DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Web Application for Layout Decomposition of Printing-Based Microfabrication

Hui Cheng

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Web Application for Layout Decomposition of Printing-Based Microfabrication

Web-Anwendung für die Layout-Zerlegung bei der druckbasierten Mikrofabrikation

Author:	Hui Cheng
Supervisor:	Prof. Dr.-Ing. Ulf Schlichtmann
Advisor:	M.Sc. Meng Lian
Submission Date:	17. October 2022

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 17. October 2022 Hui Cheng

Abstract

Over the past decade, printing-based manufacturing has gained increasing interest as an additive manufacturing method for electronic devices. Decomposing the design layouts of printing-based microfabrication is the first and essential step to optimize the manufacturing cycle time. This project aims to build a web application that decomposes printing-based microfabrication designs. This application corrects microfabrication design prototypes in the form of SVG, PDF and DXF files, and then decomposes them into the fewest amount of rectangles by adapting traditional computational geometry algorithms. The web application adopts a decoupled architecture is deployed with Docker. The front-end program is developed with the React framework, and the back-end program is written in Python and uses the FastAPI framework. The outcome of this project is a minimum viable product for a web application that allows users to upload their own print-based microfabrication designs, preview the decomposition results online, and download the corresponding resulting vector graphics files.

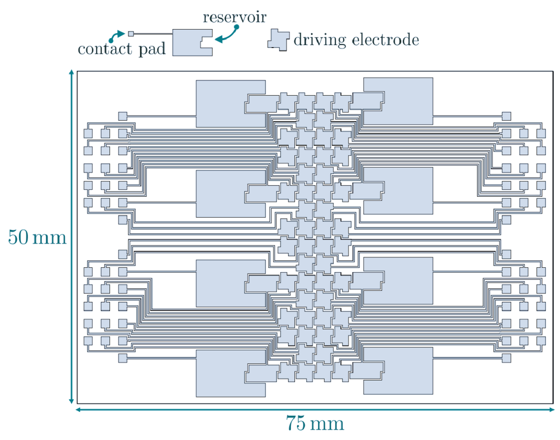
Contents

1 Introduction

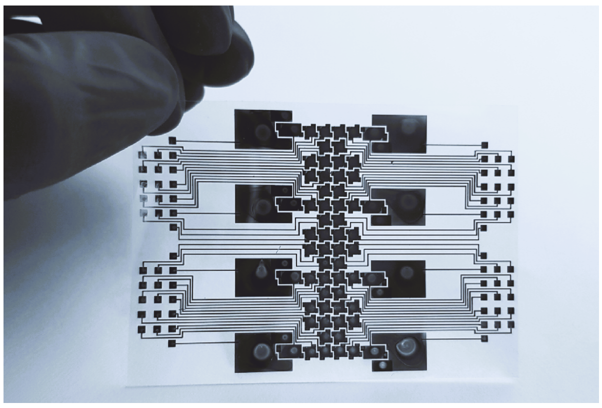
Since the invention of inkjet printers in the twentieth century, inkjet printing has become one of the most widely used printing technologies in the publishing and graphics industries [1]. This technology has recently attracted increasing attention as a manufacturing technique for depositing functional materials. Printing low-viscosity solvents containing nanometals as inks on silicon or polymer substrates is a promising approach for the cost-effective mass production of electronic components [2]. Direct inkjet printing of nanoscale microfabrication is already possible [3]. Figure 1.1 shows a sample design and the corresponding printing-based microfabrication product. However, the undesired device behaviour caused by accidental ink merging and redistribution is still a major problem for the microelectronic printing manufacturing process. For example, when printing closely spaced thin leads and electrodes simultaneously, separated patterns might be connected at some points due to ink merging, resulting in short-cuts and dysfunctionality. To reduce printing errors and shorten the manufacturing cycle time, the state-of-the-art approach decomposes the whole design, arranges the modified objects in different layers, and finally prints and dries these objects in batches [4]. A rational decomposition is essential for optimising the entire manufacturing cycle, but little attention is given to this topic. Therefore, our project is aimed to fill the gap and create a web application for printing-based microfabrication design decomposition.

The program applies traditional computational geometry algorithms, e.g. vector cross-product calculation, vertex concavity determination, line segment intersection determination, and polygon partition, to decompose the printing-based microfabrication layouts delivered as vector graphics into a minimum number of rectangles.

Besides the core function, layout decomposition, this web application should allow users to upload images, browse and download the decomposition results simply through a dynamic graphic interface in web browsers. There are many different approaches to accomplish the desired features. After considering analysis and comparison, we developed the front-end and back-end programs separately, using React as the front-end framework and Python as the back-end programming language. We also use containerised deployment techniques to provide the application with a secure and stable environment.



(a) Design



(b) Production

Figure 1.1: Printing-Based Microfabrication

2 Background

In addition to printing-based microfabrication, other key contextual concepts and definitions are covered in this chapter.

2.1 Rectilinear Polygon

A rectilinear polygon or an orthogonal polygon is a polygon containing only horizontal or vertical edges and the interior angle at each vertex of such polygon is either 90° or 270° [5]. Rectilinear polygons are not only wildly used in application areas such as computer graphics, imaging processing, and Very Large Scale Integration (VLSI) layout analysis, but are also closely related to printing-based microfabrication, the design in Figure 1.1a is composed of a series of rectilinear polygons [5].

Polygon decomposition is a common problem in computational geometry as it is often used in various applications, e.g., pattern recognition [5]. Some decomposition algorithms already exist based on the following theory

"A rectilinear polygon can be minimally partitioned into $N - L - H + 1$ rectangles, where N is the number of reflex vertices, H is the number of holes and L is the maximum number of non-intersecting chords that can be drawn either horizontally or vertically between reflex vertices [6]."

Ohtsuki has developed an $O(n^{5/2})$ time algorithm for this problem, Imai and Asano have an $O(n^{3/2} \log n$ algorithm, and Liou et al. have proposed an $O(n \log \log n)$ algorithm [7][8][9].

2.2 Vector Graphic

Printing-based microfabrication designs are typically stored and transferred as vector graphics. Vector graphics, a form of computer graphics, is usually constructed by combining multiple geometric objects such as lines, rectangles, and curves created from commands or mathematical statements [10]. The most significant advantage of vector graphics over raster images in formats such as Joint Photographic Experts Group (JPEG), Portable Network Graphics (PNG), or Graphics Interchange Format (GIF) is their outstanding scalability, allowing them to be scaled up or down without losing image quality or producing large files, keeping the geometry smooth and neat [11]. The vector graphics drawn with different software have different image file formats, among which Scalable Vector Graphics (SVG), Portable Document Format (PDF), and Drawing Exchange Format (DXF) are commonly used.

2.2.1 SVG

SVG is a language or an image file format based on Extensible Markup Language (XML) for describing two-dimensional vector graphics. It was developed as an open standard format by World Wide Web Consortium (W3C) [12]. Basic geometries such as rectangles, circles, and ellipses can be defined in SVG using the corresponding Document Object Model (DOM) elements [13]. For example, `<rect x="10" y="10" height="30" width="50"/>` represents a rectangle with a length of 30px and a width of 50px, and its top-left vertex is at $(10, 10)$. An alternative representation is to use the `<path>` element. `<path d="M10 10 h50 v30 h-50 z">` denotes the same rectangle, where `M` indicates a "move to" operation, `h` or `v` stand for a relatively positioned horizontal or vertical line segment, and `z` means to close the path.

SVG images can be easily embedded into HTML files or other XML languages, because of its XML base. Cascading Style Sheets (CSS) which describes how HTML elements should be displayed, is also applicable for SVG images. SVG is widely used to share graphics content on the Internet, popular browsers such as Google Chrome, Firefox, and IE all support SVG image rendering. There are various ways to create or edit SVG images, from commercial or open-source graphic editors such as CorelDRAW and Inkscape to online web applications, e.g., Method Draw Vector Editor; people with basic XML knowledge can even write SVG images in plain text.

2.2.2 PDF

PDF is built to exchange documents across platforms and evolved from the Camelot project proposed by Adobe co-founder Dr. John Warnock in 1991 [14]. After nearly three decades of continuous development, PDF documents are not only able to display flat text and pictures but also can contain interactive elements such as forms, as well as functions like encryption [15]. Vector graphics in PDF files are composed of paths, and a path object is an arbitrary shape consisting of straight lines, rectangles, and cubic Bézier curves on the Cartesian plane. The expression of the paths in PDF uses the reverse polish notation [16], where the path construction operator is placed behind the coordinate operand [17]. One PDF expression for a rectangle starting from $(10, 10)$ with a length and a width 30 and 50 respectively is `10 10 50 30 re`. The art style of geometries is defined by other operators.

Many browsers also have built-in PDF-viewer, and common PDF desktop software is mostly PDF-readers or multifunctional editors that allow drawing simple shapes on a PDF file; there are few professional graphics applications. However, due to the wide application of PDF documents, many professional graphic programs, e.g., CorelDraw, Inkscape, and AutoCAD, all support exporting their images to PDF files.

2.2.3 DXF

DXF is a tagged Computer-aided Design (CAD) data representation [18]. Autodesk developed it for the data exchange between AutoCAD and other software [19]. A DXF file may contain all or some of the following sections: HEADER, CLASSES, BLOCKS, TABLES, OBJECT, ENTITIES, and THUMBNAILIMAGE [18]. Among them, the ENTITIES section is the most important because it contains the geometric information of the graphic objects. For example, group code `100` is the subclass marker, `100 AcDbLine` means that the following data describes a line segment. Group codes `10(11)`, `20(21)`, and `30(31)` correspond to X, Y and Z values of line segment's start point (endpoint). `100 AcDbLine 10 35 20 50 30 0 11 40 21 65 31 0` constitutes the minimal geometric information describing a line segment starting at $(35, 50, 0)$ and ending at $(40, 65, 0)$.

Compared with SVG and PDF, DXF is less prevalent in areas other than technical drawing. Almost all browsers do not support DXF file rendering; DXF files must be drawn and viewed using CAD software, e.g., AutoCAD.

2.3 Web Application

A web application is a program that is invoked with a web browser over the Internet [20]. Unlike traditional desktop applications, users do not need to download and install the client application on the local computer; they only need to open the browser to visit the corresponding website to use the functions of a web application.

When the Internet and the World Wide Web first entered the public consciousness in the 1990s, web applications were primarily collections of interlinked static HTML files containing text, images, or videos. Through nearly two decades of development, new network technologies, languages, frameworks, and methodologies have brought more dynamics and possibilities for web applications [20].

Nowadays, there are two common web application architectures, monolithic and decoupled. A single application handles everything from data conversion to web page rendering in monolithic architectures. While in decoupled architecture, the front-end for user interactions and the back-end for data and logic processing serve as two independent applications and communicate with each other via Application Programming Interface (API) [21].

HyperText Markup Language (HTML), Cascading Style Sheets (CSS), and JavaScript are the cornerstones of the front end. HTML is like the skeleton of a web page, CSS is its beautiful skin, JavaScript is the muscle that makes it move. It is enough to make simple websites just using them. However, in the last decade, a variety of front-end frameworks, e.g., React, Vue.js, and Angular, have been created and enhanced with new features. These frameworks not only make web applications more functional and aesthetic, but also make the front-end development more convenient and accessible.

The back-end environment has more possibilities and options than the front-end. Python, JAVA, PHP, and Ruby can all be used to develop back-end programs. On top of these different languages, various frameworks are designed to match the development of front-end technologies and enhance user requirements. Django, Flask, and FastAPI are all Python-based back-end frameworks. Laravel, Zend, and Codeigniter are great PHP-based backedn frameworks.

For front and back-end communication, a prevailing style nowadays is REST API. The concept of Representational State Transfer (REST) was first introduced by Roy Fielding as part of his doctor dissertation in 2000 [22]. REST is not a protocol or standard that must be followed, but a network-based software architecture style. A web service needs to meet the following restrictions to be defined as an authentic REST API: uniform interface, client–server, stateless, cacheable, layered system, and code on demand (optional). In practice it can be interpreted straightforwardly as: using Uniform Resource Identifier (URI) to locate resources in the system and implementing CRUD (create, retrieve, update, delete) operations on them via Hypertext Transfer Protocol (HTTP) methods, e.g., POST, GET, and DELETE.

3 Analysis

This chapter uses critical analysis methods from software engineering to articulate the problem that this application needs to solve and the requirements it should meet.

3.1 Problem Statement

An essential purpose of developing this project is to extract the patterns from the draft of the printing-based microfabrication design containing only rectilinear polygons and split them into a minimum number of rectangles. However, these drafts are not always perfect; they may contain slashes that are not parallel to the horizontal or vertical axis. The prototype may also have "small steps", as shown in Figure 3.1. These steps may be part of the design, or errors generated during the drawing process, which can lead to incorrect partition results. Users should have the option to keep these small steps or smooth them out. Furthermore, as mentioned in chapter 2, the design of printing-based microfabrication can be present in different file formats depending on the software used by the author. Previewing vector images in different formats often requires the installation of different programs as well. There lacks a unified platform to provide previews of PDF, SVG, and DXF images. So a web application that can parse and present the contents of all three kinds of files would also be advantageous.

3.2 Requirement Analysis

Software engineering usually has two types of requirements, functional and non-functional. Functional requirements describe what the system should do. They state the system's reactions to specific user inputs and behaviors in particular situations. Non-functional requirements are not directly related to the system's specific functions but constrain the whole system [23]. Functionally, this web application should:

- allow the user to upload PDF, SVG, and DXF image files from their devices,
- allow the user to adjust the smooth parameter,
- correct the slash into a straight line that is parallel to the horizontal or vertical axis,
- smooth the "step error" according to the parameter selected by the user,
- extract the rectilinear polygon information from the user-uploaded vector graphics,
- decompose rectilinear polygons into a minimum number of rectangles,

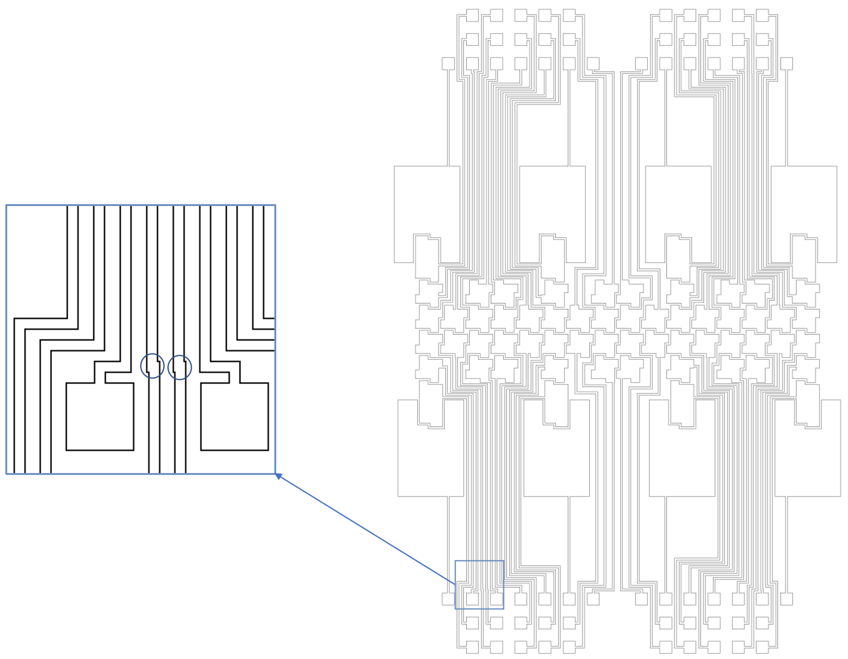


Figure 3.1: Step Error

- display the original image content,
- display the decomposition result image and the details of each rectangle,
- let the user download decomposition results in PDF, SVG, and DXF forms.

As a demo of the web application, even if a comprehensive analysis cannot be done due to the lack of information such as hardware parameters, the following core non-functional requirements still need to be taken into account during the development process:

Usability The website should be intuitive, with each functional component named accordingly, so that users can use it without additional learning.

Adaptability The development environment is was Windows while the operating system of general web application server is Linux, so the system needs to have the ability to adapt to different operating systems.

Integrability The system needs to provide interfaces to functions and components for future work

Safety The whole application should run in a stable and secure environment.

4 Design and Implementation

Software technology is developing rapidly and there are countless possibilities and options to design and implement a web application. Architecturally, although it is far less painful to develop and deploy a monolithic structure for this small project with only one developer, separating the front-end and back-end is still preferred due to the scalability, modularity, flexibility, and extensibility of the decoupled architecture [24]. This chapter will introduce the design and implementation of the front-end and back-end programs respectively with help of the sequence diagram shown in Figure 4.1 and Figure 4.2.

4.1 Front-end

The front-end is taking care of user interaction functions such as uploading and downloading PDF, SVG, and DXF image files, previewing raw images and decomposition results, and changing smoothing parameters.

4.1.1 Frameworks

React and TailwindCSS frameworks are used in this project and significantly improve development efficiency.

React

React is the most popular front-end framework nowadays [25]. It is developed and maintained by Facebook engineers, and thanks to its large community, React is also receiving contributions from developers around the world [26]. Not only Web applications of Facebook, but also other famous websites such as Netflix use React [27]. Instead of the "real" Document Object Model (DOM), an API for HTML documents, React uses the concept of the virtual DOM and binds this term to React elements to optimize DOM operations through reconciliation methods and thus improve the performance of the React applications [28]. React constructs the website with reusable components, which makes the code less redundant and can be easily extended [29]. Moreover, benefiting from a growing and active community, there are numerous available open-source React extension libraries.

TailwindCSS

TailwindCSS is a utility-first CSS framework that embeds CSS styles into the class attributes of markups through shortcuts, thus avoiding switching between HTML and CSS files during

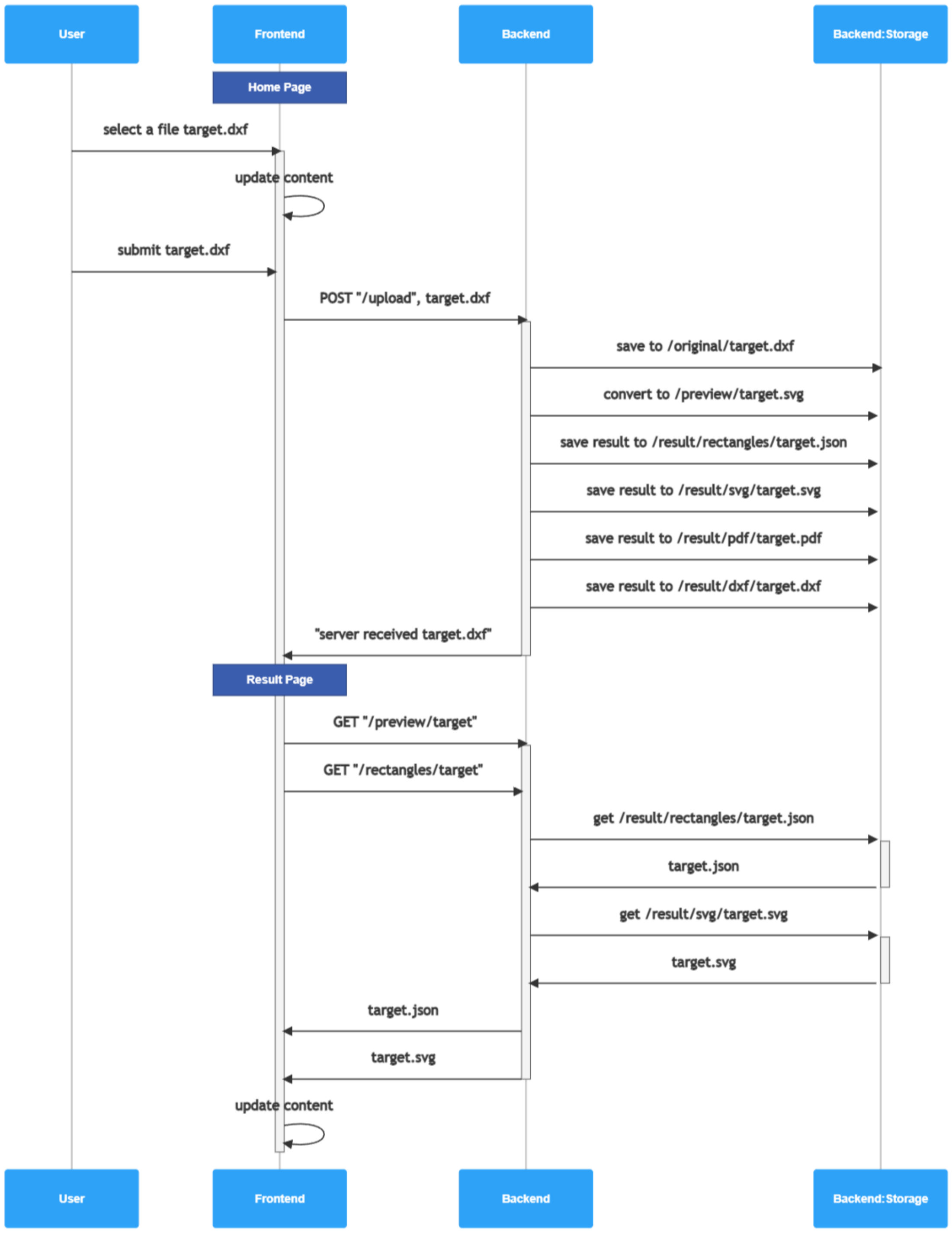

User
Frontend
Backend
Backend:Storage
Home Page
select a file target.dxf
update content
submit target.dxf
POST "/upload", target.dxf
save to /original/target.dxf
convert to /preview/target.svg
save result to /result/rectangles/target.json
save result to /result/svg/target.svg
save result to /result/pdf/target.pdf
save result to /result/dxf/target.dxf
"server received target.dxf"
Result Page
GET "/preview/target"
GET "/rectangles/target"
get /result/rectangles/target.json
target.json
get /result/svg/target.svg
target.svg
target.json
target.svg
update content
User
Frontend
Backend
Backend:Storage

Figure 4.1: Sequence Diagram - Part 1

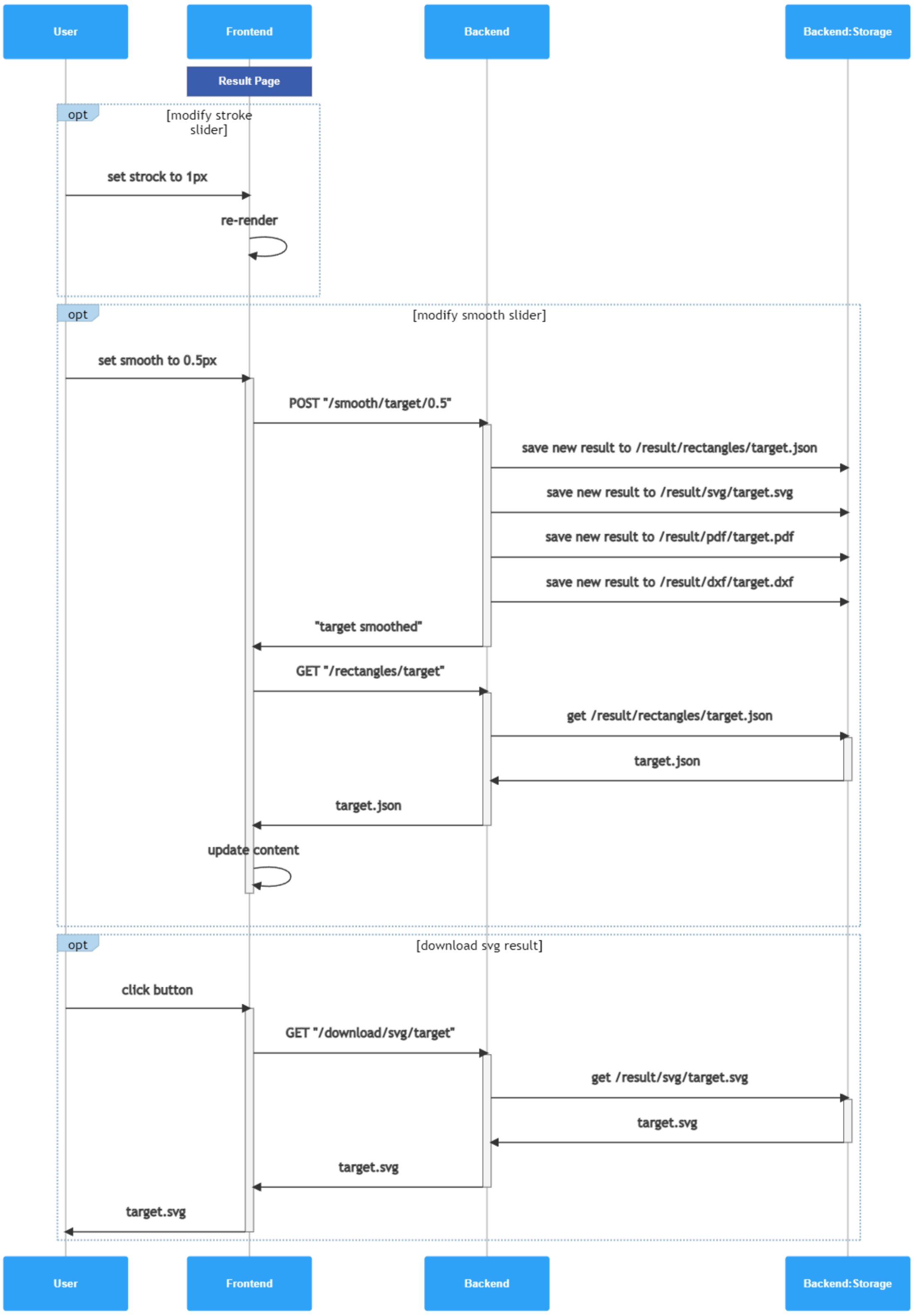


Figure 4.2: Sequence Diagram - Part 2

development.

4.1.2 User Interface

Users interact with the application through mainly two web pages: the home page and the decomposition result page. A 404 not found page is also set up to alert users that the page they are visiting does not exist or that the link provided is incorrect.

Home Page

The homepage contains three components:

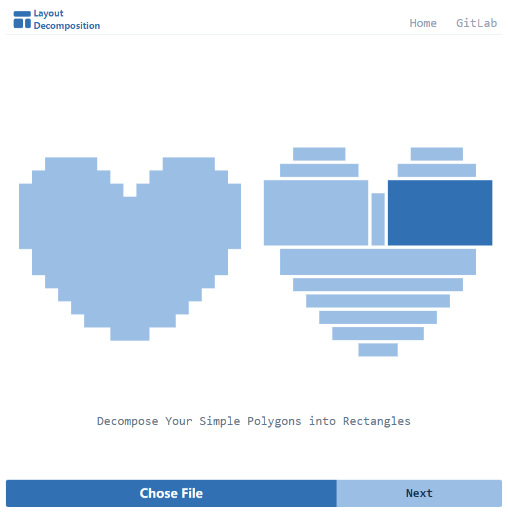


Figure 4.3: Home Page

Navigation Bar The leftmost part of the navigation bar is a simple logo composed of text and an icon. Text "Layout Decomposition" indicates the main functionality of this Web application, and the icon also has imagery of decomposition. When the mouse hovers over the text on the right, the color of the corresponding text will change from gray to blue. "Home" can be clicked to access the home page. When "GitLab" is clicked, a new tab showing the LRZ GitLab repository for this project will be opened in the browser.

Banner The banner provides an intuitive explanation of the website's functionality - a rectangular decomposition of simple rectilinear polygons. The pixel art heart image on the left is a simple rectilinear polygon. On the right is its decomposition result containing a minimum number of rectangles whose color turns dark blue when the mouse hovers over them.

File Uploader The user can select the file to be uploaded by clicking the button on the left. The file type is limited to SVG, PDF or DXF, other types of files will not appear in the options. After a file is selected, the text of the button will change from "Chose File" to the corresponding file name. The selected file will not be sent to the back-end for subsequent processing until the "Next" button is clicked. The file transmission uses the package Ky, a tiny but elegant HTTP client for providing React with an interface to request and response operations [30]. The following code shows how Ky posts the selected files to the backend:

```
const fileUploadHandler = async () => {
    const fd = new FormData();
    fd.append('file_name', filename);
    fd.append('image', selectedFile);
    await ky.post(ROUTES.BACKEND_BASE_URL + "/upload", {body: fd})
        .text()
        .then(res => {
            console.log(res);
            navigate(`/result/${filename.substring(0,filename.indexOf('.'))}`);
        })
        .catch(err => {
            console.log(err);
        });
}
```

Result Page

The URL of decomposition result pages has the format `/result/:filename`, where `filename` is the name of the file uploaded by the user in the last step, indicating that this page shows the decomposition result of the specific image. At the top of the Result Page is again the navigation bar, with the same functions as the one on Home Page. The body is divided into two columns, a total of four parts. The left part is a big decomposition result display area, and on the right, from top to bottom, are a thumbnail of the original image, sliders, and download buttons. The specific functions of each component are as follows:

Result Preview The images uploaded by the user are split into a minimum number of non-overlapping rectangles and displayed in this area. When the user hovers over a rectangle, the color of the rectangle changes and an information box appears nearby showing the length and width. The back-end will store the coordinates, the length and the width of the rectangles of the layout in a JSON (JavaScript Object Notation) File. When switching to the result page, the front-end will request the corresponding JSON file from the back-end, parse the data, reassemble it into a new SVG image containing only `<rect>` tags, and display the SVG image in this area. The dynamic response to mouse hover action is implemented using react-tooltip [31], which has a ready-made floating message box component `<ReactTooltip>`. Instead of writing complex functions from scratch, the desired functionality could be achieved by

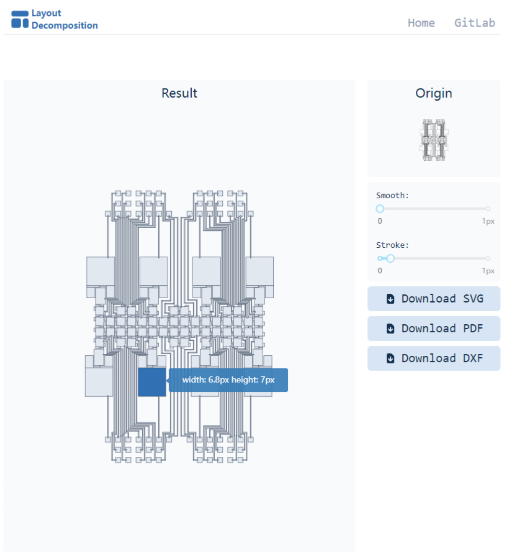


Figure 4.4: Result Page

simply appending the `<ReactTooltip>` component, then adding the corresponding `data-for` attribute in `<rect>` tags, and finally adding the data-tip attribute to declare the information that needs to be displayed, i.e., the length and width of the rectangle. The code is shown as below:

```
<div className="py-3␣w-full␣mx-auto">
  <svg xmlns="http://www.w3.org/2000/svg"
       className="my-auto␣mx-auto"
       viewBox={viewbox}>
    <g className="fill-gray-200␣stroke-gray-600"
       stroke-width={strokewidth} >
      {rects.map(item =>
        (<rect
          className="hover:fill-tum-blue-300␣focus:fill-tum-blue-300"
          data-tip={‘width: ${item[2]}px height: ${item[3]}px‘}
          data-for="svgTooltip"
          x={item[0]} y={item[1]}
          width={item[2]} height={item[3]}/>))}
    </g>
  </svg>
  <ReactTooltip id="svgTooltip" place="right" type="info"/>
</div>
```

Origin Thumbnail This area shows a thumbnail of the original image, which can be clicked to zoom in and pop up to the middle of the screen. Images uploaded by users will be firstly converted to SVG format and stored in the back-end. This SVG preview image will be requested while fetching the JSON file, and then wrapped in a `<Zoom>` component, another external component provided by react-medium-image-zoom [32].

Smooth Slider This component is used to adjust the smooth threshold parameter, and thus to eliminate the "step error". Users can adjust the value between 0-1px by dragging the small light blue circle horizontally. The implementation of this component is based on the `<Slider>` provided by rc-slider [33], and the `smooth` function is bound to the `onChange` event. In the smooth function, the changed values will be sent to the back-end, which will re-correct the image and generate new decomposition result. After that, the front-end will request the JSON file containing the rectangle information again and re-render the result preview area.

Stroke Slider This slider is used to control the stroke width of the rectangles in the result preview. Users can adjust the value between 0-1px to keep the boundaries of the rectangles clear and non-overlapping. Similarly, this component reuses the `<Slider>`, but instead of binding the `smooth` function, it binds the `setStrokeWidth`, as the following code shows:

```
<Slider min={0} max={1} marks={{0: 0, 1: "1px"}} step={0.1}
    onChange={smooth} defaultValue={0}/>
<Slider min={0} max={1} marks={{0: 0, 1: "1px"}} step={0.1}
    onChange={setStrokeWidth} defaultValue={0.1}/>
```

Download Buttons Users can download SVG, PDF and DXF files respectively by clicking on download buttons: `Download SVG`, `Download PDF` and `Download DXF`. These three buttons use the same `<DownloadButton>` component, which consists a file download icon and the button name. The icon uses the `<DocumentDownloadIcon>` directly from Heroicons, a React package providing sets of free high-quality SVG icons [34]. Clicking the button will trigger the download process, which is coded as:

```
async function download(fileType) {
    await ky.get(BACKEND_BASE_URL+`/download/${fileType}/${filename}`)
        .blob()
        .then(res => {
            const url = window.URL.createObjectURL(new Blob([res]));
            const link = document.createElement('a');
            link.target = "_blank";
            link.href = url;
            link.download = `${filename}_res.${fileType}`
            document.body.appendChild(link);
            link.click();
            document.body.removeChild(link);})}
```

The front-end will first request the file in the target format, and store the returned image data temporarily in a BLOB (Binary large object). The data in the BLOB can then be downloaded to the users local machines, by creating an invisible <a> tag and simulating a click on it.

404 Not Found Page

404 is a common HTTP status code, which is returned when the server cannot find the requested resource or the URL accessed by the user is not recognized. Many websites have a 404 error page. It is like a signpost telling the user that you are lost. Our 404 page retains the navigation bar at the top, which can help the user get back to the home page. The 404 right in the middle is made up of a series of rectangles, alluding to the rectangular decomposition functionality of this Web application.

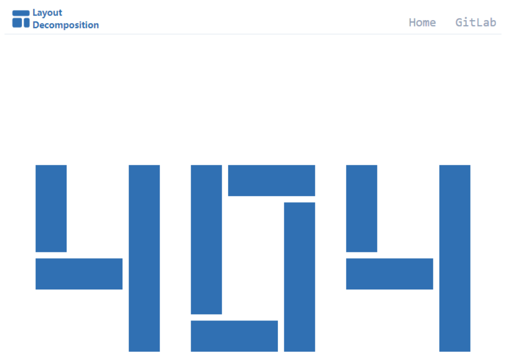


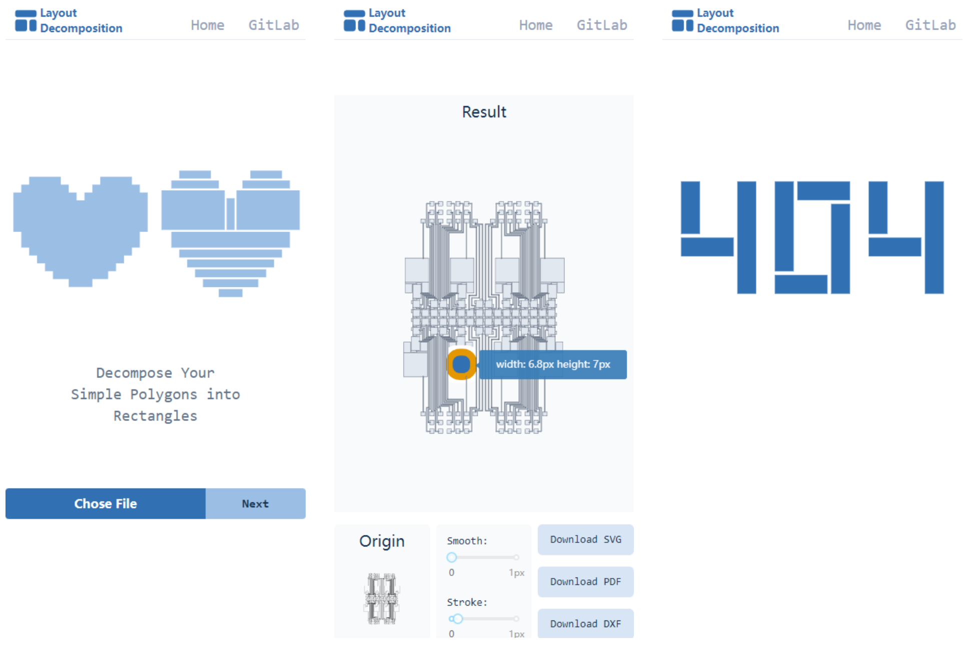
Figure 4.5: 404 Not Found Page

Responsive Web design

The responsive design has been added to make the web page look good on different size of browsers. In TailwindCSS, it can be achieved simply by prefixing utilities with screen width breakpoints. The following line of sample code indicates that the width of the image is 16 by default, 32 on a medium screen, and 48 on a large screen.

```
<img class="w-16␣md:w-32␣lg:w-48" src="...">
```

Figure 4.6 shows how the pages look on a mobile screen. Images and text are downsized on small screens. The spacing between components is also reduced accordingly. Besides, the layout on the results page also changes. The vertically aligned elements on the right are placed horizontally to the bottom of the screen. The icon of the download buttons turns to be invisible.



(a) Home Page (b) Result Page (c) 404 Page

Figure 4.6: Pages on Slim Screens

4.2 Back-end

The back-end program is responsible for functions like file format conversion, layout extraction, auto-correction, and layout decomposition. In addition, the back-end requires the proper API to answer the requests from the front-end. The file storage system is also part of the back-end. The `images` folder under the back-end project holds all original images, preview images, corresponding JSON files of the decomposition result, and the result images in SVG, PDF and DXF formats.

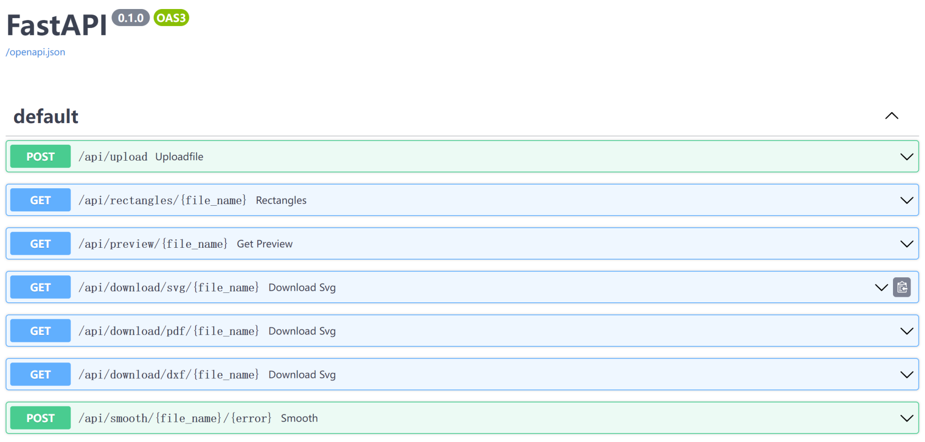
4.2.1 FastAPI

In less than four years since Sebastián Ramírez first committed it on November 24, 2018, FastAPI already has 42.7k starts on GitHub. As its name suggests, it is one of the fastest Python web frameworks. Using FastAPI can improve back-end performance and reduce human bugs and code redundancy, thus increasing system stability and robustness. The excellent editor support provides a comfortable development experience, and the interactive API documentation makes testing much easier [35]. Figure 4.7 shows the back-end interface documentation. There are seven interfaces in total, two for the `POST` methods and five for the `GET` methods. FastAPI is also easy to learn and use. Implementing the function of requesting a preview image takes only the following four lines of code:

```
@app.get("/api/preview/{file_name}")
```

```python
def get_preview(file_name: str):
    imgpath = f"images/preview/{file_name}.svg"
    return FileResponse(imgpath)
```

The `file_name` in the routing path is passed as a parameter to the `get_preview` function, which gets the corresponding image path in the storage system and then returns the image data to front-end.


FastAPI 0.1.0 OAS3
/openapi.json

default

```
POST /api/upload Uploadfile
GET /api/rectangles/{file_name} Rectangles
GET /api/preview/{file_name} Get Preview
GET /api/download/svg/{file_name} Download Svg
GET /api/download/pdf/{file_name} Download Svg
GET /api/download/dxf/{file_name} Download Svg
POST /api/smooth/{file_name}/{error} Smooth
```

Figure 4.7: Interactive API Documentation

4.2.2 File Conversion

As introduced in the chapter 2, SVG format images can be easily integrated into the HTML page with more readable and editable source code so that SVG will be used as the primary file format of our back-end procedure. PDF and DXF format files need to be converted into SVG files before the subsequent correction, smooth and decomposition. The following software or python packages are used in the file conversion:

Inkscape Inkscape is an open source cross-platform vector graphics editor using standardized SVG as its native document format that supports different operating systems, including Windows, Linux and macOS. Inkscape has a GUI where users can draw and modify their designs. Operations such as resizing and file format conversion can also be executed from the terminal via corresponding commands [36]. Python programs can call the subprocess to run the Inkscape commands, thus incorporating the functionality of Inkscape into the backend program and enabling conversion between different vector graphics.

ezdxf A Python package to read, modify, and write DXF files [37].

svgwrite A Python package to create new SVG drawings [38].

scour An SVG cleaner written in Python that reduces the size of a SVG file by optimizing its structure and removing unnecessary data [39].

Figure 4.8 illustrates the back-end file format conversion process. PDF format files can be converted directly through the Inkscape command. However, the SVG files converted by Inkscape will contain a lot of tags that do not contain any meaningful geometric information and may fail the rendering of the image on the web page. Similarly, user-uploaded SVG files may also have non-essential tags. Therefore, the SVG images from these two sources need to be cleaned and optimized using scour. There is no suitable external software to convert DXF files to SVG files using simple commands. Thus the conversion of DXF files is performed by reading the data through ezdxf and then writing the image information to the SVG file using svgwrite. The SVG files obtained this way contain only essential image information, and there is no need to optimize it with scour.

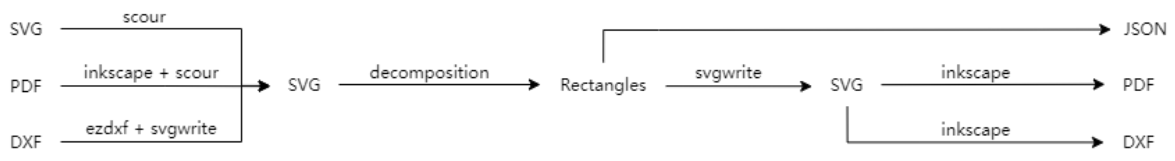


Figure 4.8: File Conversions

Decomposed rectangles are written to the SVG file by svgwrite. Later, the final PDF and DXF result images are converted from the generated SVG file with Inkscape.

4.2.3 Layout Extraction

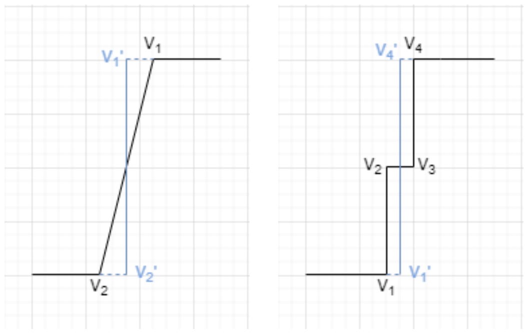
The converted or optimized SVG file should contain a number of Path tags, each representing a closed path with a rectilinear polygon shape. The vertex coordinates of these closed paths could be extracted using the Python package svgpathtools [40]. Sometimes there are several consecutive edges located on the same line. Such overlap will not cause a different appearance of the rendered image but can lead to erroneous results in subsequent decomposition. So it is necessary to reformat vertices, and for consecutive vertices lying on the same line, only the vertices that formed the longest line segment will be kept, and the rest should be removed.

4.2.4 Auto Correction

Auto-correction consists of two steps, the default slash correction and smooth if the threshold is given.

Slash Correction

As defined in chapter 2, rectilinear polygon contains only horizontal or vertical edges, so before starting the decomposition process, correcting the slash segment of the input graph into a horizontal or vertical edge is necessary. The basic idea of slash correction is to replace



(a) Slash Correction (b) Smooth

Figure 4.9: Auto Correction

the slash segment with its horizontal or vertical projection that passes through its midpoint. As Figure 4.9a shows, the black line is the original slash, V_1 and V_2 are its two end vertices, M is its midpoint, the blue line is the new edge, and V_1' and V_2' will overwrite the coordinates.

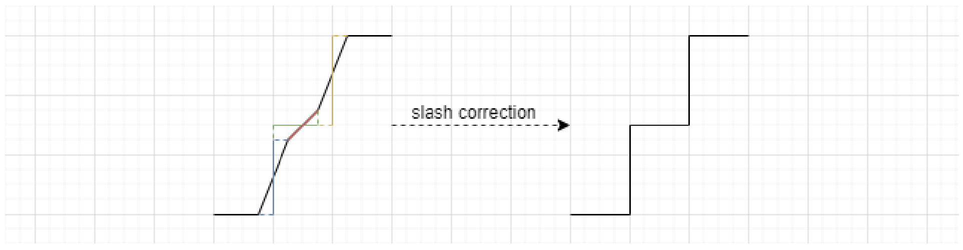
Smooth

In order to smooth out the "step error", edges with lengths shorter than the user-selected smoothing threshold parameter should be removed by deleting the two endpoints that make up the micro-line segment and then correcting the new slash formed by the adjacent vertices. In Figure 4.9b, suppose the edges formed by V_2 and V_3 are shorter than the threshold, then the blue line is perpendicular to it and passes over its midpoint, and V_1' and V_4' are the projections of V_1 and V_4, respectively. After smoothing, V_2 and V_3 should be removed from the list of vertices, while the coordinates of V_1 and V_4 should be updated.

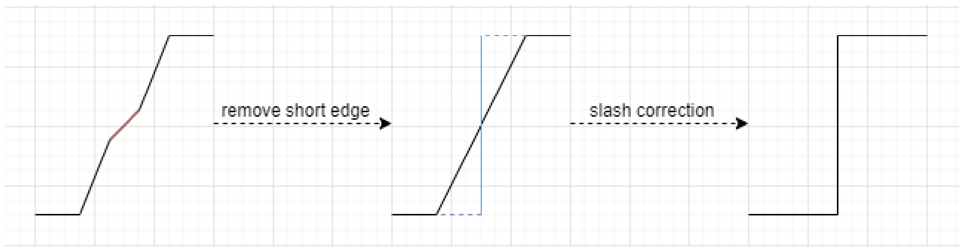
The reason for performing smooth after slash correction instead of removing all the micro-line segments at the beginning and then performing slash correction is that the former is sometimes milder than the latter. For example, the sample pattern shown in Figure 4.10 contains three consecutive slash segments. Assuming that the smooth threshold is 3, if we first delete the edges shorter than this length and then perform slash correction uniformly, as Figure 4.10a shows, these three consecutive segments eventually become one line segment. When we perform 3 successive slash corrections at first, as in Figure 4.10b, the result still contains three line segments, each longer than the threshold. It is also possible to get the same result as the other approach yields by appropriately increasing the smooth threshold value.

4.2.5 Rectilinear Polygon Decomposition

The decomposition algorithm used in this project is based on a transcription of Ohtsuki's $O(n^{5/2})$ algorithm [7] by San-Yuan Wu in [41]. It uses the following steps to draw a series of line segments along which the rectilinear polygon will be cut into a minimum number of rectangles:



(a)



(b)

Figure 4.10: Two Order of Correction

Step 1 Find a maximum matching of the bipartite graph that represents the chords.

Step 2 Find a maximum independent set of where no two chords intersect using the maximum matching of Step 1.

Step 3 Draw the chords in the maximum independent set from step 2 and deconstruct the polygon into smaller rectilinear polygons.

Step 4 From each of the concave vertices from which a chord was not drawn in step 3 draw a maximum length vertical line that is wholly within the smaller rectilinear polygon created in step 3 that contains this vertex.

In step 3 we may draw horizontal lines or vertical ones. The rectangles enclosed by the polygon edges, chords of step 2, and lines of step 3 form a minimal non-overlapping rectangular decomposition of the rectilinear polygon. To get more details of rectangles, e.g. vertex coordinates, width, and height, further sorts and matches are needed. The adapted approach is to apply the divide-and-conquer principle to this partition algorithm as follows:

Step 1 Find a maximum matching of the bipartite graph that represents the chords using Hungarian algorithm.

Step 2 Find a maximum independent set of where no two chords intersect using the maximum matching of step 1.

Step 3 The chords in this maximum independent set partitions the polygon into smaller rectilinear polygons and these sub-polygons are without chords.

Step 4 For concave vertices in sub-polygons draw a vertical and a horizontal line segment that meets the boundary and is wholly within the smaller rectilinear polygon, choose the shorter line segment and this line divides the rectilinear polygon into either a rectangle or another smaller rectilinear polygon without chords. Repeat this step until all partitions are rectangles.

Convexity of Vertices

The line segments forming the contour of the polygon are edges. The endpoints of the edges are the vertices. In rectilinear polygons, a vertex is concave if its interior angle is 270°, and is convex if the interior angle is 90°. The concavity of the vertices can be determined based on vector product. Three consecutive vertices $V_1(x_1, y_1), V_2(x_2, y_2), V_3(x_3, y_3)$ construct two vectors as the following equations:

$$\vec{a} = \overrightarrow{V_2V_1} = (x_1 - x_2, y_1 - y_2)$$
$$\vec{b} = \overrightarrow{V_2V_3} = (x_3 - x_2, y_3 - y_2)$$

The vector product is calculated as the following:

$$\vec{c} = \vec{a} \times \vec{b} = (x_1 - x_2)(y_3 - y_2) - (x_3 - x_2)(y_1 - y_2)$$

The angle range between vector $\vec{a}$ and $\vec{b}$ can indicate the interior angle of vertex V_2 and be judged by the positive-negative nature of vector product. When the vertices are aligned continuously clockwise, V_2 is convex if $\vec{c}$ is smaller than 0, otherwise concave. Conversely, when the vertices are aligned counterclockwise, then v2 is concave if $\vec{c}$ is smaller then 0, otherwise convex. To avoid the ambiguity of the vector product result caused by different orders, a convex vertex, i.e., the vertex with the largest coordinates, needs to be found first. The vector product of the vectors formed by this convex vertex and its preceding and following vertices is calculated as the indicated direction. A vertex is also convex if its vector product of the vectors formed with this vertex and its preceding and following vertices has same direction as indicated, otherwise concave [42].

Algorithm 1: Cross Product

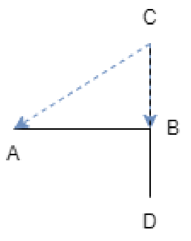
```
input  : Vertices of two vectors
output : Cross product of two vectors
1 Function cross_product(a_start, a_end, b_start, b_end):
2     return $(a\_end.x - a\_start.x) * (b\_end.y - b\_start.y) - (b\_end.x - b\_start.x) * (a\_end.y - a\_start.y)$
```

Intersection of Line Segments

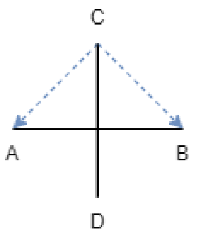
In Euclidean geometry, the intersection of two line segments can be the empty set, a point, or another line segment. When the intersection is a point, two line segments intersect. The

Algorithm 2: Concave Vertices

```
input  : A set of vertex $V$ that forms a rectilinear polygon
output : A set of concave vertex $C$
Function get_concave_vertices(V):
    $C \leftarrow \emptyset$
    $max\_x \leftarrow max(\{v.x : v \in V\})$
    $snd \leftarrow last(sort(\{v : v.x = max\_x, v \in V\}))$
    $fst \leftarrow V[(snd.index - 1 + |V|)\%|V|]$
    $trd \leftarrow V[(snd.index + 1)\%|V|]$
    $direction \leftarrow cross\_product(snd, fst, snd, trd)$
    for $i \leftarrow 0$ to $|V|$ do
        $fst \leftarrow V[(i - 1 + |V|)\%|V|]$
        $snd \leftarrow V[i]$
        $trd \leftarrow V[(i + 1)\%|V|]$
        if $direction * cross\_product(snd, fst, snd, trd) < 0$ then
            $C \leftarrow C \cup \{snd\}$
    return $C$
```



(a) Meet



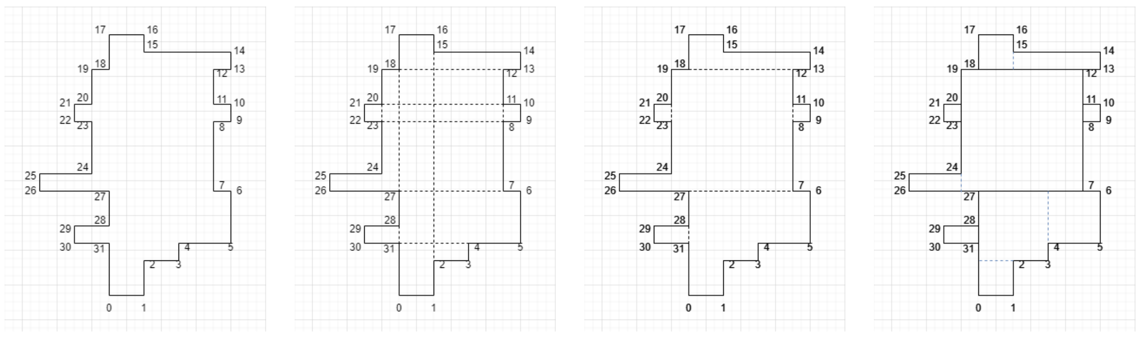
(b) Cross

Figure 4.11: Intersection

algorithmic determination of the geometric relationship of the line segments again involves the utilization of the vector product. In Figure 4.13 two endpoints of line segment CD are located on two different sides of the line AB, $(\overrightarrow{AB} \times \overrightarrow{AC}) * (\overrightarrow{AB} \times \overrightarrow{AD}) < 0$. In Figure 4.11a, the vertex B is located directly on CD, where $(\overrightarrow{CD} \times \overrightarrow{CB}) = 0$, thus $(\overrightarrow{CD} \times \overrightarrow{CA}) * (\overrightarrow{CD} \times \overrightarrow{CB}) = 0$, the line segment AB meets the line segment CD at B; In Figure 4.11b line segment AB crosses line segment CD, $(\overrightarrow{CD} \times \overrightarrow{CA}) * (\overrightarrow{CD} \times \overrightarrow{CB}) > 0$.

Algorithm 3: Intersection

```
input  : Two line segment $AB$ and $CD$
output : True or False
1 Function cross(AB, CD):
2     return $cross\_product(A,B,A,C) * cross\_product(A,B,A,D) \leq 0$ and
        $cross\_product(C,D,C,A) * cross\_product(C,D,C,B) \leq 0$
3 Function intersect(AB, CD):
4     return $cross\_product(A,B,A,C) * cross\_product(A,B,A,D) \leq 0$ and
        $cross\_product(A,B,A,C) + cross\_product(A,B,A,D) \neq 0$ and
        $cross\_product(C,D,C,A) * cross\_product(C,D,C,B) \leq 0$ and
        $cross\_product(C,D,C,A) + cross\_product(C,D,C,B) \neq 0$
```



(a) (b) (c) (d)

Figure 4.12: Decomposition of Sample Rectilinear Polygon

Chord

A chord is a horizontal or vertical line segment that formed by two different concave vertices and lies wholly within the polygon, not crossing any edge. Line segment (p_{13}, p_{18}) is a horizontal chord of rectilinear polygon in Figure 4.12a, and (p_{20}, p_{23}) is one of vertical chords. The algorithm 4 to obtain the horizontal(vertical) chords in RP is motivated by the bucket sort, where first all concave vertices with the same y(x) coordinates are placed in the same bucket, and then the concave vertices in the bucket are arranged according to their x(y) coordinates. If the line segment formed by two adjacent concave vertices in the same bucket is not an edge of the polygon and does not cross any edge, then the line segment is a horizontal(vertical) chord of this rectilinear polygon. The horizontal chord set of the rectilinear polygon in Figure 4.12a are $\{(p_{13}, p_{18}), (p_{11}, p_{20}), (p_8, p_{23}), (p_7, p_{27}), (p_4, p_{31})\}$, and the set of vertical chords are $\{(p_{20}, p_{23}), (p_{18}, p_{27}), (p_{28}, p_{31}), (p_2, p_{15}), (p_8, p_{11})\}$

The intersection between the horizontal and vertical chords can be presented as a bipartite graph, where each vertex represents a chord and each edge means two chords intersect.Figure 4.13a presents the intersection bipartite graph of chords of polygon in Figure 4.12a

Algorithm 4: Horizontal Chords

```
input  : A set of concave vertices $C$; A set of edges $E$
output: A set of horizontal chords $H$
1  Function get_horizontal_chords(C, V):
2      $H \leftarrow \emptyset$
3      forall $v \in C$ do
4          sort $v$ to buckets so that, $\forall bucket \in buckets, \forall m, n \in bucket, m.y = n.y$
5      forall $bucket \in buckets$ do
6          sort bucket according to x coordinates
7          for $i \leftarrow 0$ to $|bucket|$ do
8              $h \leftarrow Line(bucket[i], bucket[i+1])$
9              if $h \notin E$ and $\{e : e \in E, cross(e,h)\} = \emptyset$ then
10                 $H \leftarrow H \cup \{h\}$
11     return $H$
```

Maximum Matching

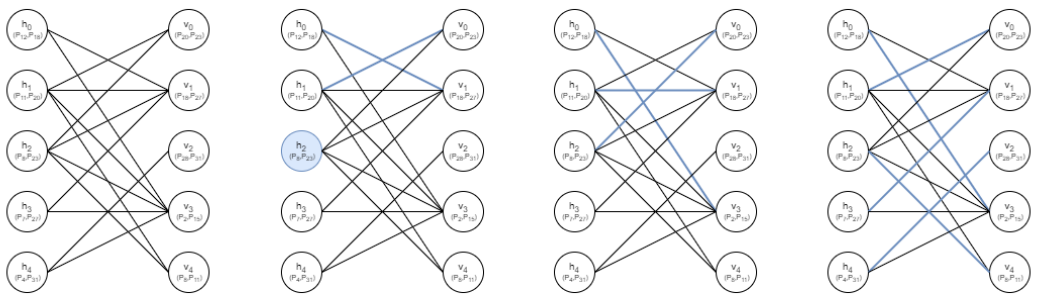
Let $G = (H \cup U, E)$ be a bipartite graph indicating the intersection of the chords, f a maximum matching of G, $D(f) \subseteq V$ the domain of map f, and $R(f) \subseteq H$ the range of f. One of the common algorithms to find the maximum matching is Hungarian or Kuhn-Munkres algorithm, originally proposed by H. W. Kuhn in 1955 [43] and refined by J. Munkres in 1957 [44]. It uses augmented paths to find the maximum matching in $O(n^3)$ time for a bipartite graph with a partition of size n. An augmenting path starts at an unmatched vertex, traverses unmatched edge, matched edge, and unmatched edge alternatively, and ends at another unmatched vertex of the bipartite graph. By swapping the relationship between matched and unmatched edges in the augmenting path, a new matching is added. The Hungarian algorithm iteratively finds augmented paths for all vertices in a partition of a bipartite graph until none are found. For bipartite graph Figure 4.13a, h_0 and h_1 encounter their unmatched vertex easily at first attempt. When h_2 tries to match with v_0, it finds that v_0 is already matched with h_1, and the augmenting path continues to extend to v_0, until it meets a unmatched vertex v_3. Reversing the matching relationship by eliminating the original match between v_0 and h_1, as well as v_1 and h_0, and then marking the edges h_2v_1, h_1v_1, and h_0v_3 with blue lines, a new temporary matching is then illustrated in Figure 4.13c. After same searches for h_3 and h_4, a final maximum matching for Figure 4.13a is shown in Figure 4.13d.

Maximum Independent Set

Let $G = (H \cup V, E)$ be a bipartite graph. Let f be a maximum matching of G, and f is a bijection map, $|D(f)| = |R(f)|$. Let F be the set of free vertices that are not matched, $F = (H \cup V) - (D(f) \cup R(f))$. A maximum independent vertex set S of graph G contains all the free vertices and one single vertex of each match. The algorithm 6 is used for constructing

Algorithm 5: Maximum Matching

```
input  : A bipartite graph $G = (H \cup V, E)$
output : A maximum matching $f : D(f) \to R(f)$,
Function has_augmenting_path(h, U):
    forall $v \in V$ do
        if $v \notin U$ and $(h, v) \in E$ then
            $U \leftarrow U \cup \{v\}$
            if $v$ is not matched or has_augmenting_path($f(v)$, used) then
                $f(v) \leftarrow h$
                return True
    return False
Function get_maximum_matching(G):
    forall $h \in H$ do
        $U \leftarrow \emptyset$
        has_augmenting_path($h, U$)
    return $f$
```



(a) (b) (c) (d)

Figure 4.13: Hungarian Algorithm

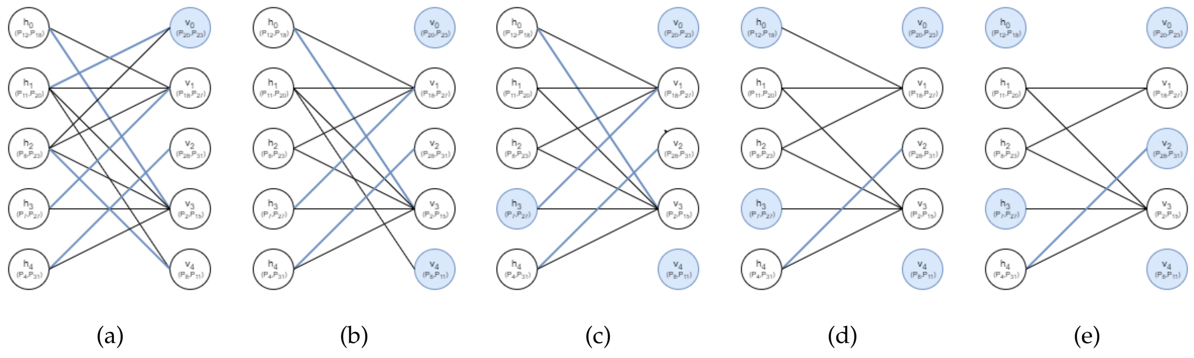


Figure 4.14: MIS Constructing Process of Givern Maximum Matching Figure 4.13d

a maximum independent set by collecting free vertices and breaking the match to create new free vertices and then collecting them. This approach base on the MaxInd algorithm from [41] and a d flag is added in our cases to get a more diverse result, especially for a perfect matching like in Figure 4.13d. In Figure 4.13d every vertices in the left are matched to one vertex in the right, there is no free vertex. V_0 is picked firstly as a new free vertex and the match between v_0 and h_1 is to be removed. v_0 intersect h_2 who also matched with v_4, v_4 is then added to F. There is an edge between v_4 and h_1, however h_1 is no longer matched to any other vertex, thus the match between v_1 and h_3 comes into processing and h_3 is added to S. h_0 and v_2 then join S accordingly as show in Figure 4.14. Plotting the chords of the maximum independent set $S = \{v_0, h_3, v_4, h_0, v_2\}$ on the rectilinear polygon yields Figure 4.12c.

Decomposition

A chord can cut the rectilinear polygon into two sub-polygons by separating the set of its vertices into two parts.For example, A vertical chord $v_0(P_{20}P_{23})$ in 4.12c divides the polygon into a rectangle formed by vertices $\{P_{20}, P_{21}, P_{22}, P_{23}\}$ and a sub-polygon formed by vertices $\{P_0, ..., P_{19}, P_{24}, ..., P_{31}\}$. Cutting along each chord in S, the rectilinear polygon is divided into the three rectangles and tree chord-free sub-polygons. These sub-polygons do not contain chords, but still have one or several concave vertices. From one of the concave vertices c, draw a horizontal and a vertical line towards the interior of the polygon, and these two line segments meet a vertical or a horizontal edge at c_h and c_v respectively. The shorter line segment will be preserved and it splits the sub-polygon into two rectangles, or a rectangle and a smaller polygon. The algorithm 7 cuts all sub-polygons and the produced smaller polygons into rectangles iteratively. The final decomposition result of rectilinear Figure 4.12a is shown in Figure 4.12d.

The coordinates of the top left vertex, length and width of a rectangle can be easily derived given four vertices, and these information will be used for layout reconstruction.

Algorithm 6: Maximum Independent Set

```
input  : A bipartite graph $G = (H \cup V, E)$; A maximum matching $f : V \to H$ with
         domain $D(f)$ and range $R(f)$
output : An maximum independent set $S$ such that $S \subseteq H \cup V$
Function get_maximum_independent_set(G,f):
    $S \leftarrow \emptyset$
    $F \leftarrow \{u : u \notin D(f) \cup R(f)\}$
    $d \leftarrow 0$
    while $(F \neq \emptyset)$ or $(D(f) \neq \emptyset)$ do
        if $F \neq \emptyset$ then
            $u \leftarrow F[0]$
            $S \leftarrow S \cup \{u\}$
            $F \leftarrow F - \{u\}$
        else if $D(f) \neq \emptyset$ then
            if $d=0$ then
                $u \leftarrow D(f)[0]$
                $E \leftarrow E - \{(u, f(u))\}$
                $D(f) \leftarrow D(f) - \{u\}$
                $d \leftarrow 1$
            else
                $v \leftarrow f(D(f)[0])$
                $u \leftarrow f(v)$
                $E \leftarrow E - \{(v, u)\}$
                $D(f) \leftarrow D(f) - \{v\}$
                $d \leftarrow 0$
            $S \leftarrow S \cup \{u\}$
        if $u \in H$ then
            forall $v \in \{v : v \in V, (v, u) \in E\}$ do
                $E \leftarrow E - \{(v, u)\}$
                if $v \in D(f)$ then
                    $D(f) \leftarrow D(f) - \{v\}$
                    $F \leftarrow F \cup \{f(v)\}$
        else
            forall $h \in \{h : h \in H, (u, h) \in E\}$ do
                $E \leftarrow E - \{(u, h)\}$
                if $h \in R(f)$ and $\exists v \in D(f), f(v) = h$ then
                    $D(f) \leftarrow D(f) - \{v\}$
                    $F \leftarrow F \cup \{v\}$
    return $S$
```

Algorithm 7: Decompose Sub-polygons

```
input  : A set of vertices $P$ that forms a sub-polygon
output : A set of vertex sets of rectangles
Function get_rectangles(P):
    if $get\_concave\_vertices(P) = \emptyset$ then
        return $\{V\}$
    $H$: set of horizontal edges
    $V$: set of vertical edges
    $c \leftarrow get\_concave\_vertices(P)[0]$
    forall $v \in V$ do
        $c_v \leftarrow (c.x, v.start.y)$
        if $intersect(cc_v, v)$ and $cc_v \notin H$ then
            Break
    forall $h \in H$ do
        $c_h \leftarrow (h.start.x, c.y)$
        if $intersect(cc_h, h)$ and $cc_h \notin V$ then
            Break
    if $|cc_v| \leq |cc_h|$ then
        $c' \leftarrow c_v$
    else
        $c' \leftarrow c_h$
    $V_1, V_2 \leftarrow divide(V, c, c')$
    return get_rectangles($V_1$) $\cup$ get_rectangles($V_2$)
```

5 Deployment

The previous chapter described how we used React and Python to implement front and back-end functionality on the developer's laptop. However, a complete web application product must be deployed to run on a secure and stable server. This chapter introduces the tools and steps taken in the deployment process.

5.1 Virtualization

Unfortunately, our project currently does not have an available dedicated server. Thus virtualization is essential. Container-based and hypervisor-based virtualization are the two leading virtualization technologies on the market nowadays [45]. The hypervisor allows multiple Virtual machines (VMs) with full copies of operating systems run on a single machine. Containers are an abstraction at the app layer that only packages resources needed for the services or applications together. Multiple containers can run on the same machine as isolated processes in user space, sharing the host OS kernel. Compared to VMs, containers are more lightweight, portable and efficient [46].

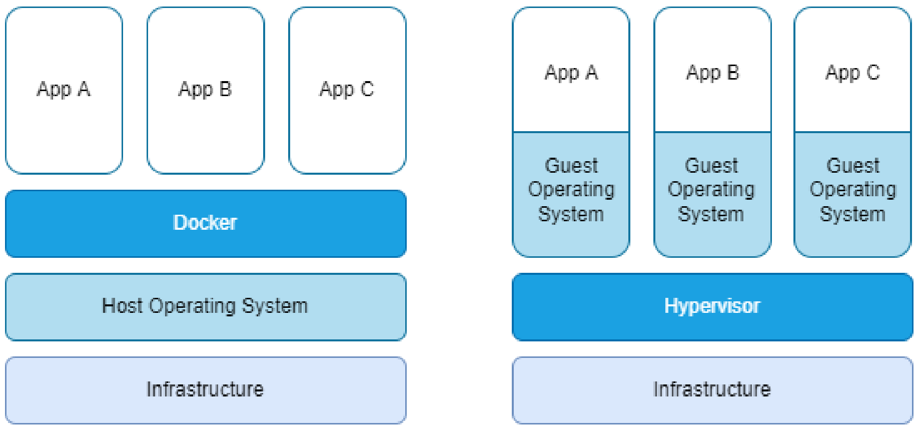
Docker is an open-source virtualization platform launched in 2013 for developing, shipping, and running applications based on containers [47]. It created and popularized modern software containers, which are the foundation for millions of enterprise applications [48]. Docker is available on various Linux, macOS and Windows platforms [49]. Applications can be easily migrated between a range of devices of development and deployment environments such as laptops, virtual machines or cloud servers regardless of the hardware differences.

The configuration of programs, libraries, resources, and parameters required by a Docker container is encapsulated in a prebaked file system Docker image [50]. On the Docker Hub, many high-quality images produced and maintained by Docker and various open source project teams can be found and directly used for production. New customized images can also be built via Dockerfile on top of existing images and then be reproduced anywhere. A Dockerfile is a text document that contains a series of instructions to assemble an image. Here are some common basic docker instructions:

FROM A valid Dockerfile always start with FROM instruction specifying the base image for subsequent instructions.

ENV An instruction sets the environment variable that could be used and replaced in the subsequent instructions.

WORKDIR An instruction sets the working directory for subsequent instructions.



(a) Container-Based (b) Hypervisor-Based

Figure 5.1: Virtualization

COPY An instruction copies files from build content source path and adds them to the file system of the container.

RUN An instruction will execute commands on top of the current image and commit the results.

EXPOSE An instruction declares network ports that the container intends to use at run time.

CMD The main purpose of a CMD instruction is to provide default process and associated parameters for an executing container.

A Docker image is built from a series of layers representing instructions [51]. Instructions like RUN and COPY that modify the container file system increase the build's size. To reduce the number of layers, firstly, try to write multiple lines of commands under the same RUN instruction connected by `&&` symbols, and secondly, where possible, use multi-stages builds and copy only the required artefacts into the final image [52].

Docker also provides a solution for defining and running multi-container Docker applications. After defining the necessary services, for example, the images and the networks, in a `docker-compose.yml` file, the different subsystems of an entire application can run together in an isolated environment [53].

5.2 Front-end Dockerfile

The front-end uses a multi-stage build. In the first stage, the React project is built using NodeJS and NPM, and the build file is copied to the Nginx image to set up a proxy server container. The front-end Dockerfile is defined as Listing 5.1.

Listing 5.1: Front-end Dockerfile

```
# STAGE 1
FROM node:alpine as builder
WORKDIR /app
COPY package.json .
RUN npm install --legacy-peer-deps
COPY . .
RUN npm run build

# STAGE 2
FROM nginx:stable-alpine
COPY --from=builder /app/build /usr/share/nginx/html
RUN rm /etc/nginx/conf.d/default.conf
COPY nginx.conf /etc/nginx/conf.d
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

As mentioned earlier, the front-end project uses many external packages. `npm install` will install all modules listed as dependencies in `package.json` under `node_module` directory. Each package's exact source and version will also be detailed in `package-lock.json`. Flag `--legacy-peer-deps` provides a way forward to resolve packages that cannot be installed because of overly strict peer dependencies.

`COPY package.json .` copies the `package.json` file to the working directory and `COPY . .` copies all files except those declared in `.dockerignore` in the front-end folder to the working directory. The `package.json` is actually copied twice. This minor duplication can significantly reduce the time spent building images during development and testing. Docker has a cache system. If there is no change in the current and preceding layers, it will re-use the same layer in the cache instead of creating a new one. The following is the output of two successive builds:

```
PS E:\layout-decomposition-webapi\frontend> docker build .
[+] Building 132.9s (16/16) FINISHED
 => [stage-1 1/4] FROM docker.io/library/nginx:stable-alpine (3.7s)
 => [builder 1/6] FROM docker.io/library/node:alpine (10.9s)
 => [builder 2/6] WORKDIR /app (0.1s)
 => [builder 3/6] COPY package.json . (0.0s)
 => [builder 4/6] RUN npm install --legacy-peer-deps (108.2s)
 => [builder 5/6] COPY . . (0.0s)
 => [builder 6/6] RUN npm run build (11.1s)
 => [stage-1 2/4] COPY --from=builder /app/build /usr/share/nginx/html (0.0s)
 => [stage-1 3/4] RUN rm /etc/nginx/conf.d/default.conf (0.3s)
 => [stage-1 4/4] COPY nginx.conf /etc/nginx/conf.d (0.0s)
```

```
PS E:\layout-decomposition-webapi\frontend> docker build .
[+] Building 1.6s (16/16) FINISHED
 => [builder 1/6] FROM docker.io/library/node:alpine (0.0s)
 => [stage-1 1/4] FROM docker.io/library/nginx:stable-alpine (0.0s)
 => CACHED [builder 2/6] WORKDIR /app (0.0s)
 => CACHED [builder 3/6] COPY package.json . (0.0s)
 => CACHED [builder 4/6] RUN npm install --legacy-peer-deps (0.0s)
 => CACHED [builder 5/6] COPY . . (0.0s)
 => CACHED [builder 6/6] RUN npm run build (0.0s)
 => CACHED [stage-1 2/4] COPY --from=builder /app/build /usr/share/nginx/html (0.0s)
 => CACHED [stage-1 3/4] RUN rm /etc/nginx/conf.d/default.conf (0.0s)
 => CACHED [stage-1 4/4] COPY nginx.conf /etc/nginx/conf.d (0.0s)
```

In the first build, the most time-consuming instruction `RUN npm install --legacy-peer-deps` took nearly two minutes. But in the second build, the direct use of the cached layers was done instantly. Copying the `package.json` alone in advance can save lots of time by skipping the step of installing dependencies once the code is changed, but no more external modules are added to the dependency.

The command `npm run build` creates a `build` directory with an optimized and minimized production build of React application. Inside the `build` folder, there is an `index.html` - the single web page that holds and renders all the UI components, a `static` sub-folder containing the hashed JavaScript and CSS files, and the public resources. All the files in the `build` directory will be copied to the Nginx image in the next stage.

Nginx, pronounced "Engine X", is an advanced load balancer, web server, and reverse proxy. It is commonly used to serve static Web content and proxy servers. The file `/etc/nginx/conf.d/default.conf` contains the default HTTP server configuration [54]. Modifying the configuration file is essential to build a full Nginx image. The content of original `default.conf` is replaced with Listing 5.2, which declares that the Nginx server listens on port 80, dispatches the received requests starting with “`/api`” to the back-end container, and serves the `index.html` built in the early stage.

Listing 5.2: Nginx Configuration

```
server {
  listen 80 default_server;
  listen [::]:80 default_server;
  # server_name domain_name public_ip;
  location / {
    root /usr/share/nginx/html;
    index index.html index.htm;
    try_files $uri $uri/ /index.html;
  }
  location /api {
     proxy_pass http://127.0.0.1:8000;
  }
}
```

5.3 Back-end Dockerfile

Although there are many Python and Inkscape images on Dockerhub, it is hard to find an image with both environments. So unlike the front-end, where subsequent operations can be performed directly on the existing node and Nginx images, the back-end image needs to install both software separately on the Ubuntu base image. The Listing 5.3 contains the content of the back-end Dockerfile.

Listing 5.3: Back-end Dockerfile

```
FROM ubuntu:20.04
ENV TZ=Europe/Berlin
RUN apt update \
    # install python3
    && apt -y install python3 \
    && apt -y install python3-pip \
        # fix timezone issue
        && ln -snf /usr/share/zoneinfo/$TZ /etc/localtime \
        && echo $TZ > /etc/timezone \
    # install inkscapes
    && apt -y install software-properties-common \
    && add-apt-repository ppa:inkscape.dev/stable \
    && apt update \
    && apt -y install inkscape
# set working directory;
WORKDIR /app
# copy and install Python dependencies;
COPY requirements.txt .
RUN pip3 install -r requirements.txt
COPY . .
# finally, start server;
CMD ["python3","-m","uvicorn","main:app","--host","0.0.0.0","--port","8000"]
```

Python could be esaily downloaded via Advanced Package Tool (APT) of Linux and the installation instructions for Inkscape on Ubuntu can be found on the official website [55]. Time zone information will be required when installing Inkscape. It must be set in advance to avoid blocking the build due to pending manual input during the Docker image build. After installing the essential software, it simply copies the project files to the working directory and installs the Python dependencies using pip, the package installer for Python. For the same reason for saving secondary build time by taking advantage of the Docker caching mechanism, requirements.txt is copied separately, just like package.json in the front-end. Finally, the back-end process is started using uvicorn, a recommended high performance Asynchronous Server Gateway Interface (ASGI) web server implementation for FastApi [56]. It binds the socket to port 8000 of localhost.

5.4 Docker Compose

Docker Compose is a tool for defining and running multi-container Docker applications. The docker-compose.yml file specifies the location of the Dockerfiles, declares the container names, and defines the mapping of the host ports to the container ports. To start the layout-decomposition-frontend and layout-decomposition-backend containers, only one command line, docker compose up, is needed. When both front and back-end containers are running, the browser will return the homepage as the result of accessing http://localhost:3000, and the activities in Figure 4.1 and Figure 4.2 can be triggered appropriately. Accessing http://localhost:8000/docs will lead to the FastApi interactive documentation page as shown in Figure 4.7. To stop the whole application and close and delete the corresponding containers also requires only one line of instruction docker compose down.

Listing 5.4: Docker Compose File

```yaml
version: '3'
services:
  frontend:
    build: ./frontend
    container_name: layout-decomposition-frontend
    ports:
      - "3000:80"
  backend:
    build: ./backend
    container_name: layout-decomposition-backend
    ports:
      - "8000:8000"
```

6 Conclusion

Overall, this project is a qualified prototype for the Printing-Based Microfabrication Layout Decomposition Web application. The requirements stated in chapter 3 can all be met by developing a front-end project based on the React framework, a back-end project based on Python FastApi, and using a containerised deployment approach.

Functionally, sample design layout Figure 1.1a in form of SVG, PDF and DXF can all be decomposed correctly via the Web application. With a smooth threshold value of 0.2px, the step errors shown in Figure 3.1 are eliminated. The decomposition result shown as Figure 6.1 is presented on the Result Page and can be downloaded as an SVG, a PDF or a DXF file.

For users, the web application is easy to use. The user interface is straightforward, and each functional component has explanatory labels, so the user does not need to spend any effort learning how to use it.

For successor developers, the project is extendable and portable. Many components such as the navigation bar and download button in the front-end program can be directly reused to develop new features. Containerized deployments allow projects to run on any machine that has Docker is installed and to be easily deployed to physical servers or to the cloud when a product environment is in place.

6.1 Open Points

Although the project has achieved the necessary functionality, it is still far from being a publicly available website. Firstly, because of the limited number of available printing-based microfabrication designs, large-scale testing is not possible. The SVG formats exported by different software are so varied that even with optimization using scour, not all unnecessary information can be removed completely. Some residual SVG invalid formats are "inert" and do not negatively affect the results, but others are "pernicious" and may result in SVG files being incorrectly parsed as input or failing to render on a web page as output. The lack of diverse example images makes applying targeted and aggressive SVG optimizer difficult. Secondly, the necessary resources to host a public website are currently missing, not just the server but also a valid domain name and required documents like certificates. Finally, due to time and technology constraints, there is no comprehensive solution for security issues such as image data protection and defence against cyber attacks.

6.2 Outlook

The current application has only one main purpose, to decompose the layout of the printing-based microfabrication. In the future, it could be further developed based on the study of [4] by adding new features which assign rectangles without printing conflicts to the same layer to optimize the final product with layered printing while saving drying time.

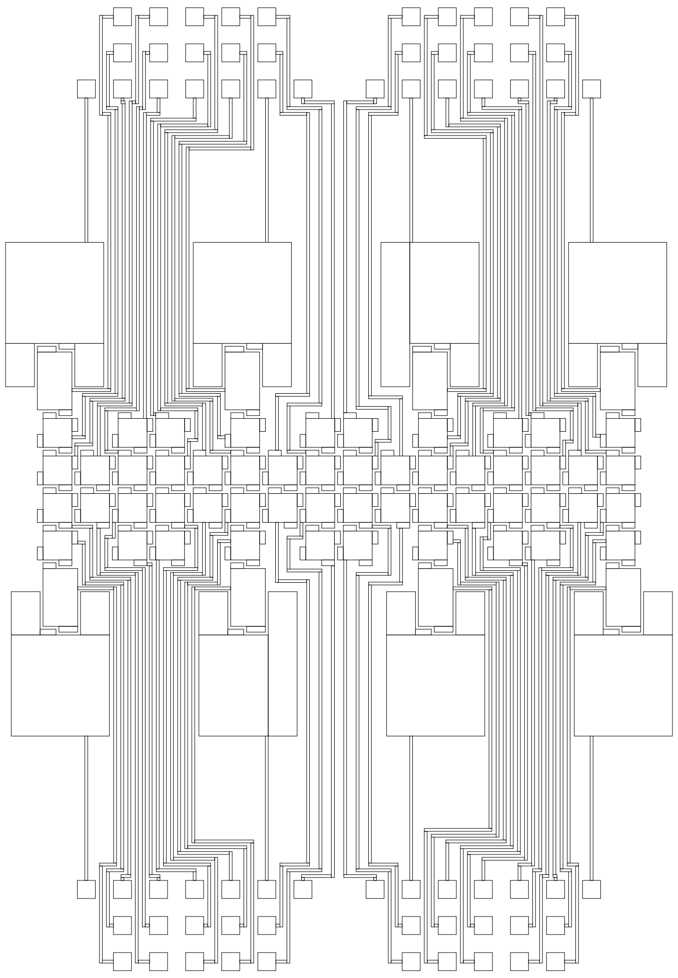


Figure 6.1: Decomposition Result of Sample Printing-Based Microfabrication

List of Figures

List of Algorithms

Listings

Bibliography

[1] L. P. Hue. “Progress and trends in ink-jet printing technology”. In: *Journal of Imaging Science and Technology*. 1st ser. 42 (1998), pp. 49–62.

[2] G. Cummins and M. P. Desmulliez. “Inkjet printing of conductive materials: a review”. In: *Circuit world* 38.4 (2012), pp. 193–213.

[3] C. W. Sele, T. von Werne, R. H. Friend, and H. Sirringhaus. “Lithography-Free, Self-Aligned Inkjet Printing with Sub-Hundred-Nanometer Resolution”. In: *Advanced Materials* 17.8 (2005), pp. 997–1001. DOI: `10.1002/adma.200401285`.

[4] T.-M. Tseng, M. Lian, M. Li, P. Rinklin, L. Grob, B. Wolfrum, and U. Schlichtmann. “Manufacturing Cycle-Time Optimization Using Gaussian Drying Model for Inkjet-Printed Electronics”. In: *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 2021, pp. 1–8. DOI: `10.1109/ICCAD51958.2021.9643438`.

[5] J. Mark Keil. “Chapter 11 - Polygon Decomposition”. In: *Handbook of Computational Geometry*. Ed. by J.-R. Sack and J. Urrutia. Amsterdam: North-Holland, 2000, pp. 491–518. ISBN: 978-0-444-82537-7. DOI: `https://doi.org/10.1016/B978-044482537-7/50012-7`. URL: `https://www.sciencedirect.com/science/article/pii/B9780444825377500127`.

[6] J. M. Keil. “Polygon Decomposition.” In: *Handbook of computational geometry* 2 (2000), pp. 491–518.

[7] T. Ohtsuki. “Minimum dissection of rectilinear regions”. In: *Proc. 1982 IEEE Symp. on Circuits and Systems, Rome*. 1982, pp. 1210–1213.

[8] H. Imai and T. Asano. “Efficient algorithms for geometric graph search problems”. In: *SIAM Journal on Computing* 15.2 (1986), pp. 478–494.

[9] W. Liou, J. Tan, and R. Lee. “Minimum partitioning simple rectilinear polygons in O(n log log n)-time”. In: *Proceedings of the fifth annual symposium on Computational geometry*. 1989, pp. 344–353.

[10] B. Lutkevich. *What are Vector Graphics?* 2021. URL: `https://www.techtarget.com/whatis/definition/vector-graphics`.

[11] 2006. URL: `https://techterms.com/definition/vectorgraphic`.

[12] 2018. URL: `https://www.w3.org/TR/SVG/`.

[13] URL: `https://www.w3schools.com/graphics/svg_rect.asp`.

[14] *Everything you need to know about the PDF.* URL: `https://www.adobe.com/acrobat/about-adobe-pdf.html`.

[15] URL: `https://www.adobe.com/acrobat/features.html`.

[16] *Document management — Portable document format — Part 1: PDF 1.7*. 2008.

[17] W. Kneale. "Aristotle's Syllogistic from the Standpoint of Modern Formal Logic. By Jan Lukasiewicz. (Oxford: Clarendon Press. 1951. Pp. xi 141. 15s.)" In: *Philosophy* 27.102 (1952), pp. 279–282. DOI: `10.1017/S0031819100034227`.

[18] 2018. URL: `https://help.autodesk.com/view/OARX/2018/ENU/?guid=GUID-235B22E0-A567-4CF6-92D3-38A2306D73F3`.

[19] URL: `https://www.coreldraw.com/en/pages/dxf-file/`.

[20] M. Jazayeri. "Some Trends in Web Application Development". In: *Future of Software Engineering (FOSE '07)*. 2007, pp. 199–213. DOI: `10.1109/FOSE.2007.26`.

[21] Z. Molnar. *Decoupled architecture: how to modernise your frontend*. 2020. URL: `https://inviqa.com/blog/decoupled-architecture-how-modernise-your-frontend`.

[22] R. T. Fielding. "REST: architectural styles and the design of network-based software architectures". In: *Doctoral dissertation, University of California* (2000).

[23] I. Sommerville. *Software Engineering*. Pearson Education India, 2011.

[24] V. Joshi. *Seven Reasons Why A Website's Front-End And Back-End Should Be Kept Separate*. 2018. URL: `https://www.forbes.com/sites/forbestechcouncil/2018/07/19/seven-reasons-why-a-websites-front-end-and-back-end-should-be-kept-separate/?sh=8ac70e34fca2`.

[25] 2021. URL: `https://insights.stackoverflow.com/survey/2021#most-loved-dreaded-and-wanted-language-love-dread`.

[26] URL: `https://reactjs.org/community/team.html`.

[27] 2015. URL: `https://netflixtechblog.com/netflix-likes-react-509675426db`.

[28] URL: `https://reactjs.org/docs/faq-internals.html`.

[29] URL: `https://reactjs.org/docs/components-and-props.html`.

[30] S. Sorhus. *Tiny and elegant JavaScript HTTP client based on the browser Fetch API*. 2020. URL: `https://github.com/sindresorhus/ky`.

[31] Z. Wang. 2015. URL: `https://github.com/wwayne/react-tooltip`.

[32] R. Pearce. 2020. URL: `https://github.com/rpearce/react-medium-image-zoom`.

[33] 2015. URL: `https://github.com/schrodinger/rc-slider`.

[34] 2020. URL: `https://github.com/tailwindlabs/heroicons`.

[35] URL: `https://fastapi.tiangolo.com/#performance`.

[36] URL: `https://inkscape.org/doc/inkscape-man.html`.

[37] M. Moitzi. 2020. URL: `https://github.com/mozman/ezdxf`.

[38] M. Moitzi. 2012. URL: `https://github.com/mozman/svgwrite`.

[39] 2004. URL: `https://github.com/tailwindlabs/heroicons`.

[40] A. A. Port. 2015. URL: https://github.com/mathandy/svgpathtools.

[41] S.-Y. Wu and S. Sahni. "Fast algorithms to partition simple rectilinear polygons". In: *VLSI Design* 1.3 (1994), pp. 193–215.

[42] R. Fu and H. Shen. "An algorithm for determining concave vertex of object based on vector product". In: *Journal of Electronics (china)* 27 (Mar. 2010), pp. 212–217. DOI: 10.1007/s11767-010-0312-3.

[43] H. W. Kuhn. "The Hungarian method for the assignment problem". In: *Naval research logistics quarterly* 2.1-2 (1955), pp. 83–97.

[44] J. Munkres. "Algorithms for the assignment and transportation problems". In: *Journal of the society for industrial and applied mathematics* 5.1 (1957), pp. 32–38.

[45] T. Bui. "Analysis of docker security". In: *arXiv preprint arXiv:1501.02967* (2015).

[46] URL: https://www.docker.com/resources/what-container/.

[47] B. B. Rad, H. J. Bhatti, and M. Ahmadi. "An introduction to docker and analysis of its performance". In: *International Journal of Computer Science and Network Security (IJCSNS)* 17.3 (2017), p. 228.

[48] M. Wheatley. *Docker hits double unicorn status after raising $105m in series C funding round*. Apr. 2022. URL: https://siliconangle.com/2022/03/31/docker-hits-double-unicorn-status-raising-105m-series-c-funding-round/.

[49] URL: https://docs.docker.com/engine/install/.

[50] C. Anderson. "Docker [Software engineering]". In: *IEEE Software* 32.3 (2015), pp. 102–c3. DOI: 10.1109/MS.2015.62.

[51] URL: https://docs.docker.com/storage/storagedriver/.

[52] URL: https://docs.docker.com/develop/develop-images/dockerfile_best-practices/#minimize-the-number-of-layers.

[53] URL: https://docs.docker.com/compose/.

[54] D. DeJonghe. *Nginx CookBook*. O'Reilly Media, 2020.

[55] URL: https://inkscape.org/release/inkscape-dev/gnulinux/snap/dl/.

[56] URL: https://fastapi.tiangolo.com/deployment/manually/.