UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

Comic Art Editor Development

Author:
Nina BONDAR

Supervisor:
Mykhailo IVANKIV

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2021

Declaration of Authorship

I, Nina BONDAR, declare that this thesis titled, "Comic Art Editor Development" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences
Faculty of Applied Sciences

Bachelor of Science

Comic Art Editor Development

by Nina BONDAR

Abstract

This work is about researching how very specific storytellers convey their messages through a very specific medium - the sequential art of comics. This aims to present a web-based PoC of how organizing the pipeline of comics creation in one place could help professional artists automate their work to larger extent. It targets the features of comics software that have the potential to become really useful and important to the creators - based on research of existing comic series and author interviews. The demo in progress can be found in this github repository.

Acknowledgements

I would like to express my biggest gratitude to my supervisor Mykhailo Ivankiv, without advice and guidance of whom I would surely get lost among the approaches and priorities - thank you for understanding my passions and turning them into a potential project of a lifetime.

I am deeply grateful to the authors of comics who eagerly responded to our requests and broke down their approaches to us.

I am grateful to the Faculty of Applied Sciences at Ukrainian Catholic University, for gathering the best teachers and making it possible to be proud of the faculty one studies at.

Without any doubt, I am the most grateful to my friends and family, who never doubted my ideas.

Finally, I would like to thank myself for not giving up, no matter how unpredictable life was.

Contents

List of Figures

To those who don't give up on their dreams

Chapter 1

Problem statement and domain background

1.1 Motivation

As a newcomer in the industry of comics and drawing in general, I had no idea which software would be the most friendly and powerful simultaneously. Solving a problem like this nowadays is the matter of looking it up in Google search, but the approaches and needs of every artist are different - it makes way more sense to test it yourself. I clearly knew the path of mastering a certain tool is meant to be a bit windy, because the professional tool should be advanced enough to fulfil the variety of artistic needs of artists around the world. Still, once I got to draw digitally, I quickly realised all of the tools are complex and frankly similar. The complexity didn't come out of nowhere, but I truly had zero idea where to apply all of that - and when I wanted to warp, blur or mask the image in an unusual manner, it all was so complex I had to watch several video tutorials before actually using a feature. I realised how not user friendly many editors were. Later, talking to digital artists / comic artists, I heard them confirming that many features are either hidden or so unpopular that no one really has the answers except for figuring it out with manual investigation. Although there's beauty to this freedom and diversity of graphical receptions, as stressful as the work of comic artist is, there's enough complexity on the way to final results. This is why this research is to determine and rank the real pains of comic artist's work and propose an MVP solution of how, based on the principle of Occam's razor, some parts of comic production can be automatised at a cost of an open-source product.

1.1.1 Comics as a medium

There are many types of comics out there. From European visual novels to Korean webtoons and Japanese manga, comics of any type aim to tell a story and convey some message. However, in contrast to how it can be done with writing or cinematography, comics as a medium uses several forms of communication at once, which might bring a deeply satisfying effect to the reader in different ways at once. Little do outsiders of the industry know, however, this industry requires a lot of artist's patience that might be not paid back for at all. Artists and their assistants tend to overtime very often, but there is no certainty in their work to become phenomenally popular, because the competition is very tense as well. The deadlines in the industry, knowing there are weekly zines with comics, are very strict. Thus, the time and comfort become the main factors and pain points for comic artists job. This is why we want to make it easier for them to accomplish their goals in the most effective way.

1.1.2 The role of comics nowadays

No matter how comics get treated in society, the industry of entertainment understood comics are perfect source material for movies and animation long ago. Not only do they already have character designs and panel layouts with angles ready-to-use, they are also fully scripted. Even companies like Warner Bros. take it serious and produce films based on comics - like "Edge of tomorrow"(2014) was based on original 2004 manga "All You Need Is Kill" by Hiroshi Sakurazaka and Takeshi Obata. There are TV series like "Deadly Class" based on American comics as well. There are many other examples: action films, games, comic cons, even theme parks - anything entertainment industry can propose. But not only that. There are examples of comic-based educational books. Linear algebra, physics, calculus, philosophy - anything can be taught in the form of comics, which proves it to be just another medium, just a bit more powerful - it redefines reading and storytelling. According to Forbes, even after a year of COVID-19 pandemics, the industry of comics is healthy, but also faced significant growth of interest towards Asian light novels and manga. One of the named reasons for this growth is the fact editors and authors had to face the breaches ans weaknesses of their work process and fix it quickly to keep up with the market. Nevertheless, growth of interest and publishing optimizations only state that this industry is only getting started.

Chapter 2

Research and assumptions

There are good desktop solutions on the market for drawing in general. People use tools like Adobe Photoshop, Illustrator and Clip Studio Paint. Only the last one has a well-defined pipeline for comics creation and even downloadable 3d models. According to comic artists that we managed to interview in person, tools are only a matter of habit. However, every single one of them told the process of comic art creation requires huge amounts of time. To create something effectively, many artists seek for comfort of perspective guidelines, layout templates, shading tips etc to save time they spend on one work. Not every tool has this, especially on the surface. After talking to the creator of manga "Apple Black", who himself uses Clip Studio Paint, we realized there might or might not be tools for anything one wants to accomplish inside the software they use, however if it comes to advanced and useful features like applying fisheye efect to his perspective drawing, he as an artist eyeballs it instead of using some helpful feature that could exist in the application.

There is more to that. Artists from Ukraine whom we interwied, who are being actively published and take part in international comic cons, have never tried using Clip Studio Paint, but use Photoshop instead. However, after the demonstration of how it would be possible to apply 3d models to repetitive panels drawing in our PoC, the author of "Bane" and other Ukrainian works, confessed that he has a physical reference - mannequin, and uses it's pictures to save time when drawing complex angles. So, not only does he take pictures on a separate device, he also has to upload them to his computer and add to the software he uses for drawing(Photoshop). This is a sequence of actions that could be prevented by having an abiility to model and scale the 3d models like mannequins inside the drawing software. He seemed genuinely sure this 3d feature has a potential because of how much time it saves.

To conclude this part, there is a field of infinite interface simplifications and feature usage analysis in this industry, especially in Ukraine.

2.1 Problems to address based on observations

2.1.1 Inking: what good brush is

No doubt, this is the most required instrument in comics inking. It is a common knowledge comics are traditionally either black-and-white or pay attention to inking and line definition a lot, even if colored. A lot of attention is devoted to stroke width in comics. Stroke width is a vital element for panel drawings, as well as its decor. Intuitively, one can understand the thickness of strokes is controlled by the pressure applied to the instrument. To make the brush really useful, we have to make it react fast to artist's gestures and make sure it is sensitive to the pressure applied to it.

2.1.2 Character design mutations overtime

As a storyline develops, the complexity of character relations and events grows respectively. To start with, it is hard to keep track of all the diverse characters and their designs to full extent. No matter how skilled and experienced the artist is, it might happen to the best of those from the industry because of strict deadlines and possible lack of comfort in the creative work authors are involved in - lots of authors tend to look back through the story to recheck the designs of introduced characters when they use those characters in the story again. Sometimes, even the fact of existence of a character might be forgotten due to complex relations and large numbers of characters that not all authors follow along on a daily basis.

For example, this happened to a character from the story that became popular on a global scale - Attack on Titan's secondary character Rico Brzenska. The author introduced the character with quite an important status and just never came back to using them. It might have been just a secondary character the author didn't want to spend panel space on, however, the majority of other personas in the story got at least a partial character development and recognition, whereas this specific character played an important role in several episodes of the story, then vanished into the thin air.

However, this is not the even the brightest example of what can possibly occur if the author keeps up with the weekly or monthly deadlines and is not able to follow along the story as a whole.

Case "JoJo's Bizarre Adventure: Stone Ocean" by Hirohiko Araki

In order to give one an idea of how immense this work is and how powerful and respected the infamous author is, it is important to mention that the works of Hirohiko Araki appeared in Louvre, on prints of Moschino, Balenciaga, Gucci etc, and the manga by him is being released for more than 30 years. The publishing of Jojo's part 1 started back in 1987. By 2021, the 8th part of it is still in progress. Will all due respect to the author, there were several episodes during the series, when the events, character's gimmicks or characters features got mixed up and confused the audience. An author this big, especially knowing the specifics of this series, can definitely manipulate the characters and plot and stay respected and sold consistently. However, to less famous authors, or to more realistic stories, this might be a very undesirable topping.

In terms of rapid character design changes, special attention should be payed to the sixth part of the series - "JoJo's Bizarre Adventure: Stone Ocean", where the character with fuchsia hair is first introduced as a woman with the clothes of certain color combination - see figure 2.1. After a certain period of time, they are reintroduced as a male character with no gimmick of changing their appearance or sex to that extent. Neither was this fact ever explained afterwards in the story. Thus, fandom first assumed the author just didn't manage to keep up with the diversity of Jojo's universe and simply forgot that this character used to look very different.

However, during one comics conference in Italy, the author explained it by the fact he wanted someone with an androgen appearance in his story. Moreover, the character would be possibly more popular with a masculine design. Still, the change was drastic.

Less dramatic changes of either the clothes designs or character features are seen in many manga and comics series due to authors being really worked out and dedicated to meet the release dates - see one of thousands of examples of a real-life

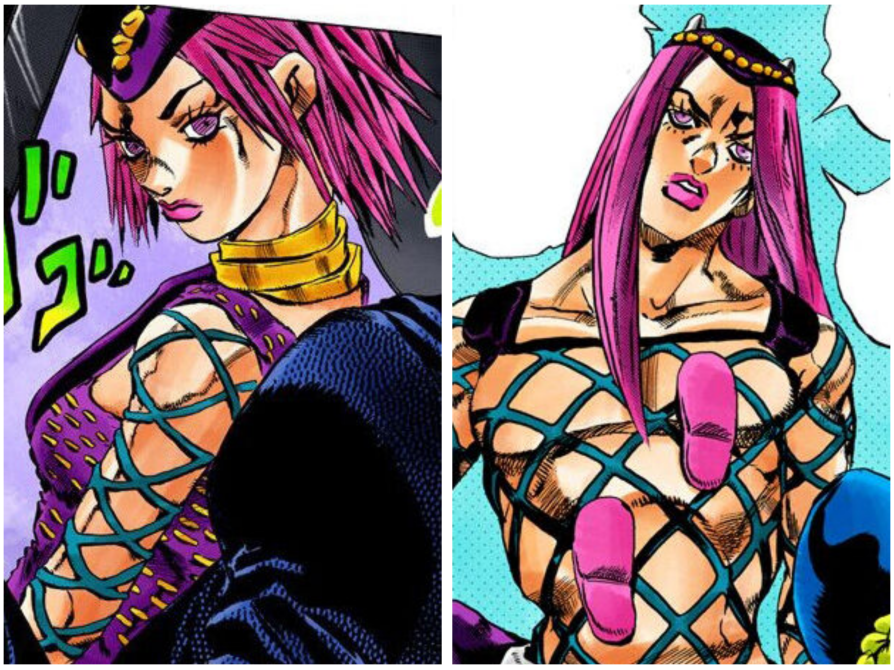

FIGURE 2.1: Frames from manga "JoJo's Bizarre Adventure: Stone Ocean" by Hirohiko Araki. The difference of character's appearance demonstration. To the left: the first character appearance. To the right: The further look of the same character

manga creator schedule. The other possible reason is a messy approach to script writing and organizing the character design layouts.

2.1.3 Setting modeling: building and city plans

Case 'Attack on Titan' by Hajime Isayama

The infamous story-turned-franchise "Attack on Titan" is a manga series that was first published in 2009. The universe is first introduced as an onion-designed system of cities surrounded by the walls(figure 2.2) of nearly 50m height with humans as an almost extinct species living inside them. Behind the walls, on the other side from human cities, there are other species rambling - huge human-eating monsters called titans. They can be relatively short, around 4 meters, or extremely tall - 20 meters and way more.

Now that we know the context of the story and the approximate parameters of the objects, let us closer inspect the frames from volume 1.

As one of the first important events of the story happens - a titan approaches and breaks the wall, the story panels keep repeatedly demonstrating the same city views from various angles to describe the scene in a more interactive way, juggling both scale and perspective (figure 2.3). This case demonstrates a very common approach to how the scene is described and depicted in comics medium. In order to make it more persuading, a lot of the times artists render the setting of the story events carefully, paying attention to at least the proportions of the objects on stage. This ensures a deeper dive-in into the atmosphere of the universe of the story.

In case of this story, the setting of a city behind the walls can be relevantly easy simplified into a 3d model based on cubes - we will get to it later. For now, see figure 3.3.

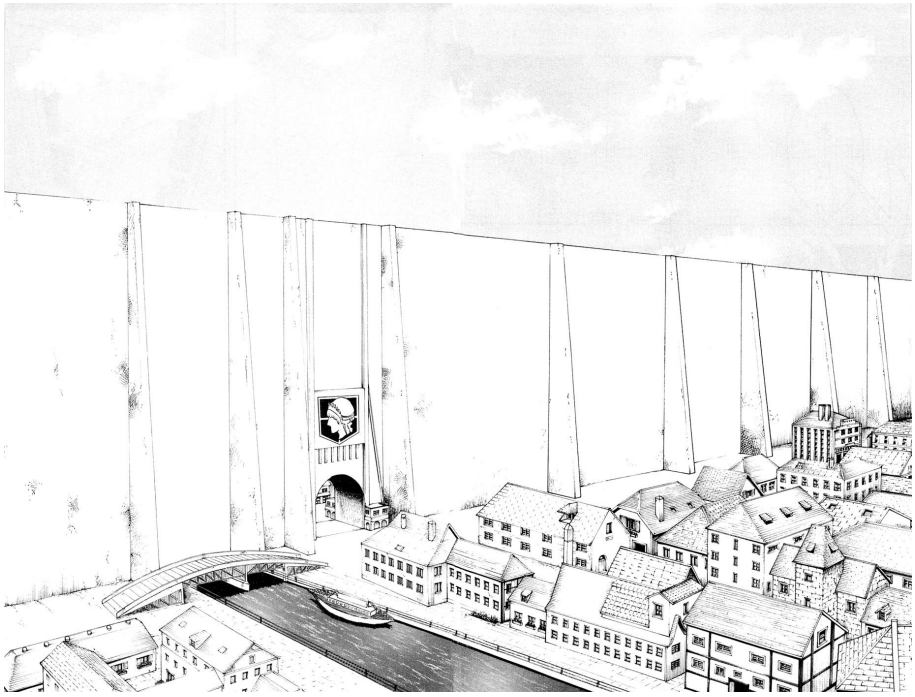

FIGURE 2.2: A page from manga "Attack on Titan" volume 1 by Hajime Isayama. The wall Maria and a city panorama.

Case 'Bane' by Andriy Dankovych

The dark fiction by a Ukrainian artist Andriy Dankovych serves as proof of how events of the story might take place in, out or near the same spot repeatedly throughout the series (figure 2.4). Thus, rendering the details not only requires good memory, but also the abstract thinking and patience.

2.1.4 Angles generation

Often, the objects in comics panels are drawn from a not straight-on perspective under a certain angle to add dynamics and keep attention. Thus, it might be potentially useful for an artist to have a tool for angles generation.

2.1.5 Frames shape: redefining the panel's shape

There still exists an outdated belief (dictated by the style of 1960-70th years' comics) that frames of comics are normally square or at least rectangular. Although it is really convenient to work with a rectangular shape of drawing's canvas, there is no requirement of how parallel the sides of a frame should be. Many modern comic artists neglect the square boundaries of panels - and sometimes even frame boundaries themselves - to add more dynamics and style to their works. See figure 2.3 for rectangular frames and figure 2.5 for differently shaped panels.

2.1.6 Lenses

A less obvious yet extremely spectacular effect artists tend to use for more dramatic effect and/or to fit more details of setting in one panel is a spherical warp - see figure 2.6. According to many artists on the internet, it is tricky and requires good

perspective knowledge. We, for one, can trust the fact it is widely used and creates a n unusual warp effect that catches one's eye.

2.2 Interviews

In this section, let us go through the interviews with professional artists and the insights we got from talking to them. There were three goals we chased during the interviews:

1. Get author's opinion on how useful the list of assumed features is, one by one.
2. Understand their work pipeline.
3. Ask about their current work-related pains.

2.2.1 Odunze Oguguo: the creator of "Apple Black"

Odunze Oguguo is a Nigerian manga artist who is known for creating several published visual novels and a bi-weekly digital English-language magazine that features diverse artists - "Saturday AM".

Software preference: Clip Studio Paint

Has assistants: Yes

Simplified pipeline:

Script writing. At this stage the author mostly uses handwriting. A fair reason to go for this tool set at this stage is that once the script just disappeared because of software glitch. Another reason is that it is faster to open a notebook and/or cross out parts of the script by hand and requires no devices nearby.

Storyboard. This stage is tied to script writing, the author has to go back and forth between the storyboard and script. This is where the final frames sketches and text bulbs are being settled.

Inking. This part is outsourced to his team of assistants and is pretty time-consuming. An insight about it is that although it is convenient to send the sketch via Internet, it requires time to fix the style after the assistants, because everyone has their own inking style. It requires a lot of time to review and fix the style of detailed panels, however it is important for the author to average the style.

Opinion on problems/features:

Character design mutations: The artist isn't prone to forgetting the characters, since he writes the script and analyses it a lot. On the other hand, he frequently checks the previous pages of the chapter to make sure all of tiny details of character's clothes aren't missing. From his words, it can possibly be useful to have a character's design board in the app, but in his case, it is a part of the final review process, so everything will be anyway checked.

Setting modeling: The author sees how it can be useful and helpful, however he himself doesn't use 3d for rendering his drawings and prefers to rely on his skill or reference folders on his computer. This is very person-specific.

Angles generation: From the author's words, he 'doesn't hate the idea', however author sees little sense in this feature, because as a creator he knows how he wants it to look and trusts his vision over computer-generated angles. This is one of the main places where he channels creativity within the pipeline, so it wouldn't be more useful than a folder with references on his computer.

Lenses: The author uses this effect and would be interested to see the implementation of spherical warp. Currently he eyeballs such effects.

New insights:

AI-powered inking. According to the author, it would be extremely interesting to see a well-trained assistant that could work with sketches of different style and clarity'. It would save a lot of time and would be also very financially beneficial for the creator.

Simultaneous editing. Another possible time saver is a feature of simultaneous editing to ensure the communication with the assistants team.

2.2.2 Andriy Diakiv: the creator of "Kurhan"

Andriy is a Ukrainian comic artist that is published by UAComix.

Software preference: Adobe products

Has assistants: No

Simplified pipeline:

Script writing.

Storyboard.By the time of storyboard designing, the author usually knows the script almost by heart.

Inking. This is a long process, because the author works autonomously.

Opinion on problems/features:

Character design mutations: The artist isn't prone to forgetting the characters, doesn't see much use in character board.

Setting modeling: During the demo session, the author confirmed it would be useful to have 3d models and space for editing them, because it would save time to many people. On the other hand, his point is that professional artists who work for years have it all in their head and require no assistance. It might still be useful for professionals to get back in the flow of drawing after a long break. To sum up, the author thinks it is more useful for beginners.

Angles generation: When the author gets to drawing, he already visualises the angles. He doesn't believe in AI-powered assistance too much.

Lenses: The author uses this effect and would be interested to see the implementation of spherical warp. Currently he eyeballs such effects.

New insights:

There has been a visible development of Ukrainian comics since around 2015. A lot of new publishing houses like UAComix, Vovkulaka, FireClaw and others emerged. It is a positive trend and a drastic change since author's school days.

2.2.3 Andriy Dankovych: the creator of "Bane"

Andriy is a Ukrainian comic artist that is published by UAComix and specialises at science fiction and dark fantasy.

Software preference: Adobe products

Has assistants: No

Simplified pipeline:

No script writing.

Storyboard. The author gets the panels straight out of his head, based on the general idea that the story tells about. For science fiction, the author uses hand drawing.

Inking/Coloring. If there is the need to ink, the author does it within Adobe Photoshop/Adobe Illustrator.

Opinion on problems/features:

Character design mutations: The artist isn't prone to forgetting the characters, doesn't see much use in character board.

Setting modeling: During the demo session,from the start, the author confirmed it would be useful to have 3d models and space for editing them, because it would save time to many people. When he needs a certain position of character that is hard to recreate in head, he takes a minified wooden mannequin and models out the position. Then he takes a picture by his phone camera and then uses it to draw over or redraw from the picture like from the reference. This sequence of actions is redundant if the author has a 3d modeling feature. Same works with complex perspectives that have repetitive architectural objects.

Angles generation: When the author gets to drawing, he already visualises the angles. He doesn't believe in AI-powered assistance too much.

Lenses: The author uses this effect and would be interested to see the implementation of spherical warp. Currently he eyeballs such effects.

New insights:

According to the author, same as Andriy Diakov, he believes there has been a visible development of Ukrainian comics since around 2015. It is a positive trend and a drastic change since author's school days (from his words, he started drawing comics at the age of 8). He used to pave his own way through to drawing comics among the first artists of this genre in Ukraine, and he sees fast development of this industry locally - he can no longer follow along the new releases.

FIGURE 2.3: Pages from manga "Attack on Titan" volume 1 by Hajime Isayama. The city is being invaded by titans. A demonstration of different angles for the same scene of action.

FIGURE 2.4: Pages from comics "Bane" by Andriy Dankovych. Repetitive illustrations of the same setting from different angles.

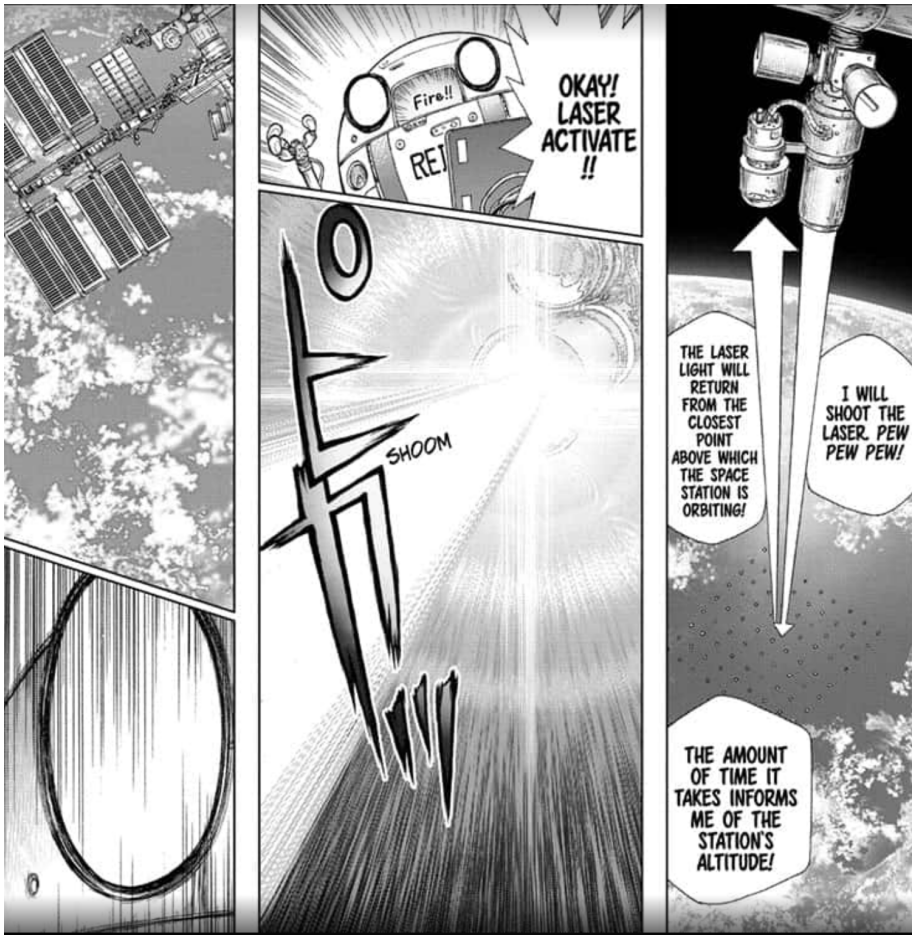

FIGURE 2.5: Frames from a side story "Dr STONE:Reboot Byakuya" illustrated by Boichi. The boundaries of panels are not strictly rectangular.

FIGURE 2.6: Art with character from manga "Naruto" by Kishimoto Masashi. An example of spherical warp.

Chapter 3

Implementation

3.1 Research outcomes, combined

Summing up the research chapter of this work, basing on it, we decided to hit the most needed basic features(like brush drawing or multiple perspectives) first and then gradually add the features our authors gave positive feedback about, moving with an agile way - delivering a ready-to-use MVP on each stage of work.

Thus, in this PoC we mainly focused on:

- Setting model simplification
- Multiple-camera setup: different points of view
- Pen and brush free drawing over the 3d models
- Layout of the panels

It would be then nice to have:

- Advanced in-app modeling
- Mannequins
- Lenses and warp effects
- Different brushes
- Custom lighting

We divided the project into three main parts - see figure 3.1. Currently, we think of the future comics panels as of a stack of 3d layer and multiple 2d layers. Later the layers are going to be saved and layed out on the page to author's liking. The first one is a very important part of the app - the 3d layer, a modeling space, where the either ready-to-use model is imported, or the model is constructed out of Three.js primitive shapes like cube and sphere.
The second part is where the drawing happens: using a pen or a brush, we draw over the 3d model that was previously made. It is tightly tied to the 3d layer with modeling space. At this point, the layer is fixed at the selected coordinates of 3d model and we draw the 2d frame to our liking. Simply saying, we first rotate and zoom the model on a 3d layer, then create a new transparent 2d layer right above these coordinates and draw over. The last part is where the layout is created. For the sake of simplicity and speed of PoC testing, we decided to go for four-paneled page. After the 2d layers of our drawing are ready, we insert them into the predefined panels and mutate the edges the way we decide.

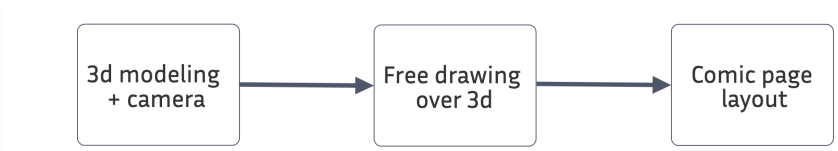

FIGURE 3.1: The main parts of the PoC project

3.2 Technological stack consideration

For the sake of simplicity and fast prototyping, as well as the simplicity of scaling the solution, web implementation with Typescript was chosen. It can be either developed further as a web solution, just like Escalidraw , or turned into a desktop solution, like the famous Clip Studio Paint.

If it wasn't for testing the hypothesis of how artists interact with comics creation pipeline, I would devote more time to speed optimisation of 2d and 3d canvas effects as the user interacts with them. For the sake of flexibility of prototype testing, we decided to develop the application using libraries that simplify the interface of vertex and fragment shaders with a relatively little sacrifice of processor time if started on a local machine.

3.3 WebGL and Three.js

One of the key concepts we'd like to test is working with 3d models bundled with 2d drawing - the simplification of drawing complex angles using customisable 3d models that we can possibly scale, rotate and turn into a reduced setting for the story told. The existing solution for three-dimensional manipulations in web is the WebGL and OpenGL Shading Language(GLSL). It is a high-level web graphics shading language with syntax based on the C programming language, which potentially makes it less comfortable to work with in web paradigm. It is used for creating anything from web games to impressive 3d wow-effects for web pages one can create using shaders. However, for basic multidimensional shapes and operations as well as for the sake of being more flexible with the time of prototype development, it is way more effective to choose one of existing libraries built over GLSL. After a decent amount of libraries research, it became obvious that the optimal solution is Three.js - one of the most popular and powerful libraries that uses a set of abstractions over GLSL. This is how it can be done with Three.js:

```
const geometry = new THREE.BoxGeometry( 1, 1, 1 );
const material = new THREE.MeshBasicMaterial( {color: 0x00ff00} );
const cube = new THREE.Mesh( geometry, material );
```

In the example above, we first create a geometric box - BufferGeometry instance that represents a carcass for the future mesh and texture applied to it. Then, we need some material to cover the carcass: in our case, we're using the simplest material that represents a polygon mesh. After creating a mesh, we finally combine the geometry and the material into the desired shape - a cube.

One more important reason to go for Three.js is the way it simplifies work with 3d space - just the way the creator needs it when thinking of angles: using cameras and stages.

3.4 Camera and stage

Just like in cinema production industry, camera and stage are the terms used to describe respectively the point of view at a given moment and “mise en scene” - everything placed on stage - screen, in our case. Professional modelling software, like Blender or Cinema 4d, has already adopted these concepts to manage scenes in three dimensions. Developed as a solution for gaming and 3d, Three.js borrowed these concepts too.

However, to see anything as an output on the screen and see our shapes and cameras take effect, we need three building blocks of a stage setup.

3.4.1 WebGLRenderer

The first thing is a Three.js WebGLRenderer - a raster workspace with a 3d context that takes in the canvas element from DOM and a context of rendering. Under the hood of it, there is a context one can get from a plain html canvas element by using .getContext('webgl') method in any browser supporting this API. It basically tells the canvas there are OpenGL calculations going to happen.

3.4.2 Scene

Then, we need at least one scene - an object of Three.js representing a visual part of the stage. For now, we might think of scenes as of layers or frames with drawn 3d content.

3.4.3 Camera

Finally, we need a camera to set up a point of view - like an image transmitter to the screen for us to see what's happening on our canvas. There are different types of cameras in Three.js, however we use the PerspectiveCamera for our scene. According to documentation, this projection mode is designed to mimic the way the human eye sees. It is the most common projection mode used for rendering a 3D scene. It takes in several arguments: the first one is FOV. FOV is the extent of the scene that is seen on the display at any given moment - see figure 3.2. Its value is in degrees.

The second argument is the aspect ratio of the stage. It speaks for itself - a ratio of screen width and height, which is a number. The third and fourth ones are near and far. Near and far planes are the numbers representing the position of how near to the camera and how far to the camera the objects rendered on stage can be. Anything positioned outside of the near and far frustums, is not going to be rendered, so will not be visible - see figure 3.2. It is used for rendering optimization and reducing the rendering of redundant, out of sight objects.

In code, it would look like this:

```
// Create a renderer
const renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);
...
// Append a canvas to DOM
document.querySelector("#root").appendChild(renderer.domElement);
```

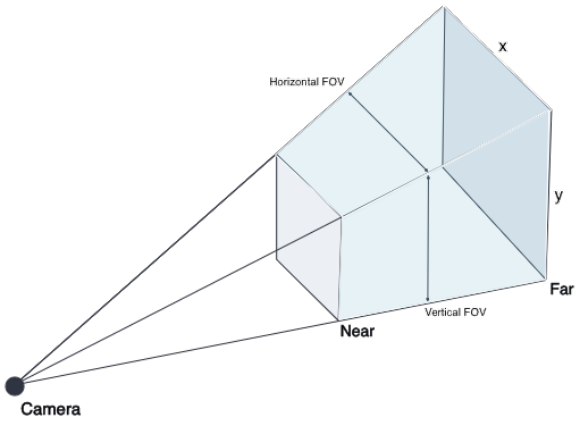

FIGURE 3.2: Three.js rendering space. The illustration of near and far frustum. Anything outside of near and far is not going to be rendered.

```
...
// Create a future scene
const scene = new THREE.Scene();

// Create a perspective camera
const camera = new THREE.PerspectiveCamera(
75,
window.innerWidth / window.innerHeight,
0.1,
1000
);
```

3.4.4 @react-three/fiber

However, the real-life application is built with React in order to simplify the UI development process, as well as to look more elegant. React is well-known for its reusable components approach. Unfortunately, it is not that elegant to write the implicit Three.js-based code(like in the example above) and apply it to the renderer in the component manner - we'd spend too much time even on creating shapes. This is why we decided to go with a powerful package called react-three/fiber. Not only does it allow us to work with complex abstractions like WebGLRenderer of Three.js in the component manner, it also allows the same with the shapes. Now the code of setting up the workspace for 3d is reduced to importing the Canvas component from the package and adding several props to it:

```
import { Canvas } from "@react-three/fiber";
...
return <Canvas
dpr={window.devicePixelRatio}
```

3.4.5 Setting simplification

Lots of art studying resources, like an internet-famous Proko, teach to simplify the composition to a set of plain shapes like cubes before getting into rendering the sophisticated details of it. It would be extremely boring to live in a Minecraft-like world of cubes only, however, a trained artist can normally eyeball fitting an object of a desired shape into a cube. Another reason for plain shapes is that they won't distract the artist from rendering the details over them, because models here only serve a helper purpose. They are not the main entities from artist's perspective. Let us explore how the setting from 2.1.3 can be turned into a simplified model we can work with.

Another example of how it can be useful is the simplification of a castle from 2.1.3.

3.4.6 Multiple-camera setup

As it was described in section 2.1.3, especially when it comes to dynamic moments, artists often use a similar to multiple-camera setup trick, when some essential story moment is shown from several angles simultaneously - see figure 3.3.

FIGURE 3.3: Multiple-camera setup effect with the help of OrbitControls model rotation.

FIGURE 3.4: Our PoC approach to multiple-camera setup implementation. A transparent 2d layer with a scene of titan invading the city over the rotated setting simplification of a high wall and a city behind it from "Attack on Titan" by Hajime Isayama

3.5 Two-dimensional drawing

3.5.1 Layers

Although there exists a possibility to add several cameras on stage, for the sake of both speed optimisation and simplicity, our solution for different angles lies within saving the coordinates of the camera at a given moment and permanently tying it to the new 2d layer that is created over these coordinates. See figure 3.4 for a visual part.

Experiments

Two approaches were tested for this functionality.

react-konva

For layers of 2d drawings, we first decided to find some lightweight library that would possibly have the layers functionality, so that it would be less time for us to integrate this feature. Of all the solutions, we selected the Konva package. Just like react-three/fiber does for Three.js, react-konva aims to provide a convenient interface for working with canvas API in React.
Looking back now, I clearly understand there were several markers of this solution to be totally not what we want from the drawing application. As easy as it is to use the package in components style, firstly, the documentation about free drawing is very short and covers only stateful path drawing. Secondly, according to this documentation, the canvas interactions will get slower if one has too many lines in the state, claiming hundreds of lines already be a lot. For our purposes, though, drawing hundreds and thousands of lines is an inevitable step in final comics page creation. Let us move further to a more performant implementation.

Native HTML5 canvas API

From the perspective of relatively complex drawings, knowing we already have a webgl-contexed canvas in the application, nothing can outbeat the plain canvas and its rich interface. Firstly, it has no middleware to slow the operations down. What is more, rather than giving one a framework, it gives full freedom of actions and a set of good practices and docs that can be found on MDN. Let's briefly go through the implementation of a pen tool.

3.5.2 Pen tool

The concept of free drawing on html canvas element is very simple: it provides an interface in scope of '2d' context for drawing lines and paths on the raster canvas. In order for drawing to feel natural, canvas has to process the emitted browser event like 'mousemove'.

```
document.onmousemove = (event) => {
ctx.beginPath();
ctx.lineTo(event.clientX, event.clientY);
ctx.stroke();
}
```

In order to start and finish the gesture where we set and release the pen, a boolean isDrawing flag is introduced. We start drawing - set isDrawing to true - once the mousedown event is emitted, we release the pen once mouseup event is emitted. This task is a trivial one, no matter the tech stack.

3.5.3 Brush tool

However, never before have we found a robust solution for brush.

There are solutions on the Internet for creating a brush for HTML5 canvas. However, none of those we have found fulfilled the needs of our potential users. There were several reasons for that. As a result, we implemented our own brush instrument that we might have never encountered on the Internet before. Let us dive in the mechanics behind the brush implementation.

Intuitive solution

What's normally done when an engineer wants to implement a brush-stroke effect is repetitive stamp-like drawing of a certain pattern - usually, a circle - on every mousemove event. Browser events' handling is an inevitable part of any brush implementation, because as it was already mentioned in subsection 3.5.2, this is how the movement of a gesture is tracked. However, this is completely not enough to state one has implemented a brush that works like a brush - see figure 3.5.

FIGURE 3.5: The "brush" tool core idea: stamp-like drawing on every mousemove event fired.

The problem of brush "bald patches"

The figure 3.5 introduces us to the problem of canvas not being able to keep track of all of our movements - there are not enough round stamps rendered during large, sweeping gestures. Unless we want to irritate the potential users, it will do no good as a tool and will not handle the long strokes that we are looking for.

Analysis of the article "Exploring canvas drawing techniques" by kangax

In the article "Exploring canvas drawing techniques", the author leads us through very different approaches and does the job of summarizing the majority of approaches one might come up with. From stamping to "furry" brushes and shadowed lines, we

were looking through all of the solutions gathered there, but none of those could do the sophisticated job of creating what's expected from a brush tool: monotonous strokes of arbitrary length, and, as it was mentioned in section 2.1.1, - handling pressure.

But let us first see the most robust solution for our case of all those proposed. The stateful brush implementation was the closest to what we wanted. It solved the baldness problem, as well as didn't look like a try to hide the poor implementation of stroke continuity - see figure 3.6.

FIGURE 3.6: The mono-width continuous brush with baldness problem solved.

In order to achieve this effect, the author saved each point coordinates to points array that were hit by the tool on mousemove. Then, with the help of .lineTo() method of 2d HTML5 canvas, they fixed the bald patches on each move. The code is pretty simple.

Pressure and pointer events

The article by kangax was written back in 2013, so there used to be no mainstream way of tracking the input device or the pressure applied to it, to my belief - only the working draft of pointer events was introduced in 2012. No matter the reason, nowadays there is a common API for detecting pressure - event.pressure of pointermove event. According to this specification, many input devices were neglected in favor of mouse as a standard input device. Since Mouse events were introduced, the second step in history was the implementation of Touch events. However, to ensure a stable unified solution for tracking touch, click and move events, Pointer events were introduced. Luckily for us, because the majority of comic artists nowadays use specialized tablets with stylus for drawing. Such physical tools generate various pressure coefficients, depending on the physical pressure detected. Now the core idea would look like this: we set a predefined radius of a single stamp we render on pointermove event, but multiply it by a coefficient of pressure applied.

```
const radius = 5
const startAngle = 0
const endAngle = 2*Math.PI

canvasElement.onpointermove = (event) => {
ctx.beginPath();
```

```
ctx.arc(event.clientX,event.clientY,radius * event.pressure,
startAngle, endAngle)
...
}
```

Important: to see the effects of pressure applied, one needs to test the event.pressure property on other than laptop touchpad/touch screen digitizer.

3.5.4 Our approach

Applying the idea of joining points

Coming back to the approach demonstrated at figure 3.6, let us apply the idea of joining the previous stamp's coordinates to the pointer events we decided to handle and see what happens.

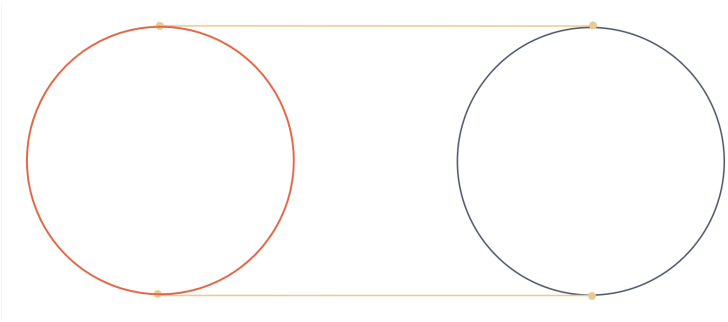

FIGURE 3.7: Joined stamps. The yellow lines are lines we would like to have between each stamp to create the continuity effect and fight the bald spots.

To start with, we use stamp-based approach to control the radius affected by pressure. Simply joining the centers of stamps would be not enough. The idea of joining the dots then has to connect our stamps themselves - see figure 3.7. From the previous subsection, we already know the radius of our stamps varies. This is an important notice before we finally get to the variant of a consistent brush stamp with beautiful brush-like strokes we want to achieve.

Tangents of stamps with different radius

We cannot be sure the radius of our stamps is similar. It is not natural for brush strokes to be consistently wide or thin. In case with similar-sized stamps, where the radius is equal, here we cannot simply use the parallel lines as a connection. The real situation rather looks like the ones in figure 3.8.

From what we know from trigonometry and unit circle, the point of where the tangent touches the circle stamp can be found with an angle and the coordinates that are equal to (cos , sin) - see figure 3.9.

For drawing the point on the arch of our stamp within the canvas 2d context, let us see how applying the cos and sin formulas actually works:

```
function drawPoint(angle,distance,label){
    var x = center_x + radius * Math.cos(-angle*Math.PI/180) * distance;
    var y = center_y + radius * Math.sin(-angle*Math.PI/180) * distance;
    ctx.beginPath();
    ctx.arc(x, y, point_size, 0, 2 * Math.PI);
```

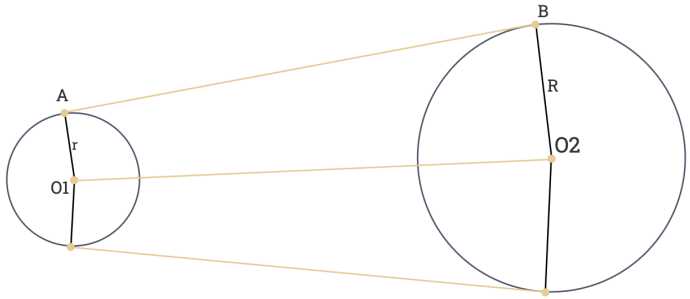

FIGURE 3.8: Joined stamps. The yellow lines are lines we would like to have between each stamp to create the continuity effect.

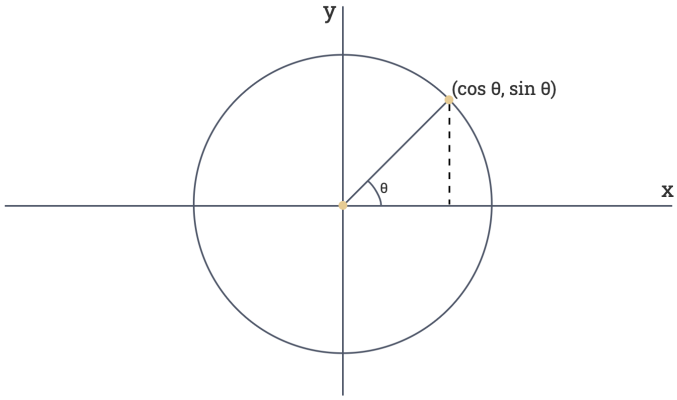

FIGURE 3.9: The unit circle.

```
}
```

Now, the way we find the length of a desirable tangent is how we normally calculate the distance between the dots:

```
const findLineLength = (point1, point2) => {
  const x = point2.x - point1.x
  const y = point2.y - point1.y
  return Math.sqrt(Math.pow(x, 2) + Math.pow(y,2))
}

const tangentLength = Math.round(findLineLength({x1, y1}, {x2,y2}))
```

The demo of this part can be found on my codepen. For the desirable stamp that we achieved, see figure 3.10.

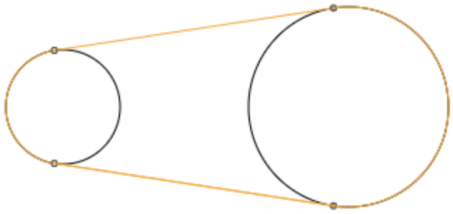

FIGURE 3.10: By joining the tangents, we achieved a new joined stamp of a beautiful shape.

The complete demo of how the final form of the brush works can be found on this github account. There is also a figure 3.11 of the desired brush strokes.

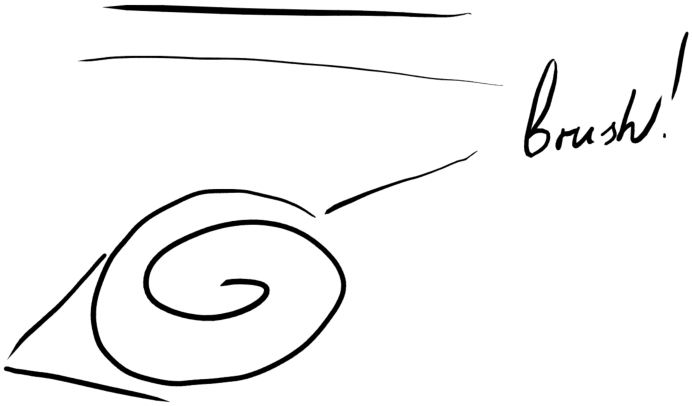

FIGURE 3.11: The final brush strokes after all trigonometric manipulations.

3.6 Panel layout

The final part of our PoC is a very important part we've touched in section 2.1.5.

3.6.1 Experiments

Two approaches were tried out during this part of implementation.

SVG Masks

When we were thinking about how to mutate the shape of already drawn rectangular canvas art, svg was the first obvious area where masking of images existed. Modern browsers support svg graphics, so there is absolutely no need in side packages to work with svg.

The concept behind it is very simple: we have a future comics page that has mutable rectangular slots. For PoC, we went for four square-shaped frames that could be further mutated into some arbitrary rectangulars.

This approach consists of two key components: svg <image/> tag and <mask/> element put over the image - see figure 3.12.

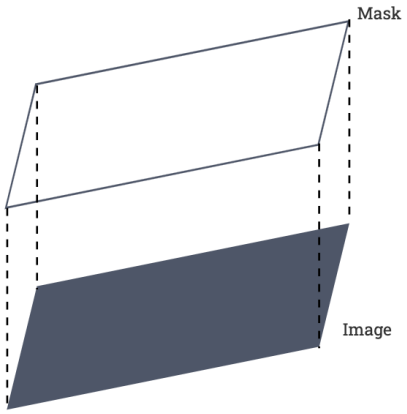

FIGURE 3.12: The order of layers in svg mask approach: the mask goes on top of the image.

To define a mask in svg, we need to create a path of the desired shape inside the mask. As the shape is square, path generation is relatively simple: we need four points in order to achieve this shape; top left(tl), top right(tr), bottom right(br) and bottom left(bl). Thus, the React component of a Mask looks the following way:

```
const Mask = ({ tl, tr, bl, br, maskId }) => {
  const path = `M${tl[0]} ${tl[1]} L${tr[0]} ${tr[1]}
  L${br[0]} ${br[1]} L${bl[0]} ${bl[1]} Z`;

  return (
    <mask id={maskId}>
      <rect x="0" y="0" width="300" height="300" fill="black" />
      <path d={path} fill="white" />
    </mask>
  );
};
```

To create an image out of a snapshot of the art on canvas, there exists a method .toDataURL() in 2d context that is useful to save the 2d art to further put it inside a panel in our case.

Then, to ensure the intuitive interface of changing the frame of the panel, I've created draggable vertices for each of the masks using another svg tag - <circle/>.

The demo of this approach can be seen in this figure 3.13.

FIGURE 3.13: The implementation of svg masks as the frames for the image. Notice they have the possibility to be placed over one another

Canvas frames with PIXI.js

The reason this approach appeared is that it might be not enough for an artist to have just frames on the page, no matter how advanced they are. There might be a case when they need to break the fourth wall. There might also be a case when they want to draw some additional details over the page (sound effects, dialogue bulbs, decorative elements etc). PIXI.js is a wonderful library for working with textures, simple shaders and 2d wow-effects as well as 2d games. The logics would be almost the same: we place a mask on top of an image. With the help of @inlet/react-pixi package, we can do it the following way:

```
const Frame = ({ coordinates }) => {
...
return (
<>
  <Graphics draw={path} ref={maskRef} preventRedraw={false} />
  <Sprite
    image={imageLink}
    height={400}
    width={400}
    anchor={0.3}
    mask={maskRef.current}
    ref={spriteRef}
  />
</>
);
```

Chapter 4

Conclusion

4.1 Outcomes

As it was discussed with the artists during the demo sessions, the functionality we've already came up with is useful and can definitely ease the pain of abstract thinking and drawing the complex angles, which is a huge win and luck. Of course, not everyone will find it useful on a daily basis to have 3d models for scenes, especially if their art style is flat and doesn't require visual perspective. However, as long as professional artists see even potential ways to apply the functionality, our PoC proves itself to have sense. What is more, we've inspected the pipeline of the professional comics creation and used the Occam's razor in order to implement a reduced yet truly useful tool for repetitive setting's drawing. We've also managed to come up with a workaround for a web-based brush that relies on trigonometric calculations and seems to be not that known on the web.

The work is not done yet. However, with what I managed to show the authors and discover about the industry, the goal of understanding and optimizing a tool for comics creation pipeline has been at least partially achieved.

4.2 Further steps

As it was described in 3.1, there are many features we would like to add to this project, if it turns into a product. After interviewing several Ukrainian artists, I feel like there is sense to introduce them to how useful and fun the tool can be.

Chapter 5

Bibliography

Comics definition. https://en.wikipedia.org/wiki/Comics

McCloud,Scott. Making Comics: Storytelling Secrets of Comics, Manga, and Graphic Novels. William Morrow Paperbacks, 2006. pp. 1-272.

Cohn, Neil (2013). The Visual Language of Comics: Introduction to the Structure and Cognition of Sequential Images. London: Bloomsbury. ISBN 978-1-4411-8145-9

*Article "Comic Art Market Still Healthy After A Year Of Covid"*https://www.forbes.com/sites/robsalkowit art-market-still-healthy-after-a-year-of-covid/?sh=86011a366171

Attack on Titan. https://en.wikipedia.org/wiki/Attack$_o n_T itan$

Hirohiko Araki. https://en.wikipedia.org/wiki/Hirohiko$_A raki$

A day in the life of a Japanese manga creator https://www.youtube.com/watch?v=t3rKrTehORY

Araki, Hirohiko.JoJo's Bizarre Adventure: Stone Ocean. Shueisha, 2000. ISBN in Japan - 978-4-08-873027-1

Isayama, Hajime. Attack on Titan. Kodansha, 2009. Comixology https://www.comixology.com/Attack-on-Titan-Vol-1/digital-comic/269338

Article "Exploring canvas drawing techniques" by kangax. http://perfectionkills.com/exploring-canvas-drawing-techniques/

*Working draft of Pointer events.*https://www.w3.org/TR/2012/WD-pointerevents-20121211/

Pointer events documentation. https://developer.mozilla.org/en-US/docs/Web/API/Pointer$_e vents$

THREE.JS https://threejs.org/

Konva.js documentation. https://konvajs.org/

PIXI.js documentation. https://www.pixijs.com/