- The control scheme shall not distract from the experience - what would not appear in the real world should not appear in the virtual world (not always possible or preferable)
- The painting tools shall draw smooth and high resolution strokes
- The user shall sit at a table both in the real and the virtual world
- The user shall not lose tools by dropping them on the ground or throwing them away, they shall return onto the table
- The painting tools shall show which color they are currently painting with
- The paper shall have guidelines for painting and cutting
- Performance shall be suitable for a VR application

Chapter 3

Solution design proposal and implementation

In this chapter I analyse and describe the application requirements and how they could be implemented. As the goal is to implement painting on a 2D paper, most ideas, suggestions and equations will assume that the object we are painting on is a 2D weakly-simple polygon (a polygon that may have touching sides, but is not self-intersecting, [DT07]) with weakly-simple polygon holes located on a 3D plane.

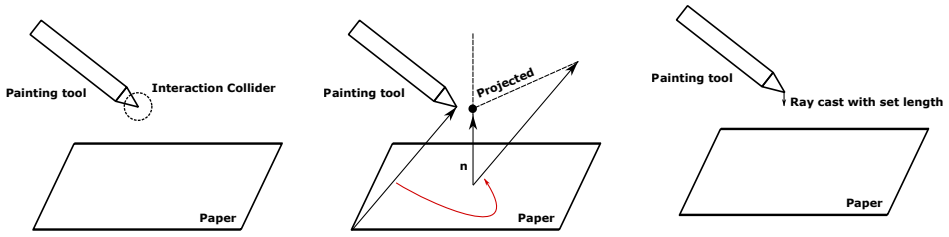
3.1 Painting on a virtual paper

To allow the user to paint on the virtual paper, we need to know when, where, what, and how to draw. In this section, we analyse how we may be able to answer those questions while trying to achieve an intuitive and smooth painting using VR controllers.

3.1.1 Translating controller position into information

To start off, we need to take the controller position and translate it into information that should tell us where and when to paint. In 2D and 3D applications, this would simply involve reading its **state** (*released* and *pressed*) and **position** on the screen. During the pressed state, we would take the controller position and draw at the corresponding location. To determine the exact location, we can directly translate the location on the screen to a location on the canvas in case of 2D applications, but in 3D, we have to use a ray cast that should give us exact UV coordinates of the hit point that correspond to a location in the object's texture (2.1.2). When using an analogue controller, we could also take the **intensity** into account.

In 3D and VR applications, one could also paint using an **intermediate object** that would be controlled by the controller. Instead of using the controller position, we may define a part of the object as the *tip* and use its position instead, though the ray cast must now originate at the tip's position and its direction must be calculated differently. We could also use the **distance** between the tip and the paper as a substitute for intensity and



(a) : The collision metric uses a collider

(b) : The algebra metric uses projection

(c) : The ray cast metric uses a ray cast with set length

Figure 3.1: Diagrams of how different metrics work

controller state, where the pressed state would be the same as the distance being under a certain threshold and the intensity would be equal to how far or close the distance is to that threshold.

Due to the assumption that the user would be holding a tracked pen (2.2), it was decided that the tool would be represented using a 3D model (the tip being defined at the tip of that model) whose virtual location would correspond to the location of the tracked pen. Assuming that the paper lies within a 3D plane (3), the ray direction should be the reverse of that plane's normal. There are several ways to measure the distance to determine the state:

- **The collision metric**: Add colliders to both the paper and the tool tip. The size of the tip collider determines the interaction distance.

- **The algebra metric**: Project the vector of distance between the paper position and tool tip onto the paper normal. The magnitude of the projected vector is equal to the distance:

  $d = \|\langle P_{pen} - P_{paper} \mid \overrightarrow{n} \rangle \overrightarrow{n}\|$

  where

  $d :=$ distance , $P_{pen} :=$ tip coordinates, $P_{paper} :=$ paper coordinates, $\overrightarrow{n} :=$ paper normal

- **The ray cast metric**: Set the ray's length equal to the maximum distance threshold. The state is pressed when the ray reaches the paper.

As Unity, the engine I am using for the implementation, provides us with a Raycast method [Tecb, Physics.Raycast] that takes a parameter defining the ray length and the ray cast is required anyway, I decided to use the ray cast metric.

(a) : Disc drawn using implicit representation.

(b) : Disc drawn using explicit representation.

Figure 3.2: Grids showing a disc get drawn using the different representations.

3.1.2 Drawing a shape

Information in the form of exact integer texel coordinates (or floating point UV coordinates which can be transformed to texel coordinates) and colour should be given by the painting tool (3.1.1). This gives us the ability to draw a colour at the specified coordinates, making the tool paint, as can be seen in figure 3.3. However, to make each tool feel unique and more interesting, some sort of a shape should be drawn instead (centred at the given coordinates). The tool intensity could then also be used to modify the size of the shape.

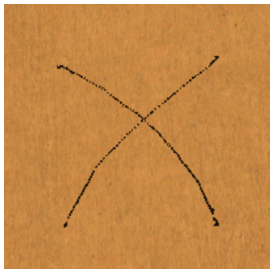


Figure 3.3: X drawn using only UV coordinates

I considered 2 possible methods of representing and drawing a shape:

- **Implicit representation**, figure 3.2a: The shape is represented as a 2D function transformed from a equation defining a shape, where drawing is allowed for a set of predefined values. Let us use the equation for a disc with radius r centred at given texel coordinates (s, t) as an example:

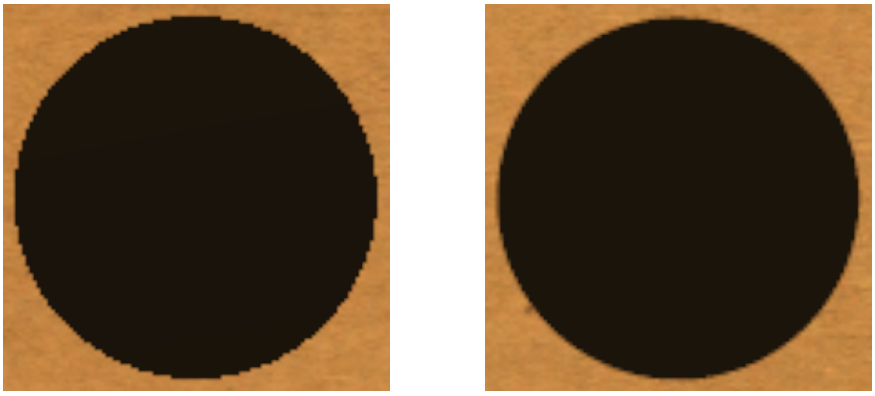
  $(x - s)^2 + (y - t)^2 \leq r^2$

  We may now rearrange the equation so that all variables (x and y) and constants (s, t, and r) are on one side of the equation and define it as a 2D function:

  $F(x, y) = (x - s)^2 + (y - t)^2 - r^2 \leq 0$

  This tells us that for any set of coordinates (x_s, y_t) the value of $F(x_s, y_t)$ determines whether they are inside (negative values) or outside (positive values) the disc (equality additionally suggests that they lay exactly on the edge). Additionally, changing r changes the size of the disc. The disc would then be drawn using

  1: $d \leftarrow F(x_s, y_t)$
  2: **if** $d \leq 0$ **then**
  3: DRAW(x_s, y_t)
  4: **end if**



(a) : Disc drawn without anti-aliasing **(b) :** Disc drawn with level 1 anti-aliasing

Figure 3.4: Comparison of discs drawn using implicit representation with and without anti-aliasing

- **Explicit representation**, figure 3.2b: The shape is represented as a 2D matrix with boolean values (aka *bitmap*), where *true* means draw. This allows for any shape to be defined as long as it fits inside the matrix. Dimensions of the matrix should be odd so that the centre is clearly defined. The bitmap would then be drawn using

```
1: x, y ← TRANSLATE(x_s, y_t, s, t)        ▷ Depends on axis orientation
2: if bitmap[x][y] is true then
3:     DRAW(x_s, y_t)
4: end if
```

Both representations can be modified to achieve a smoother looking shape (see figure 3.4b) by allowing the drawn colour to have an **intensity** (until now, each texel would either stay the same or be replaced by the tool's colour). Jagged edges that might appear when drawing implicit shapes[1], as seen in figure 3.4a, may be removed using some sort of an **anti-aliasing** (AA in short) technique. [ŽBS05, Section 2.4] For explicit shapes[2], the bitmap can be adjusted to use **floating point** values (preferably normalised to range $\langle 0, 1 \rangle$) instead of boolean values to represent any intensity.

The implicit representation is more suitable for shapes easily defined by mathematical equations, as even though the same shape is always definable using the explicit representation as well, the implicit one is usually easily parameterised (e.g. well-defined scalable size/radius, option to add effects like gradients based on the value of $F(x, y)$ etc.).

[1] implicit shape = shape defined using an implicit representation
[2] explicit shape = shape defined using an explicit representation

I decided to use the following:

- Implicit disc shape with **FSAA super-pixel** anti-aliasing for the pen - super-pixel AA is easy to implement yet effective [ŽBS05, Subsection 2.4.1]
- Explicit shapes using **textures** for the brushes - only 1 bitmap per shape for all sizes is needed, as UV coordinates are real numbers, meaning they can be translated, scaled and rotated using linear algebra
- $SRC\_ALPHA + 1 - SRC\_ALPHA$ colour blending
- $max(ONE, ONE)$ alpha blending

3.1.3 Painting on the object

The easiest way to change an object's appearance is to modify the object's texture. At the start of the application we create or copy the original texture, then each frame use information given by the tool to modify it and let the engine display the final texture on the object. There are 2 methods I considered, although I also mention the method used by Bc. Lukáš Pospíšil in his bachelor's thesis [Pos21] for comparison:

- **CPU method**: Use the CPU to modify textures and send the final texture to the GPU to display it
- **GPU method**: Use the CPU to send small amount of information (given by the tool) to the GPU and use the GPU to modify and display the texture
- **L.P.'s method**: Create miniature coloured objects above the texture placed in the world, then use an orthogonal camera to render the final texture

Each method has its positives and negatives (see table 3.1 for comparisons), however, when testing the first 2, the CPU method proved to be much worse in terms of performance (presumably due to the bottleneck mentioned in table 3.1), so I decided to use the GPU method. To implement it, I use Unity's RenderTextures [Tecb, RenderTexture], which are textures that live primarily in GPU memory and are modified by rendering into them. They are usually used as render targets for cameras, allowing camera output to be used as textures (e.g. for real-time in-game cameras or maps) or allowing developers to implement their own post-processing effects. I use them as the *dest* parameter of Unity's static method Graphics.Blit [Tecb, Graphics.Blit], which copies (renders) a source texture into a destination render texture using a shader (see diagram 3.5). The shader is what modifies the texture. Tool information, such as centre coordinates, size and colour and other additional information is passed to the shader through shader properties (uniform variables). The

CPU	GPU	L.P.
Intermediate textures live in CPU memory	Intermediate textures live in GPU memory	No intermediate textures
Uses CPU to write into textures	Uses GPU to write into textures	Uses game objects to "write" into the texture
Heaviest load on CPU	Heaviest load on GPU	Heaviest load on CPU
Easy to read from interm. textures	Difficult to read from interm. textures	Need to check game objects
Sequential write to selected texels	Parallel write to all texels	Sequential object creation
Bottleneck in CPU to GPU transfer speed	No bottleneck	Bottleneck in game object creation speed
Modifies texture without layers	Modifies texture without layers	Does not modify texture

Table 3.1: Comparison of different methods to modify a texture.

source texture parameter of the Blit method may or may not be used, as textures are also considered a property.

The render texture is initialised with the original paper texture and set as the main texture of the paper's material. Before each render, the shader properties are set. We can render into the texture every frame, but setting up a queue of tool positions with information and a buffer of properties that can be processed in the shader at once allows us to render with different frequency than the actual FPS, which may lead to better performance (I render every 33 ms = 30 FPS).

3.1.4 Drawing a stroke

Information about the tool, namely its position, is calculated in the FixedUpdate method [Tecb, FixedUpdate], which is independent of the rendering frame rate and runs at a fixed frame rate. This causes strokes drawn with a fast motion to appear completely or partially disconnected, which can be somewhat seen in figure 3.6a. This happens because there is no information that would suggest drawing between two sequential positions calculated in the FixedUpdate method, which may happen to be spread far apart during those fast motions. The phenomenon can be fixed by using some form of interpolation between any 2 positions. I used the DDA algorithm [ŽBS05, Section 3.1], which is formally used to draw lines between 2 points in 2D graphics. It generates a set of pixels (texels in this case) that form a line in a

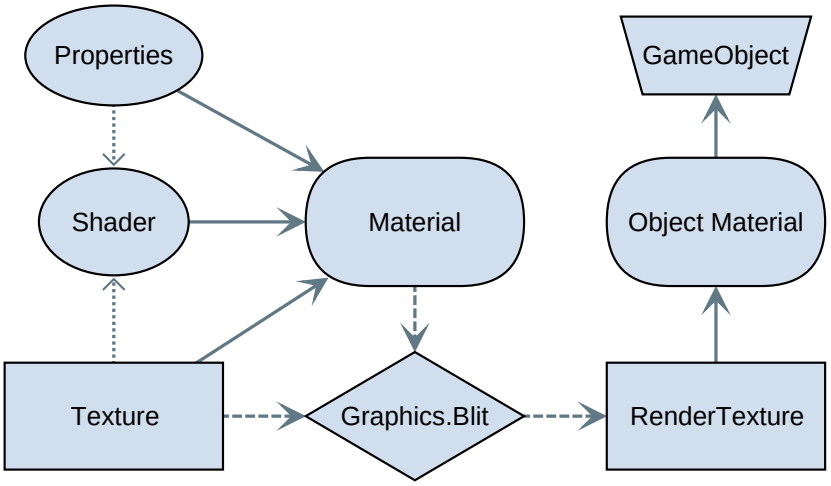
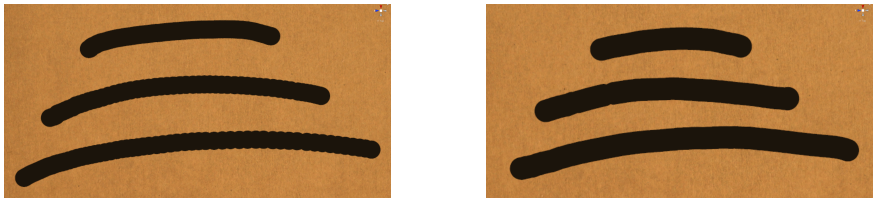


Figure 3.5: Diagram showing how the different components around Graphics.Blit relate. Even though Properties and Texture are managed through Material, they are closely related to Shader. RenderTexture is set as Object Material's main texture so it gets displayed on GameObject.



(a) : Strokes drawn without interpolation **(b) :** Strokes drawn with DDA interpolation

Figure 3.6: Comparison of strokes drawn with increasing speeds (top to bottom) with and without interpolation

discrete representation of space[3], so we have to multiply the coordinates by texture dimensions before running the algorithm and then divide them again after. This method is a form of linear interpolation [ŽBS05, Subsection 22.6.2], which is sufficient (see figure 3.6b). Cubic interpolation [ŽBS05, Subsection 22.6.3] or higher order interpolation may also be used, although they require more than 2 points to be considered.

[3]Visualisation I created and used to better understand the algorithm: https://www.desmos.com/calculator/dqupo9huzi

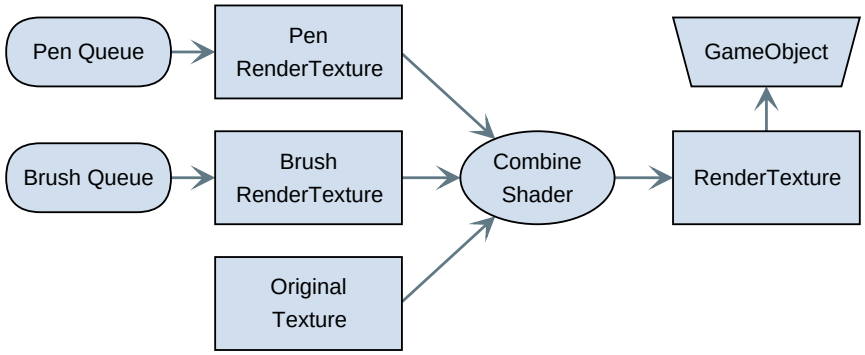


Figure 3.7: Diagram showing how intermediate textures (layers) are combined into a final one that is then displayed on a object.

3.1.5 Constraining the stroke

Digital painting applications often try to help the user paint specific shapes that would be difficult to draw by hand. Fortunately, some of these tools also exist in the real world, namely a ruler and a drawing compass[4], so we can include them without breaking the immersion. The tools would be extremely difficult to implement using physics-based simulation, moreover the user is expected to use only one controller (2.2). A much more plausible implementation would be to let the tools lay on the table and when touched, the painting tool the user is holding would start drawing the corresponding shape.

The shapes drawn by the mentioned tools, a straight line and a circle, require just 2 points to be drawn: the line is trivially defined by 2 points and for the circle, the first point gives us the centre of the circle and the distance between the first and the second point determines the circle's radius. To get these 2 points from the user, we may simply take the point where the tool first touched the paper as the first point and the last point it touched as the second. This allows the user to move the second point by staying within interaction distance and choose it by pulling away. The shape that would be drawn should be displayed to the user while they are choosing the second point, for which we could use Unity's LineRenderers [Tecc, LineRenderer].

To draw the shape itself, we can use the mentioned DDA algorithm (3.1.4) for straight lines and one of the algorithms described in [ŽBS05, Section 3.2] for the circle (which we could also allow to be an ellipse).

[4] Although other tools might exist, these are the relevant ones in our case

3.1.6 Layers

As mentioned in Realistic virtual experience (2.2), layers are an almost essential part of any application with digital painting. Without them, we paint into the original paper texture, irrecoverably changing it and overwriting any previous strokes painted in the same locations. We should have at least 3 layers, one for the original texture, one for the brushes and one for the pen so it does not get overwritten. The number of layers and the tools they accept is set in the editor, without the ability to be changed at runtime. I decided to implement the layers as an array of render textures, where each one has its own queue of tool information to be rendered and the layers are then combined into a final render texture that is set as the object's texture, as can be seen in diagram 3.7.

3.1.7 Erasing

As mentioned in Realistic virtual experience (2.2), an eraser is a good compromise between realism and an undo action. To implement it, I decided to use existing painting tools and only change their colour: without layers, the colour would correspond to the colour of the texel in the original texture in any given location and with layers, the colour only has to have its alpha component set to 0.

3.2 Cutting the paper

To allow the user to cut the paper, we need the ability to divide an existing mesh [ŽBS05, Chapter 6] in two or more pieces based on user input. It is extremely important to choose the correct representation of the paper, because the right choice can make the implementation much easier, while the wrong one much harder.

3.2.1 Prerequisites

- The paper is considered to be a **2D weakly simple polygon with holes** (3), which means there is one enclosing polygon called ***border*** with none to multiple smaller polygons inside called ***holes***. The polygon may be either convex or concave and has only one face that is defined as the area enclosed by the border excluding the holes. The area is considered as ***inside***, while the area around the border and the areas inside holes are considered as ***outside*** or ***empty***.

- Unity supports only **indexed triangular** and quad meshes for rendering surfaces. [Tecb, Mesh Topology]

Half-edge	**Vertex**
Vertex *origin*	Vector2 *coordinates*
Half-edge *twin*	Half-edge *edge*
Half-edge *previous*	**Face**
Half-edge *next*	Half-edge *outer*
Face *face*	List<Half-edge> *inner*

Table 3.2: Table of pointers that each component holds.

- After any new cut, we need to evaluate whether it has cut the mesh in a way that would separate it into pieces, which makes information about **topology** especially important. We will use the term ***the cut cuts*** when that happens.
- We will consider a cut ***valid*** when at least part of it lies on the inside of the paper and it has non-zero length.
- Cutting the mesh that is used for rendering directly would mean manipulating a data structure that is optimised for rendering without any topology information. As a result, it is more convenient to use **separate representations** for cutting and for rendering.
- It is possible to create cutting without any constraints, e.g. representing the mesh as a bitmap and use it as a mask when rendering or use marching squares to generate a mesh[5], but due to the possible complexity of the implementation and little topology information, we decided to **limit the cuts to line segments**.

3.2.2 Paper representation

We view the paper as a planar graph with $1 + count(holes)$ sub-graphs. To ease conversions between vertex coordinates and UV coordinates (used in 3.2.3), an uncut paper with dimensions $Width \times Height$ ($W \times H$) is defined as a rectangular polygon with vertex coordinates $(0, 0)$, $(r, 0)$, $(r, 1)$, $(0, 1)$ and scale s, where $s = H$ and $r = W/H$. UV coordinates of vertex $\boldsymbol{v}$ are defined as $(u, v) = (\boldsymbol{v}.x/r, \boldsymbol{v}.y)$. This means any new piece of paper cut out from the original lies in $\langle 0, r\rangle \times \langle 0, 1\rangle$ space (see figure 3.8).

For the cutting representation, I decided to use **DCEL**[6], which is a well-known data structure seen mostly in applications where topology is of high importance (e.g. modelling) and also make use of some notions from graph theory. At first, I considered using a plain graph representation, but I could not figure out how to correctly separate the graph into multiple pieces when

[5] Speculation only

[6] Doubly connected edge list, aka half-edge data structure

a cut cut. There are many DCEL versions to choose from based on one's needs. I decided to implemented my own version, as I could not find any implementation that would fit my specific needs. It is inspired by [BCK08, Section 2.2], table 3.2 shows which pointers each component holds in detail. The *inner* list of pointers in the Face component is important to correctly represent holes that are not connected to the border with edges and lone cuts (see 2 and 3 in figure 3.8). An empty face is represented as a *null face* pointer in the Half-edge component, which means the paper has only one face by default. To evaluate whether a cut has cut, we just need to check if the number of faces has increased past one. If it has, each face forms a new piece of paper.

3.2.3 Knife tool

To get coordinates of the line segment that will represent the cut (3.2.1), we need to process input from the user and get mesh coordinates of 2 different points. To do this, we can use the same methods as in Translating controller position into information (3.1.1) and Constraining the stroke (3.1.5) to get UV coordinates of the 2 points, that are then easily convertible to mesh coordinates using the conversion method mentioned previously (3.2.2). The issue mentioned in Drawing a stroke (3.1.4) could cause difficulties when trying to connect the cut with an edge, as the points would rarely get placed exactly on it and consequently the cut would never cross it (remember that the collider is defined by the visible mesh). To resolve this, the collider for receiving ray casts should have some external **padding**, so that points can be registered even outside the paper and the cuts are then able to cross the edges. We may also make the collider a simple rectangle surrounding the paper, because cuts made outside the paper are automatically considered invalid (3.2.1). We may use the same collider for painting, as painting into the texture where the texture is not displayed poses no issues.

3.2.4 Placing the cut

Assuming the cut represented as a line segment is valid (3.2.1), we have to *place* it, meaning integrate it into the DCEL representing the paper, creating new vertices and edges. There are 3 possible outcomes when placing a cut:

- The cut intersects edges, creating multiple new vertices and edges. We will call this kind of cut an **intersecting cut** (see 3 and 4 in figure 3.8).
- The cut does not intersect any edges, but both points lie inside the paper. We will call this kind of cut a **lone cut** (see 2 in figure 3.8).
- The cut does not intersect any edges and both points lie outside the paper. We will **ignore** this kind of cut.

To check which kind of cut the current cut is, we need to intersect the line segment representing it with every edge in the graph. We can interpret each

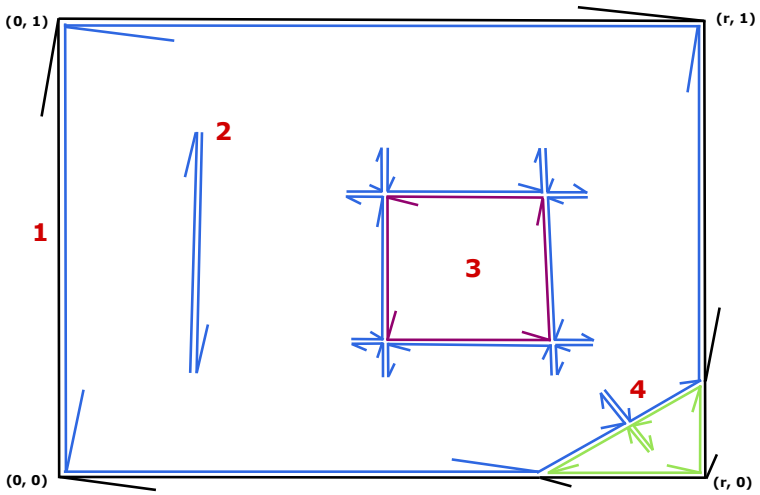


Figure 3.8: Visualisation of cut DCEL. Generated using my implementation and edited for better visibility. Every half-edge is coloured based on which face it belongs to. 1) Border, 2) Lone cut, 3) Hole made using a lone cut and then three intersecting cuts, 4) Split paper with an intersecting cut

edge as another line segment, bringing us to a problem from computational geometry, **line segment intersection**. There are algorithms that, given N line segments, return all intersections. A brute-force algorithm runs in $O(N^2)$ time. Algorithms with better runtimes are well summarised in a github repository of a JavaScript library implementing them by Andrei Kashcha [Kas], nonetheless the best runtime seems to be $O(NlogN + k)$, where k is the number of intersections, presented in [CE92]. In our case, we only need to intersect the cut line segment with all the edge line segments, so a simple brute-force algorithm would run in $O(N)$ time, proving we do not need any of the above mentioned algorithms. Algorithms that avoid the imprecision issues associated with floating-point numbers by avoiding calculating the intersection point also exist [GR00], nevertheless as we need to calculate the intersection point anyway, they need not concern us. I use the algorithm described in [Ant92] including the division-avoiding test.

If the cut is an **intersecting cut**, we need to split edges it passes through, creating vertices that we must connect with edges. To connect them in the right order, we may simply sort the intersection points, because the cut is a line and if 2 subsequent points in the sorted list were to have something in-between them, it would have intersected with that line. Another thing we have to consider is that DCEL consists of half-edge instead of regular edges, so we need to know which half-edge to connect to. This can be solved by testing whether the vertex we are connecting is on the left or on the right of the half-edge[7], the former meaning we can connect it and the latter meaning we need to connect to the half-edge's twin instead. Two analogous methods may be used: determinant sign or dot product sign. If we are connecting

[7]Half-edges are oriented and their incident face is always on the left side

to/from[8] a half-edge that has a *null* aka empty incident face, we do not connect, as we cannot create cuts where there is no paper.

To check if a cut is a **lone cut**, we need to test whether both points lie on the paper. As described in the previous paragraph, these 2 points cannot have any edges between them, meaning both points lie inside the same polygon. Additionally, a piece of paper has only one face (3.2.1), so we need not check to which face the cut would belong to. This implies we only have to test if one point lies on the paper. The test itself is analogous to the well-known computational geometry problem called ***Point-In-Polygon*** (*PnPoly* in short). There are many different strategies [Hai94], but I decided to use an implementation by W. Randolph Franklin published on his website. [Fra] He uses the crossings test mentioned in Eric Haines' article and claims the degenerate case where the ray goes through a vertex is handled correctly. He also claims his algorithm consistently assigns each point to exactly one polygon, suggesting a point lying on a common boundary of 2 polygons always gets assigned to one same polygon. A point that lies on the paper is considered to be inside (3.2.1), so we run the PnPoly test on the border and holes - if the test returns *true* for the border and *false* for the holes, the point is inside, otherwise it is outside.

3.2.5 Converting representations

When using separate representations for cutting and rendering, we need to convert one to the other. The conversion needs to happen only one way (cutting → rendering)[9], that is we are required to convert the mesh only once per paper piece's lifetime to be able to render it[10]. The simplest method of conversion is to use tessellation (specifically triangulation), though we need to consider that the polygons may be concave and the mesh may have holes.

There are 2 tessellation methods I considered that fit our needs: **Constrained delaunay triangulation** [Che89] and **GLU Tessellator**[11], which is part of GLU, the OpenGL Utility Library. Both methods are equal in terms of suitability, so I chose to use a Unity compatible C# port of the GLU Tessellator, the **LibTessDotNet library**. [Gil] The library only needs contours, a winding rule and the desired number of vertices of each polygon as input and gives a tessellated mesh as output, examples can be seen in figure 3.9. The contours are analogous to the polygons forming the paper, the winding rule I use is even-odd and the number of vertices should be set to 3 for triangulation. As the paper polygons have counterclockwise orientation and Unity uses clockwise orientation to determine the front-side of a polygon for rendering, the orientation of the tessellated polygons needs to be reversed before creating the mesh out of them. This can simply be done by reversing the order of vertex indices.

[8] If we are connecting 2 half-edges, both have to have the same incident face
[9] The other way may be needed once at the start, when we only have the render mesh
[10] Assuming we are not rendering the cuts themselves as part of the mesh
[11] http://www.glprogramming.com/red/chapter11.html

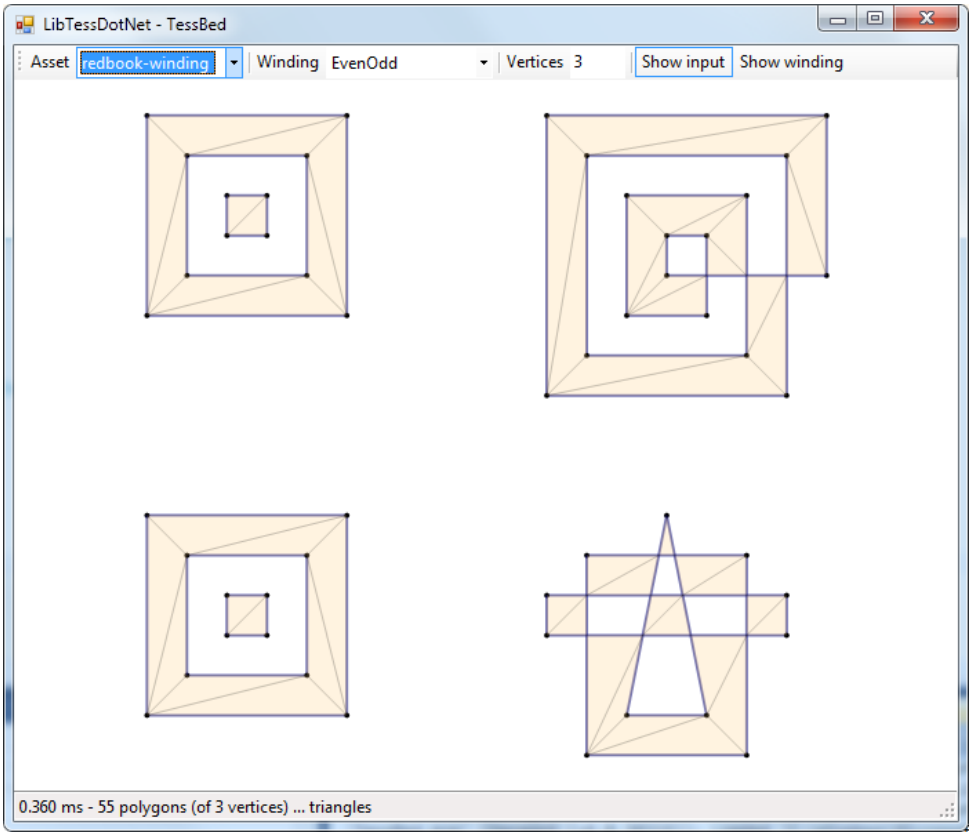


Figure 3.9: Screenshot from LibTessDotNet testbed [Gil]

If we created the mesh using only the generated tessellation, the paper would be one-sided, meaning the paper would be completely transparent from one side. This can be fixed by duplicating the mesh with reversed orientation.

3.2.6 Gluing cuts

As with the Knife tool, we could borrow another idea from the painting section, namely from Erasing (3.1.7). Instead of an eraser, we use glue to represent an undo action, which can also be expanded to not only erasing cuts, but also gluing paper pieces back together.

To **erase cuts**, we would need to allow the user to remove edges from the DCEL data structure. To ensure the user would not erase critical edges and destroy the topology, we would need to limit which edges can be removed. Luckily, non-critical edges are easily identifiable - the half-edges representing them have the same incident face (they represent *bridges* from graph theory). To identify which edges the user wants to remove, we can use a simple metric such as the distance of the interaction point from the line segment representing the edge.

To **glue pieces back together**, two meshes would need to connect and create a new single one. First, we would need to know if the connection is possible. This can be evaluated by checking if there exist any 2 pairs of vertices that meet the following requirements:

- Each vertex from a pair lies on a different piece.
- Vertices that lie one the same piece are connected via a sequence of half-edges with no incident face.
- The vertices are collinear.

If all requirements hold true, the pieces can be connected via those sequences of half-edges. The connection would involve merging or removing vertices and edges that overlap. The separate representations ensure that the paper's render mesh does not degenerate over time as it is created anew each time a new paper is formed. Special care would also need to be taken to correctly combine textures, nevertheless it could be done by rendering the visible part of each texture using a mask.

3.3 Scaling the paper

Although the user is able to zoom onto the paper by simply moving their head closer, a feature that would allow the user to artificially enlarge the paper could help when painting details, and shrinking could save time when working with larger areas. We decided to include a slider (see figure 3.10) that would change the scale of the paper using the tool the user is holding in their hand. The implemented slider has adjustable values for the step and the scale limits and is generated based on the number of steps between the limits. It saves the original scale of the paper at the start and sets a new scale based on the original value each time the step is changed. Testing with users showed that even though the slider model is simple and stands out from the environment, there were no complaints.

Figure 3.10: Slider used for resizing the paper

3.4 Changing brush colour

We decided on a limited palette that changes the brush colour when touched, in accordance with what was mentioned in Realistic virtual experience (2.2). We also agreed that the user should be able to see with what colour the brush is currently painting without needing to paint. The ideal would be to change the texture of the brush's bristles; nevertheless, there are other possibilities that may be less realistic, yet more visually clear. Figure 3.11 shows the implemented palette with a brush that has a small sphere displaying that the colour red is selected.

Figure 3.11: Palette with a brush showing it is painting with red colour

3.5 Moving the paper

There are 4 main incentives to allow the user to move the paper around:

- Repositioning the paper when painting corners, especially when the paper is enlarged
- Separating paper pieces after cutting
- Throwing away clippings
- Fitting the painted and cut out facade into the model

To move the paper, we may either use the physics engine or implement our own way. The physics engine is usually a good option, however, for our case, there are a few obstacles:

- To fully utilise the physics engine in Unity, we would need to use convex colliders. [Tecc] This should be an obvious conflict with what we have, as paper pieces might be concave or even have holes (3). A workaround is possible through approximating the mesh using smaller convex colliders, though it would be difficult to implement.
- Planar colliders do not count as convex colliders, so the paper would need to have thickness.
- Some clippings might get tiny, which might cause the physics calculations problems.

Instead of trying to resolve these issues, I suggest it is more convenient to assume the user only wants to move the paper along the table and implement a simpler method of movement. I decided to create a **special tool** to move the paper that acts as a painting tool, but instead of painting, it picks up the paper while the user is holding the grip trigger. The paper is only picked

up if the tool would normally draw. The tool casts rays from its tip down at the floor and there are 4 possible outcomes when the user drops the paper, depending on where the ray cast hits:

- If the ray hits the table, the paper gets **placed on the table** at the hit location
- If the ray hits a collider placed near the missing facade, the tool casts another ray towards the missing facade and the paper gets **placed into the facade** based on the second ray's hit
- If the ray hits an object marked as trash, the paper gets **deleted**
- If the ray does not hit any of the above, the paper gets **placed at a specified reset location**

To help the user visualise when the tool is within interaction distance and where the paper would get placed when dropped, I swap the paper's material with a semi-transparent one for the former and display a transparent copy of the paper where the paper would get placed each frame for the latter.

3.6 Calibration

To ensure the objects' locations in the real world correspond to the ones in the virtual world, we should perform some sort of calibration. There are 2 things that should be calibrated: the **table** and the **tools**.

To calibrate the **table**, we need to define at least two[12] points on the virtual table, one to determine the centre and one to determine the direction of the table's edge, and match them with their counterparts on the real table. To determine the real world counterparts, I use the positions of the controllers. The virtual position of the controllers is determined based on the tracking origin, so to align the controllers correctly, we need to translate and rotate the tracking origin. To do this, I use the following algorithm:

1: $\vec{t} \leftarrow P_{centre} - P_{right}$
2: $P_{origin} \leftarrow P_{origin} + \vec{t}$
3: $\vec{t}_{\Delta} \leftarrow P_{centre} - P_{origin}$
4: $\vec{v}_{table} \leftarrow P_{direction} - P_{centre}$
5: $\vec{v}_{controller} \leftarrow P_{left} - P_{right}$
6: $\alpha \leftarrow angle(\vec{v}_{table}, \vec{v}_{controller})$
7: $P_{origin} \leftarrow \boldsymbol{T}^{-1}(\vec{t}_{\Delta})\boldsymbol{R}(\alpha)\boldsymbol{T}(\vec{t}_{\Delta}) \cdot P_{origin}$

where P_{centre} := centre of the table along the edge, $P_{direction}$:= a point along the edge of the table left of P_{centre}, P_{right} := position of the right controller, P_{left} := position of the left controller, P_{origin} := tracking origin, $\boldsymbol{T}$:= translation matrix, $\boldsymbol{R}$:= rotation matrix (or quaternion)

[12] If the table's surface was not horizontal, we would need three

Calibrating the **tools** so that the controller touching the real table corresponds to the tool tip touching the virtual table is much more difficult. The controllers are represented using game objects in Unity, so setting the tool's parent to or making the tool follow those game objects is the first step. Although this makes the tool move by moving the controllers, it does not guarantee the correct position of the tip. As the controllers are represented using a singular point in space (the representing game object's position), there is no good way I could find to precisely figure out where specific points on the controller should correspond to in the virtual world. I decided to resolve this by approximating the correct position and rotation and then letting the user calibrate it on their own, though this proved to be confusing and troublesome for some users (4.3). Testing also showed that even though the tools were calibrated correctly for a specific position, they would drift to incorrect positions when moved elsewhere, presumably due to imprecise tracking.

Chapter 4

Implementation testing

The implementation was tested in terms of **performance** and with **users** by 2 groups of participants. The first group was aged around 8-11 years old, the second group around 12-15 years old. There were 10 participants that tested in total. Both user tests were conducted on the same day.

4.1 Performance testing

Performance testing was done in Unity Editor with AMD Ryzen 5 1500X Quad-Core processor and AMD Radeon RX 580 graphics card. Frames per second were measured using Unity's built-in statistics feature. The SteamVR package seems to limit FPS when using VR to 80 FPS, so I will consider this the maximum FPS. When testing the scenes with and without the implemented features, both ran at max FPS. Using the tools did not affect the FPS, the only exception being that when the texture of the paper passed a certain limit[1], the painting tools caused the FPS to decrease proportionally with increasing dimensions of the texture. This is presumably caused by the DDA algorithm, which iterates over all texels between two points. With increasing texture dimensions, the texel distance between the two points increases, even though the euclidean distance stays the same. This can be improved by both increasing the step of the DDA algorithm (3.1.4) to be more than 1 texel when the texture dimensions reach a certain limit, and by increasing the size of the render queue (3.1.3).

4.2 User test scenario

Testing was performed inside one of the classrooms in the university building. A matching version of Unity (2020.3.21f) had been installed on one of the computers. Oculus Quest 2 was provided and connected via a cable to the same computer. The project was run in the Unity Editor.

[1] Texture with dimensions 4096x3072 lowered the FPS to around 30 FPS

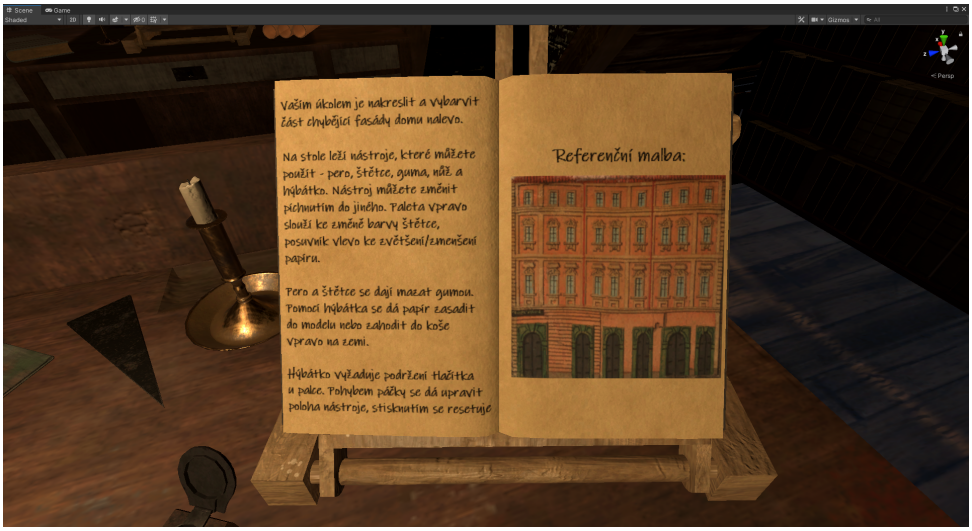


Figure 4.1: Book with instructions and a reference painting for users.

The participants from each group were divided into several smaller groups (called *teams* for clarity) consisting of 3-5 participants. Each team was then additionally split, so the testing was done with a single participant at a time from each team.

Each participant had around 20 minutes to learn the controls, complete a task and fill out a questionnaire.

4.2.1 Learning the controls

- **Expected duration**: 5 minutes

- **Course**: The participant was instructed on how to use the controller and put on the headset. Instructions on how to use the various tools were provided in the virtual world through a text written in a book (see figure 4.1), nevertheless most participants were instructed verbally as they preferred it when asked. The participant was then introduced to the pen, brush and eraser tools. Some were told about the slider for resizing during this section. When the participant was ready or they exceeded the time limit, they moved on to the next section.

4.2.2 Completing the task

- **Expected duration**: 10-13 minutes

- **Course**: The participant was asked to look at the instruction book, which had a reference painting of a facade on the second page (see figure 4.1). They were told that they should paint a similar-looking painting which would then be cut out and put inside a model located on the left where a facade was missing (see figure 2.4). They were then left to paint, but were provided additional support with adjusting the tools when needed and were reminded of the time left semi-periodically.

4.2.3 Filling out the questionnaire

- **Expected duration**: 2-4 minutes

- **Course**: When the previous task was completed or the allocated time was being exceeded, the participant was asked to take off the headset, relocate to another computer where a questionnaire (Google forms) was prepared and fill the questionnaire out. The questionnaire was anonymous and the participant was left alone unless they came across a problem, which might have lead to some to some issues with grading (4.3.2).

4.3 User test results

4.3.1 Overview

The most prevalent issues were with tool positions and cutting.

- Each tool has its own offset that does not save between sessions, so manual adjustment had to be done for each participant and each tool separately. For the first group, the adjustments were made mostly by users with some amount of help; for the second group, I decided to do a rough manual adjustment before each participant arrived. Some small adjustments still had to be made, nevertheless, the adjusting took less time and I think it felt less tedious. Although, even with the adjustments, the tracking sometimes caused the tools to lift off or sink into the table even though the controller was kept on the real table, which seemed to be caused by the distance of the controller from the headset.

- Due to cutting not having all the features that were planned (particularly snapping and erasing), it was expected to be the weakest part of the application. The cutting itself worked well, but without snapping and with imperfect tracking and offset, it was extremely hard for users to make continuous cuts and understand why the paper is not yet cut out. I changed the activation distance of the knife from 5mm to 7.5mm for the second group, which helped a bit.

4.3.2 Questionnaire results

The questionnaire (B) had 6 free-form questions, 10 questions with a choice, and 9 questions with grades.

Data about the participants extracted from answers to choice questions:

- All but one were right-handed
- None were experienced with VR, either having never touched it or having used it only a few times in the past (with only one having used the standard controllers)
- Of the ones that used VR in the past, none had physical complications
- Experience with painting was mixed, with 2 having no experience, 5 painting occasionally and 3 painting often
- 7 had no experience with gluing models
- All but the left-handed individual (who uses Procreate with a tablet and a pen) had no experience with digital art
- 7 had no physical complications after the test (including the ones that experienced VR before) - the remaining 3 described mild dizziness, tired eyes and uncertainty
- All but the left-handed individual (who would prefer it to be shorter) were content with the activation distance

The feedback in the free-form answers is mostly positive:

- The most prominent feelings were neutral to highly positive (e.g. "*excitement*", "*super good*", "*this was the first time and it was okay*")
- The favourite parts were either drawing, painting, the environment or everything
- The least favourite parts were either nothing or the cutting (one answered "*fuzzy image*", which was presumably due to sub-optimal headset configuration)
- None except one (who would prefer a larger range of colours on the palette) was missing anything

All participants gave grades in either ranges 1-3 (7) or 3-5 (3), where 1 is the best and 5 the worst. Charts 4.2 to 4.10 show the ratios of given grades.

Due to conflicting answers given in the free-form questions by participants who gave the low grades, where, for example, one said their favourite part was the environment while giving it a 5, and another giving the grade 5 six times while saying they liked everything, I present 2 versions of each chart: The left one has the original answers and the right one assumes those 3 participants inverted the scale.

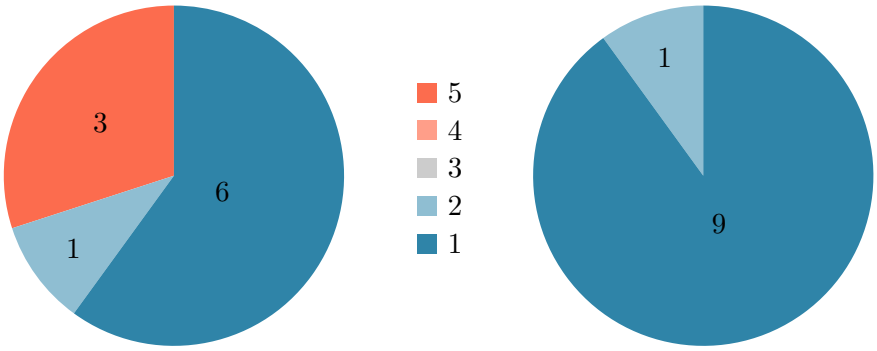


Figure 4.2: Environment grades

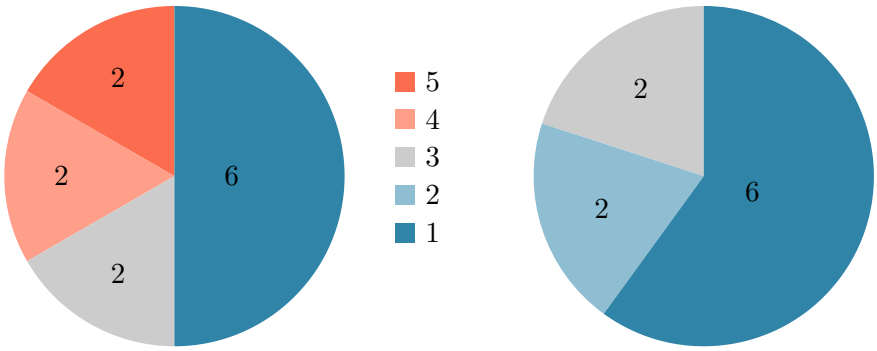


Figure 4.3: Drawing grades

4.4 User test summary

User testing was generally successful. There were no major technical difficulties and the application ran smoothly. The results show that some parts of the implementation, mainly cutting, still need some work; nevertheless, the test participants' experience seemed enjoyable overall. Most of the participants had little to no experience with VR, proving that the application is accessible for inexperienced users. The presence of the left-handed individual also proved that the application is accessible in terms of the dominant hand of the user. Regarding feedback on features that were missing, the application should be tested with more experienced users, as they might have a better understanding on the implementation's potential.

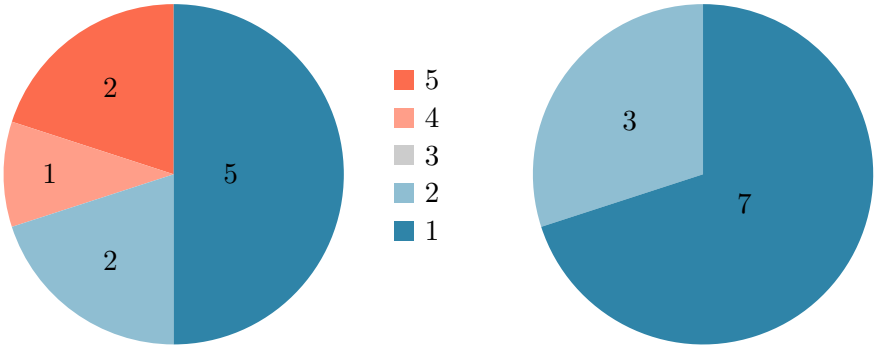


Figure 4.4: Painting grades

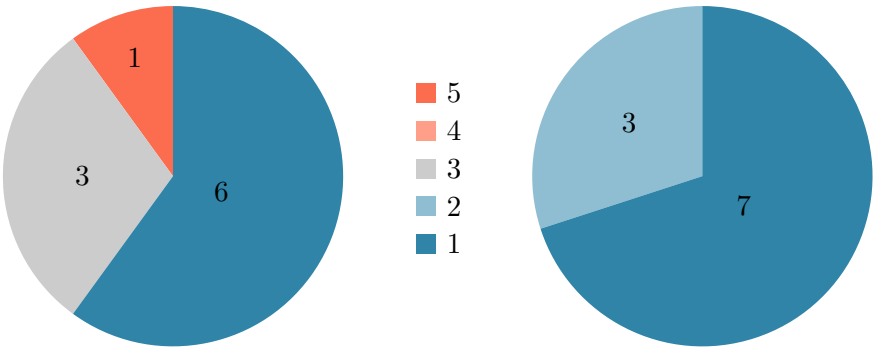


Figure 4.5: Erasing grades

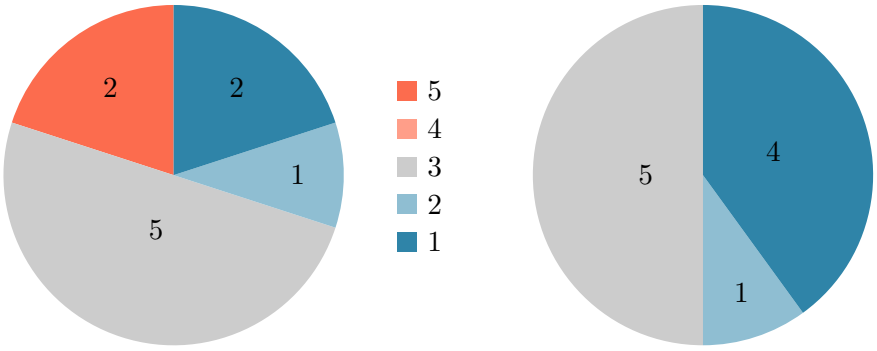


Figure 4.6: Cutting grades

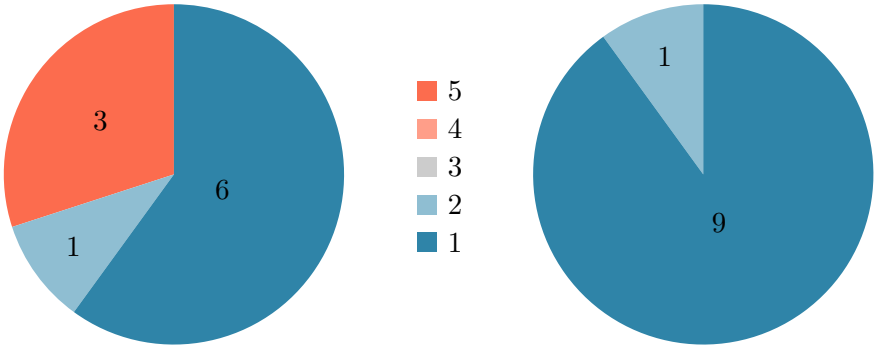


Figure 4.7: Resizing grades

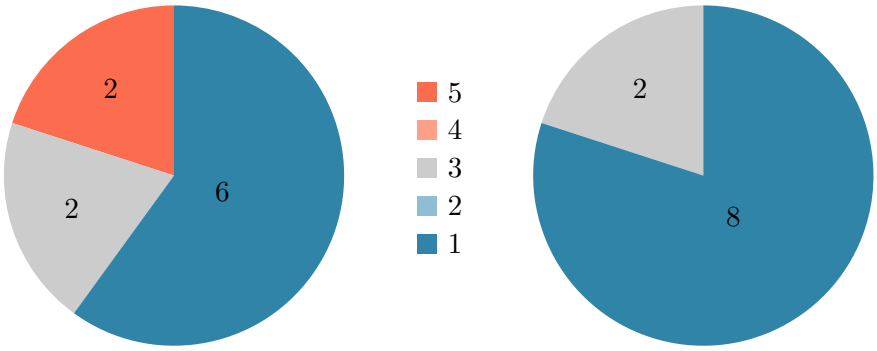


Figure 4.8: Paper movement grades

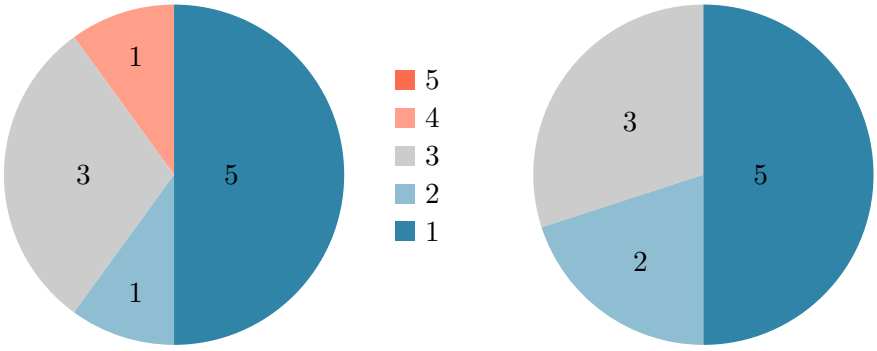


Figure 4.9: Physical controller to virtual tool correspondence grades

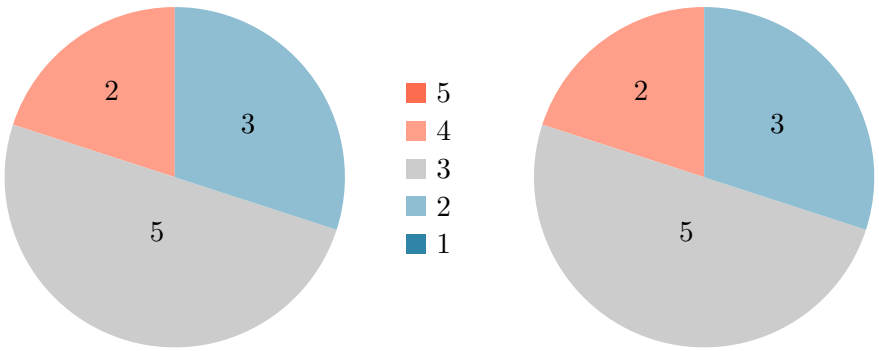


Figure 4.10: Similarity to real drawing grades

Chapter 5

Conclusion

An overview of digital painting and its methods was provided. The methods were analysed in context of creating a realistic virtual experience reminiscent of creating Langweil's model of Prague, and relevant adjustments were proposed. The proposed methods were then implemented and used in conjunction with a realistic scene to create the experience. Painting and drawing using brushes and a pen were implemented successfully (excluding constrained painting); nevertheless, the implementation is ready for improvements wherever appropriate. Cutting was partly implemented, yet user test results show that adjustments and features like cut erasing and gluing are needed for the experience to be sufficiently enjoyable. Tool calibration should be improved, as the current method is not intuitive for users, takes a long time to set up and the drifting sometimes makes using the tools difficult or even frustrating. The rest of the suggested features were implemented without issues. Testing was done both performance-wise, showing no major issues, and with users, of which the relevant outcomes have been discussed.

Bibliography

[Che89] L. P. Chew. 'Constrained delaunay triangulations'. In: *Algorithmica* 4 (1989), pp. 97–108. DOI: 10.1007/BF01553881.

[Ant92] Franklin Antonio. *Graphics Gems III*. Ed. by David Kirk. Academic Press Professional, Inc., 1992, pp. 199–202. ISBN: 0124096719.

[CE92] B. Chazelle and H. Edelsbrunner. 'An Optimal Algorithm for Intersecting Line Segments in the Plane'. In: *J. ACM* 39.1 (1992). ISSN: 0004-5411. DOI: 10.1145/147508.147511.

[Hai94] Eric Haines. *Graphics Gems IV*. Ed. by Paul S. Heckbert. Academic Press Professional, Inc., 1994, pp. 24–46. ISBN: 0123361559.

[BF96] K. Bečková and M. Fokt. *Svědectví Langweilova modelu Prahy*. 1st ed. Schola ludus Pragensia, 1996. ISBN: 80-900668-8-7.

[GR00] M. Gavrilova and J.G. Rokne. 'Reliable line segment intersection testing'. In: *Computer-Aided Design* 32.12 (2000), pp. 737–745. ISSN: 0010-4485. DOI: 10.1016/S0010-4485(00)00050-6.

[ŽBS05] J. Žára et al. *Moderní počítačová grafika*. 2nd ed. Computer Press, 2005. ISBN: 80-251-0454-0.

[DT07] A. Dumitrescu and C. D. Tóth. 'Light Orthogonal Networks with Constant Geometric Dilation'. In: *Lecture Notes in Computer Science*. Vol. 4393. Springer, Berlin, Heidelberg, 2007. DOI: 10.1007/978-3-540-70918-3_16.

[BCK08] M. Berg et al. *Computational Geometry. Algorithms and Applications*. 3rd ed. Springer Berlin, Heidelberg, 2008. ISBN: 978-3-540-77973-5. DOI: 10.1007/978-3-540-77974-2.

[Pos21] Lukáš Pospíšil. 'Virtuální zážitek „Tvorba Langweilova modelu Prahy"'. bachelor thesis. Czech Technical University of Prague, 2021. URL: https://dspace.cvut.cz/handle/10467/92807.

[Suc21] Filip Suchý. 'Sledování reálných objektů ve virtuální realitě'. master thesis. Czech Technical University of Prague, 2021. URL: http://hdl.handle.net/10467/95329.

[Fra] W. R. Franklin. *Point Inclusion in Polygon Test*. URL: https://wrf.ecse.rpi.edu/Research/Short_Notes/pnpoly.html (visited on 01/05/2022).

[Gil] Rémi Gillig. *LibTessDotNet library*. URL: https://github.com/speps/LibTessDotNet/ (visited on 01/05/2022).

[Kas] Andrei Kashcha. *Segments intersection detection library*. URL: https://github.com/anvaka/isect/ (visited on 01/05/2022).

[Met] Meta. *Meta Quest 2*. URL: https://store.facebook.com/quest/products/quest-2/ (visited on 01/05/2022).

[Mus] Prague City Museum. *Langweilův model Prahy*. URL: https://www.muzeumprahy.cz/navstivte-nas-historie-langweiluv-model-prahy/ (visited on 01/05/2022).

[Sof] Valve Software. *SteamVR Unity Plugin*. URL: https://valvesoftware.github.io/steamvr_unity_plugin/ (visited on 01/05/2022).

[Stu] YORD Studio. *Projects*. URL: https://yordstudio.com/cs/projekty/ (visited on 01/05/2022).

[Teca] Unity Technologies. *Unity*. URL: https://unity.com/ (visited on 01/05/2022).

[Tecb] Unity Technologies. *Unity Scripting Reference 2020.3*. URL: https://docs.unity3d.com/2020.3/Documentation/ScriptReference/index.html (visited on 01/05/2022).

[Tecc] Unity Technologies. *Unity User Manual 2020.3*. URL: https://docs.unity3d.com/2020.3/Documentation/Manual/index.html (visited on 01/05/2022).

Appendix A

Attachment list

Attachments:

- Unity_2D_VR_painting.7z.001-011 - Unity project containing the implementation
- questionnaire.pdf (B) - Questionnaire used after testing (4.3.2)
- imgs.zip - Images showcasing the project and implementation
- showcase.mp4 - Video showcasing the implementation
- README-en.pdf (C) - English version of the README describing how to build the project
- README-cz.pdf (C) - Česká verze README popisující jak na build projektu

Appendix B

Questionnaire

Testování tvorby Langweilova modelu ve VR

*Povinné pole

1. Jakou verzi aplikace jsi používal/a? *

   Označte jen jednu elipsu.

   - ◯ VR1 - kovalmat
   - ◯ VR2 - papayrob

2. Kterou rukou píšeš? *

   Označte jen jednu elipsu.

   - ◯ Levou
   - ◯ Pravou
   - ◯ Levou i pravou stejně dobře.

3. Jak často používáš VR headset? *

   Označte jen jednu elipsu.

   - ◯ Nikdy jsem jej nepoužíval/a. *Přeskočte na otázku 6*
   - ◯ Velmi příležitostně jej používám (párkrát v roce). *Přeskočte na otázku 4*
   - ◯ Často (v průměru alespoň 1x měsíčně) *Přeskočte na otázku 4*

Zkušenosti s VR headsetem

4. Jaký způsob ovládání u headsetu jsi v minulosti používal/a?

   Zaškrtněte všechny platné možnosti.

   - ☐ Volné ruce (bez ovladačů)
   - ☐ Standardní ovladače
   - ☐ Jiné (např. specializované ovladače)

5. Měl/a jsi někdy při práci s VR headsetem fyzické potíže (závrať, točení hlavy apod.)?

Označte jen jednu elipsu.

- ◯ Ano, výrazné.
- ◯ Ano, ale jen mírné.
- ◯ Ne, neměl jsem.

Tvorba grafického obsahu

6. Jaký je tvůj vztah ke kreslení na papír? *

Označte jen jednu elipsu.

- ◯ Téměř vůbec si nekreslím, nemám na to talent.
- ◯ Kreslím příležitostně
- ◯ Kreslím často (v průměru alespoň 1x měsíčně)

7. Lepil/a jsi někdy papírové modely? *

Označte jen jednu elipsu.

- ◯ Nikdy jsem papírový model nelepil/a.
- ◯ Papírový model jsem několikrát slepil.
- ◯ Tvořím často (v průměru alespoň 1x měsíčně)

8. Jaký je tvůj vztah ke tvorbě grafického obsahu (obrázky, 3D modely apod.) na počítači? *

Označte jen jednu elipsu.

- ◯ Nikdy jsem grafický obsah netvořil/a. *Přeskočte na otázku 11*
- ◯ Obsah tvořím jen příležitostně. *Přeskočte na otázku 9*
- ◯ Tvořím často (v průměru alespoň 1x měsíčně) *Přeskočte na otázku 9*

Tvorba grafického obsahu detailně

9. Jaké editory pro tvorbu grafického obsahu používáš?

10. Jaká zařízení při tvorbě používáš?

Zaškrtněte všechny platné možnosti.

- [ ] Počítačovou myš
- [ ] Dotykový tablet bez pera
- [ ] Dotykový tablet s perem
- [ ] Jiné: _____

Dotazník po testu

11. Jak se cítíš po fyzické stránce? Měl/a jsi nějaké potíže (závrať, točení hlavy apod.)?

_____

12. Jaký je tvůj nejvýraznější pocit z používání aplikace?

_____

13. Jak jsi spokojený s prostředím? *

Označte jen jednu elipsu.

	1	2	3	4	5	
Velmi spokojený	◯	◯	◯	◯	◯	Velmi nespokojený

14. Jak jsi spokojený s kreslením perem? *

Označte jen jednu elipsu.

	1	2	3	4	5	
Velmi spokojený	◯	◯	◯	◯	◯	Velmi nespokojený

15. Jak jsi spokojený s malováním štětcem? *

Označte jen jednu elipsu.

	1	2	3	4	5	
Velmi spokojený	◯	◯	◯	◯	◯	Velmi nespokojený

16. Jak jsi spokojený s mazáním? *

Označte jen jednu elipsu.

	1	2	3	4	5	
Velmi spokojený	◯	◯	◯	◯	◯	Velmi nespokojený

17. Jak jsi spokojený s řezáním? *

Označte jen jednu elipsu.

	1	2	3	4	5	
Velmi spokojený	◯	◯	◯	◯	◯	Velmi nespokojený

18. Jak jsi spokojený se zvětšováním/zmenšováním papíru? *

Označte jen jednu elipsu.

	1	2	3	4	5	
Velmi spokojený	◯	◯	◯	◯	◯	Velmi nespokojený

19. Jak jsi spokojený s přemísťováním papíru? *

Označte jen jednu elipsu.

	1	2	3	4	5	
Velmi spokojený	◯	◯	◯	◯	◯	Velmi nespokojený

20. Jak jsi spokojený/á s korespondencí pozice virtuálního nástroje a reálného ovladače? *

Označte jen jednu elipsu.

	1	2	3	4	5	
Velmi spokojený	◯	◯	◯	◯	◯	Velmi nespokojený

21. Co se ti při práci v aplikaci nejvíc líbilo?

_____

22. Co se ti při práci v aplikaci nejvíc nelíbilo?

_____

23. Je něco, co v aplikaci chybí?

_____

24. Jaký je tvůj názor na rozmanitost nástrojů? *

Označte jen jednu elipsu.

	1	2	3	4	5	
Nástrojů je málo.	◯	◯	◯	◯	◯	Nástrojů je moc.

25. Jaký je tvůj názor na vzdálenost aktivace nástrojů (např. kdy začne pero kreslit)? *

Označte jen jednu elipsu.

- ◯ Nechal/a bych tak jak to je teď.
- ◯ Preferoval/a bych kratší vzdálenost (nástroje mohou vynechávat).
- ◯ Preferoval/a bych delší vzdálenost (nástroje mají tendenci aktivovat se když to není žádáno).
- ◯ Preferoval/a bych možnost aktivovat nástroj pomocí stisknutí tlačítka.
- ◯ Jiné: _____

26. Přišla ti práce v aplikaci podobná kreslení na papír *

Označte jen jednu elipsu.

	1	2	3	4	5	
Velmi podobná	◯	◯	◯	◯	◯	Vůbec nebyla podobná

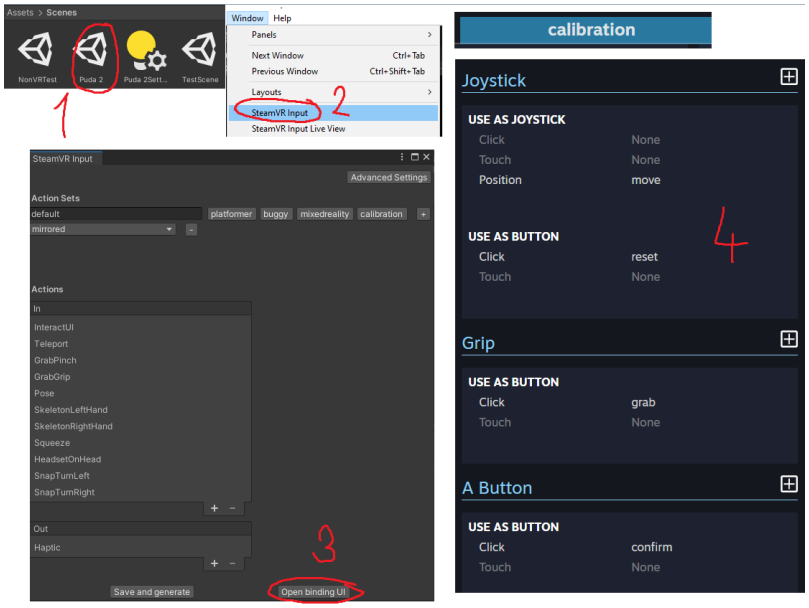
Appendix C

Readme

There is an attached video showcasing the implementation (uncompressed version on project's git repository)

Instructions on how to build and use the implementation accompanying this thesis

- You can download the implementation from the project's archive in parts or from https://gitlab.fel.cvut.cz/langweil-mmp/vr_kresleni_papay
- You should have a Unity project in a folder called **Unity_2D_VR_painting**
- Now install
  - Unity version 2020.3.21f1 (other versions are NOT guaranteed to work) from https://unity3d.com/get-unity/download/archive
  - SteamVR from https://store.steampowered.com/app/250820/SteamVR/
- Open the project using Unity and start SteamVR
- Open the scene **Puda 2** in folder Scenes (1)
- Open Window → SteamVR Input → Open binding UI → Edit (current binding) → Calibration (2-4)
- Set up bindings (4)

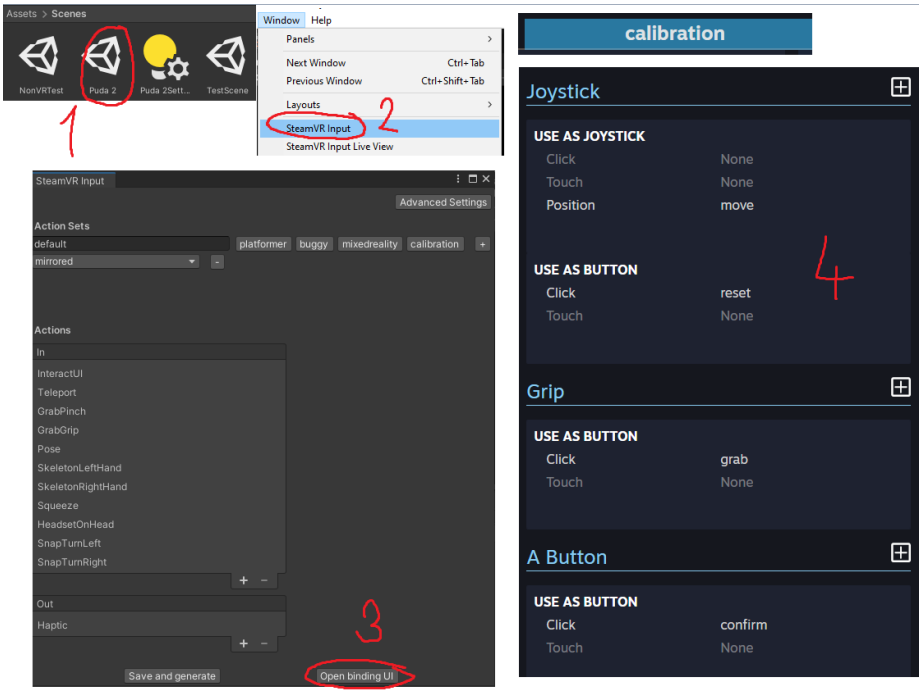


- Click **Play**
- Put right controller at the center of the edge of the table
- Put left controller on the edge of the table left of right controller
- Click button you bound **confirm** to on the controller of your dominant hand
- Refer to in-game book for further instructions

K projektu je přiložené video s ukázkou implementace (verze bez komprese na git repozitáři projektu)

Instrukce na build a použití implementace doprovázející tuto práci

- Implementaci stáhněte buď z archivu projektu po částech nebo z https://gitlab.fel.cvut.cz/langweil-mmp/vr_kresleni_papay
- Měli byste mít Unity projekt ve složce **Unity_2D_VR_painting**
- Teď nainstalujte
  - Unity verzi 2020.3.21f1 (funkčnost s ostatními verzemi NENÍ zaručena) z https://unity3d.com/get-unity/download/archive
  - SteamVR z https://store.steampowered.com/app/250820/SteamVR/
- Otevřete projekt pomocí Unity a spusťte SteamVR
- Otevřete scénu **Puda 2** ve složce Scenes (1)
- Otevřete Window → SteamVR Input → Open binding UI → Edit (aktuální binding) → Calibration (2-4)
- Nastavte bindings (4)



- Klikněte **Play**
- Položte pravý ovladač na střed okraje stolu
- Položte levý ovladač na okraj stolu nalevo od pravého ovladače
- Klikněte tlačítko nastavené na **confirm** na ovladači Vaší dominantní ruky
- Další instrukce jsou v knize v herním světě