Trimap Matting

Scuola d'Arti e Mestieri di Trevano (SAMT)
Documentation

Paolo Bettelini

Contents

1 Introduction

1.1 Abstract

Background removal has been a longstanding practice in digital image processing. While removing the background from images with simple borders is relatively straightforward, more complex borders, such as hair, require the use of more sophisticated techniques. Trimap matting is a novel and effective approach to solving the latter. This project involves the development of a user-friendly command-line and web graphical user interface to make use of this algorithm.

1.2 Information

This is a project of the Scuola Arti e Mestieri di Trevano (SAMT) under the following circumstances

- **Section**: Computer Science
- **Year:** Fourth
- **Class:** Progetti Individuali
- **Supervisor:** Geo Petrini
- **Title:** Trimap Matting
- **Start date**: 2022-12-12
- **Deadline**: 2023-04-06

and the following requirements

- **Documentation**: a full documentation of the work done
- **Diary**: constant changelog for each working session
- **Source code**: source code of the project

All the source code and documents can be found at https://github.com/paolobettelini/trimap-matting [1].

2 Requirements

Req-00	
Name	CLI tool
Priority	1
Version	1.1
Notes	none
Description	A CLI tool to execute background removal must be developed.
Subrequirements	
Req-00_0	The target image must be specified.
Req-00_1	The trimap image can be specified.
Req-00_2	The soft mask can be specified.
Req-00_3	Either the soft mask or the trimap must be specified.
Req-00_4	The program can save the generated background mask.
Req-00_5	The program can remove the background and replace it with an image.
Req-00_6	The program can remove the background and fill it with a color.
Req-00_7	The program can remove the background and leave it transparent.

Req-01	
Name	Image formats
Priority	1
Version	1.0
Notes	none
Description	Multiple image formats must be supported.
Subrequirements	
Req-00_0	The JPG format must be supported.
Req-00_1	The PNG format must be supported.
Req-00_2	The WebP format must be supported.

Req-02	
Name	Size check
Priority	1
Version	1.0
Notes	none
Description	The executable must assert that the target image and trimap are of the same size.

Req-03	
Name	GUI
Priority	1
Version	1.0
Notes	none
Description	A GUI application must be developed in other to interact with the program features.

Req-04	
Name	Default name
Priority	1
Version	1.0
Notes	The name must be generated using the current timestamp.
Description	The mask and output files must be assigned a default name none is specified.

Req-05	
Name	Logging
Priority	1
Version	1.0
Notes	none
Description	The programs must have logging capabilities

3 Trimap Matting

Matting is a technique used to extract an object from an image. Trimap matting is a term used to refer to the process of generating alpha mattes[9] for an object in an image given an initial approximation of its borders.

The goal of this process is to determine how much each pixel of a target image is part of the object that needs to be extracted. This means that given a pixel $P_{x,y}$ we want to find a value $\alpha \in \mathbb{R}$ such that

$$P_{x,y} = \alpha F_{x,y} + (1 - \alpha) B_{x,y}, \quad \alpha \in [0; 1]$$

where F represents the foreground color and B represents the background color at a given pixel.

Note that the multiplicative operator here is the scalar vector multiplication. This is because the pixels are represented by a vector of values, usually $\mathbb{R}^3$ or $\mathbb{R}^4$ (for transparent images).

The trimap is an approximation of the alpha mattes. The black dye represents background-only space, the white dye represents foreground-only space whilst the gray one delimits the distinction between the two.

Here are some examples using an image of a plant.

Figure 1: Plant Image

Figure 2: Plant Trimap

Figure 3: Plant Soft Mask

Once the alpha mattes are generated (soft mask) we can use them to extract the object from the target image. Thus, we can remove the background behind the object, fill the background with a color, replace it with another image, or leave it transparent.

Given a target image with pixels $T_{x,y}$, the alpha mattes $\alpha_{x,y}$ and the pixels of the replacement image $R_{x,y}$ we can compute the pixels of the output image $T'_{x,y}$ as follows

$$T'_{x,y} = \alpha_{x,y} \cdot T_{x,y} + (1 - \alpha_{x,y}) \cdot R_{x,y}$$

If we want to replace the background with a color, we can consider. $R = (r, g, b)^t$
If we just want to leave the background transparent, meaning $T' \in \mathbb{R}^4$, we have

$$T'_{x,y} = \alpha_{x,y} \cdot T_{x,y}$$

and the alpha value for $T_{x,y}$ is set to $\alpha_{x,y}$.

Figure 4: Transparency

Figure 5: Color fill

Figure 6: Replacement

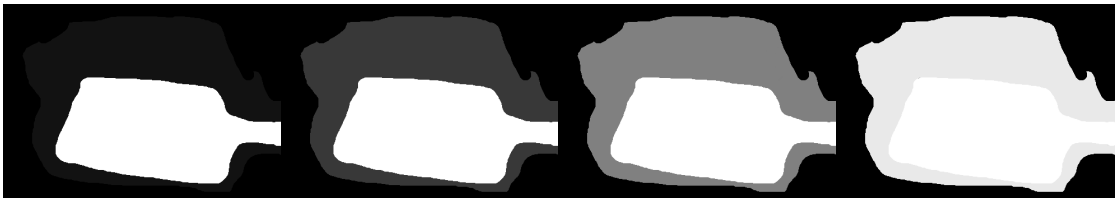
4 Technologies

4.1 OpenCV

OpenCV[**opencv**] is a library for computer vision. It contains a large amount of tools, from GUIs, video analysis, machine learning, object detection, image processing and many more[7]. The library also contains a module about *alpha matting*, which contains a function to generate alpha mattes given a trimap [5].

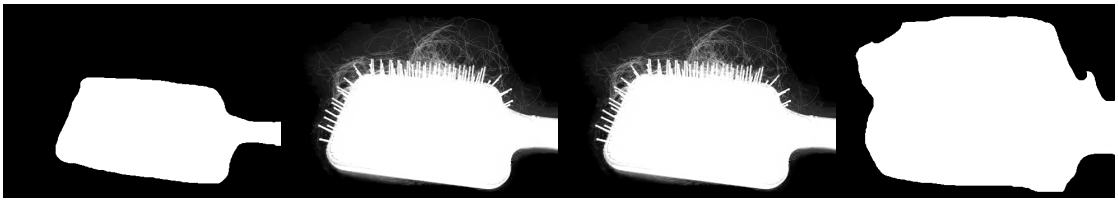
4.1.1 Trimap colors

As stated in the last section, the gray color represents the unknown pixels. The library, however, accepts a greyscale trimap. The question is whether different grays have different meanings. This is not documented and would require to read the original paper of implementation.
In order to find out, here are a bunch of samples with different gray colors:



(a) Trimap 1 (b) Trimap 2 (c) Trimap 3 (d) Trimap 4

Figure 7: Hairy brush trimaps



(a) Mask 1 (b) Mask 2 (c) Mask 3 (d) Mask 4

Figure 8: Hairy brush masks

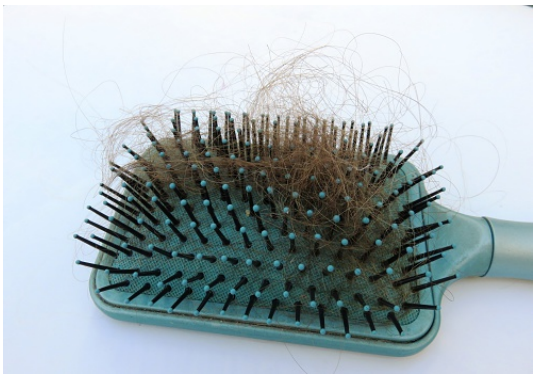


Figure 9: Hairy brush

Given that this is the original image, we can confidently infer that different gray values have the same meaning. However, there is a range according to which a color is considered `black`, `gray` or `white`. As we can see in 8(a) and 8(d), the unknown pixels have been considered to be background or foreground respectively, because the gray value was too low or too high.

4.2 Rust

Rust[8] is a generic compiled programming language. The code is compiled using LLVM to machine code and its speed is comparable to C and C++. Rust is the first programming language to guarantee memory safety; memory is not manually freed nor garbage collected. It is not possible to dereference a null pointer, cause memory segfaults, core dumps and memory leaks. Code that could cause undefined behavior can still be written, but it is strictly bounded in blocks where the compiler is relaxed. This relaxation implies that the language is also low-level. Another key feature to the performance of Rust is zero cost abstraction, which means that generic types and function abstractions are resolved at compile-time. Conditional compilation and compile-time computations are also extensively used.
There are also many features concerning the programming experience, such as advanced metaprogramming and code generation using macros, intelligent compiler, dependency system (Cargo), modern syntax and many tools to ease development.

Note: a Rust *crate* refers to a library. A *feature* is an optional component of library. A *module* is a logical section of a program or library.

5 Compilation and usage

5.1 CLI

5.1.1 Compilation

The executable can be compiled using the `cargo` package manager.

```
$ cd matting-cli
$ cargo build --release
```

This will generate an executable (`matting-cli`) in `./target/release`. In order to make this executable globally available, you can move it into a folder in the $PATH environment variable, such as `/usr/bin`. Ye may also modify the executable file name to change its invocation name.

```
$ sudo mv target/release/matting-cli /usr/bin/
```

The executable can now be invoked by just writing

```
$ matting-cli
```

5.1.2 Usage

The following shows the output of the command upon setting the `--help` or `-h` flag.

```
Matting CLI

Usage: matting-cli [OPTIONS] --target <TARGET>
                   <--mask <MASK>|--trimap <TRIMAP>>

Options:
  -i, --target <TARGET>        Target image
      --mask <MASK>            Background mask image
      --trimap <TRIMAP>        Trimap image
      --save-mask <SAVE_MASK>  Save mask path
  -o, --output <OUTPUT>        Output image
  -f, --fill <FILL>            Fill background action
  -t, --transparent            Transparent background action
  -r, --replace <REPLACE>      Replace background action
      --verbose                Verbose flag
  -h, --help                   Print help information
  -V, --version                Print version information
```

The `--target` parameter specifies the image on which the operation needs to be applied. This parameter is mandatory.

The `--trimap` parameter specifies the trimap image which will be used to generate the alpha mattes.

The `--mask` parameter specifies the image containing the alpha mattes to use.

The parameter `--trimap` and `--mask` are mutually exclusive, and one of them is mandatory.

The advantage of using `--mask` over `--trimap` is that the alpha mattes are already given rather than having to be computed. This can save lots of computational times. The alpha mattes image can be saved on the file system by specifying the `--save-mask` parameter.

The parameter `--save-mask` can either specify a filename for the mask. If left blank, the program will generate a name (PNG format) using the timestamp.

There are 3 different operations that can be applied to the background of the result: `--transparent`, `--replace` or `--fill`. These operations are mutually exclusive, and if one is specified, the `--output` parameter can be set to specify the path where the resulting image will be saved.

If the `--output` parameter is not specified then it will produce a file name with the current timestamp.

Note: the argument of `--color` can be any valid CSS color. See [4] for the documentation.

The `--verbose` flag is optional and will print additional information about what the program is doing and the elapsed time of each operation. Se the logging section for more information.

5.1.3 Examples

The following command generates a mask of the alpha mattes given a trimap and saves it to `mask.png`.

```
$ matting-cli -i target.jpg --trimap trimap.png
    --save-mask mask.png
```

The following command generates a mask of the alpha mattes given a trimap and saves it to a file (e.g. `mask_20230313-133117.png`).

```
$ matting-cli -i target.jpg --trimap trimap.png
    --save-mask
```

The following command removes the background of an image given its mask.

```
$ matting-cli -i target.jpg --mask mask.png
    -o out.png --transparent
```

The following command removes the background of an image given its mask and replaces it with the color `#A3BF13`.

```
$ matting-cli -i target.jpg --mask mask.png
    -o out.png --fill "#A3BF13"
```

The following command removes the background of an image given its mask and replaces it with the image `background.png`.

```
$ matting-cli -i target.jpg --mask mask.png
    -o out.png --replace background.png
```

The following command removes the background of an image given its mask and leaves it transparent. The output is saved to a file (e.g. `target_20230313-134153.png`).

```
$ matting-cli -i target.jpg --mask mask.png
    --transparent
```

The following shows the output of the program when the `--verbose` flag is set. This command computes the alpha mattes given a trimap, then it saves the generated mask, fills the background of the image with the color red, and then saves the result.

```
$ matting-cli -i target.jpg --trimap trimap.png
    --save-mask mask.png -o out.png --fill red --verbose

Reading target image... Done! [4.021485ms]
Reading trimap image... Done! [1.976477ms]
Generating soft mask... Done! [7.104532642s]
Reading target image... Done! [178.884124ms]
Saving soft mask... Done! [509.050896ms]
Filling background with color... Done! [117.90393ms]
Saving output... Done! [954.085622ms]
```

5.2 Web application

5.2.1 Compilation

The executable can be compiled using the `cargo` package manager.

```
$ cd matting-web
$ cargo build --release
```

This will generate an executable (`matting-web`) in `./target/release`. In order to make this executable globally available, you can move it into a folder in the $PATH environment variable, such as `/usr/bin`. You may also modify the executable file name to change its invocation name.

```
$ sudo mv target/release/matting-web /usr/bin/
```

The executable can now be invoked by just writing

```
$ matting-web
```

5.2.2 Usage

The following shows the output of the command upon setting the `--help` or `-h` flag.

```
Matting WEB

Usage: matting-web [OPTIONS] --www <WWW>

Options:
    -w, --www <WWW>             WWW static files folder
    -a, --address <ADDRESS>     Listening address [default: 0.0.0.0]
    -p, --port <PORT>           Listening port [default: 8080]
    -l, --log-file <LOG_FILE>   Log file
    -h, --help                  Print help
    -V, --version               Print version
```

The `--www` parameter specifies the path of the static website files. This parameter is mandatory.

The `--address` parameter specifies the listening address of the server.

The `--port` parameter specifies the listening port of the server.

The `--log-file` parameter specifies a file to log to instead of the STDOUT and STDERR.

5.2.3 Examples

The following command starts the server using the default address and port (`0.0.0.0:8080`).

```
$ matting-web --www /path/to/wwww
```

The following command starts the server listening to (`::1:80`).

```
$ matting-web -w /path/to/wwww -a ::1 -p 80
```

The following command starts the server and logs everything to a file.

```
$ matting-web -w /path/to/wwww -l logfile.log
```

6 Planning

6.1 Initial Gantt Chart

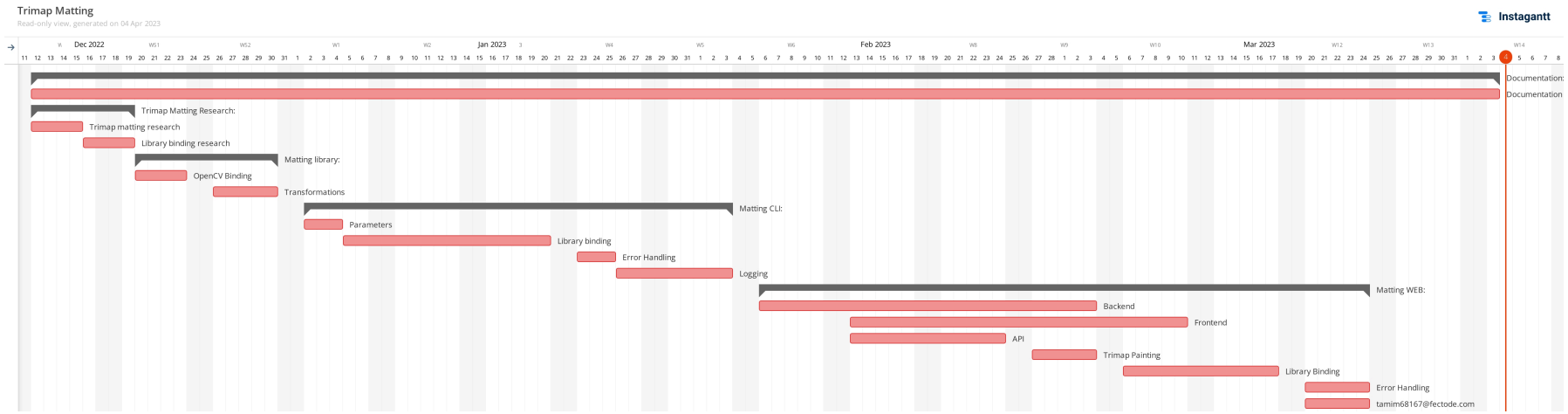


Figure 10: Initial Gantt Chart

6.2 Final Gantt Chart

7 Implementation

7.1 OpenCV fundamental types

The fundamental OpenCV type used in this project is the **Mat**[6] type. A mat is essentially an n-dimensional array. It can represent tensors, vector fields, matrices and much more, both with real numbers or complex numbers. For the purpose of this project, any mat represents a grayscale or a colored image.

7.2 OpenCV Binding

The OpenCV library is primarily written in C++. In order to access its functions in Rust, I used a crate called `opencv`[**rustopencv**], which is a binding to the FFI of opencv.

It does not fully implement the Rust idioms and best practices, I found it very easy to accidentally cause a core dump whilst calling a function.

The *Flow Alpha Matting* module was developed by Muskaan Kularia at Google Summer of Code 2019.

The code of interest is in the `opencv::alphamat` module, and it contains the following function which computes alpha mattes of an object in an image.

```
pub fn into_flow(
    image: &dyn ToInputArray,
    tmap: &dyn ToInputArray,
    result: &dyn ToOutputArray,
) -> Result<()>
```

Parameters:

- **image:** Input RGB image
- **tmap:** Input greyscale trimap image
- **result:** Output alpha matte image

The function intoFlow performs alpha matting on a RGB image using a greyscale trimap image, and outputs a greyscale alpha matte image.
The output alpha matte can be used to softly extract the foreground object from a background image.

The traits `ToInputArray` and `ToOutputArray` are implemented by the `Mat` type.

7.3 Core dump handling

The opencv matting module will crash if the parameters are incorrect. For instance, the image parameters are represents by a `Mat` structure, and if the mats are of different sizes or have different color depth, the library may try to access a mat as if it was bigger, causing a memory segmentation and a core dump.

These incidents can be prevented by checking the size of the image and ensuring the correct color depth before calling the library, however, some incidents are harder to predict; if the trimap is only white (foreground) or only black (background), then the library will cause a core dump.
It is also possible to prevent these crashes by checking every single pixel of the canvas, however, I decided not do so because it is practically impossible to prevent every (obscure) core dump. There may be more strange core dumps that I have not encountered.

In order to fix this issue and prevent the server from abruptly stopping, the trimap matting operations are implemented by calling the `matting-cli` tool. This will start a secondary process. If the secondary process crashes, the server mustn't be impacted. The images are passed to the other software using temporary file in the `/tmp` directory (automatic cleanup is ensured by the Rust memory safety model).

Although it works, this solution is not how I would have wished to have implemented the functionality. For this reason, I also implemented the direct FFI call.
By compiling with the feature *no_matting_cli*, e.g.

```
cargo build --release --features no_matting_cli
```

the program will not call the `matting-cli` executable.

7.4 API Routes

The routes are the following. Note that each of them must include a `multipart/form-data` attribute.

- **POST**: **/api/matting**
  This endpoint returns in image representing the alpha mattes mask.
  The caller must specify two images: **target** and **trimap**.
- **POST**: **/api/fill/<color>**
  This endpoint replaces the background with a given color.
  The caller must specify two images: **target** and **mask**.
- **POST**: **/api/transparent**
  This endpoint removes the background and leaves it transparent.
  The caller must specify two images: **target** and **mask**.
- **POST**: **/api/replace**
  This endpoint replaces the background with a given image.
  The caller must specify three images: **target**, **mask** and **replacement**.

7.5 Matting library

The core of this project is certainly the alpha matting algorithm. In order to develop both a CLI and a GUI tool to use this algorithm, I created a separate library (`matting-lib`) containing all the matting and image processing related functions.

The library exports the following functions, modules and structures

```
pub use image::{ImageFormat, DynamicImage};
pub use opencv::imgcodecs::*;

/// Export error module
pub mod error;

pub fn generate_mask(target: &Mat, trimap: &Mat) -> MessageResult<Mat>

pub fn same_size<P: AsRef<Path>>(path1: P, path2: P) -> MessageResult<bool>

pub fn bytes_to_mat(data: &[u8], flags: i32) -> MessageResult<Mat>

pub fn read_as_mat(filename: &str, flags: i32) -> MessageResult<Mat>

pub fn read_as_image(filename: &str) -> MessageResult<DynamicImage>

pub fn bytes_to_image(data: &[u8]) -> MessageResult<DynamicImage>

pub fn mat_to_dynamic_image_gray(mat: &Mat) -> MessageResult<DynamicImage>

pub fn remove_background(image: &RgbImage, mask: &RgbImage) -> DynamicImage

pub fn replace_background(
    image: &RgbImage,
    mask: &RgbImage,
    replacement: &RgbImage,
) -> DynamicImage

pub fn fill_background(
    image: &RgbImage,
    mask: &RgbImage,
    color: [u8; 4],
) -> DynamicImage
```

```
pub fn image_to_format(image: DynamicImage, format: ImageFormat) -> Vec<u8>
```

The `error` module exports error management utils

```
#[derive(Debug)]
pub struct MessageError {
    pub message: String,
}

impl MessageError {
    pub fn new(message: String) -> Self;
}

/// Result type
pub type MessageResult<T> = Result<T, MessageError>;


/// Conversion traits implementation

impl From<String> for MessageError;

impl From<&str> for MessageError;

impl From<OpencvError> for MessageError;

impl From<ImageError> for MessageError;
```

7.6 Website

7.6.1 Features

The website implements a set of features:

1. If a transformation is applied whilst the mask is loading, the site will wait until the mask is ready.
2. The trimap panting tool aren't enabled until an image is uploaded.
3. The mask download button and the transformation selector aren't enabled until a mask has been generated or uploaded.
4. The header dropdown is responsive and needs to be toggled when the page is small.

7.6.2 Painting mechanics

The painting is done via the Canvas API [**canvasapi**].

The canvas CTX has the following drawing properties set

1. **lineCap**: “round”
2. **lineJoin**: “round”

Lines are drawn onto the canvas using paths.

```
ctx.beginPath();
ctx.moveTo(x0, y0);
ctx.lineTo(x1, y1);
ctx.lineTo(x2, y2);
ctx.lineTo(x3, y3);
...
ctx.lineTo(xn, yn);
ctx.stroke();
```

Each path has its own color and thickness.
The colors are defined as follows:

- **Background** (black) → `#000000`
- **Foreground** (white) → `#FFFFFF`
- **Border** (gray) → `#808080`

7.6.3 Flood fill

The flood fill algorithm is a simple implementation with a stack data structure.

Algorithm 1 Flood Fill

Input: The pixel grid `pixels`, the fill color `fill`, the clicked pixel `clicked`
Output: The processed pixels
if $\text{clicked}_{\text{color}} = \text{fill}$ **then**
    return
stack ← []
while ¬stack.empty() **do**
    point ← stack.pop()
    if $\text{point}_x < 0$ **or** $\text{point}_y < 0$ **or** $\text{point}_x \geq \text{pixels}_{\text{width}}$ **or** $\text{point}_y \geq \text{pixels}_{\text{height}}$ **then**
        continue
    if $\text{point}_{\text{color}} \neq \text{clicked}_{\text{color}}$ **then**
        continue
    $\text{pixels}_{x,y} \leftarrow \text{fill}$
    stack.push($\{x : \text{point}_x + 1, y : \text{point}_y\}$)
    stack.push($\{x : \text{point}_x - 1, y : \text{point}_y\}$)
    stack.push($\{x : \text{point}_x, y : \text{point}_y + 1\}$)
    stack.push($\{x : \text{point}_x, y : \text{point}_y - 1\}$)

7.6.4 Subpixel rendering and antialiasing

The trimap drawing, due to its purpose, shall not have antialiasing and subpixel rendering. These features will create a smooth grayscale border, inserting some pixels in the wrong category of the trimap.

I was not able to fully remove those features, however, some of the following procedures may still enhance performance and quality of the trimap.

Trimap CTX

```
ctx.imageSmoothingEnabled = false
```

Trimap canvas CSS properties

```
#trimap {
    image-rendering: pixelated;
    image-rendering: crisp-edges;
    image-rendering: optimizeSpeed;
}
```

7.6.5 Undo feature

Everything drawn onto the canvas, meaning each line and fill operations, is stored in a stack data structure. The user presses CTRL+Z, the last operation is popped from the stack and everything is drawn again.

This feature is not optimized, so if a few flood fills are made the undo feature can get slow, but it is not usually a problem.

7.6.6 Design

7.7 Dependencies

Here's a list of all the libraries used within the project

Dependency table (matting-lib)			
Name	**Description**	**Version**	**Features**
opencv	Image processing	0.78.0	
image	Image processing	0.24.5	webp, webp-encoder

Dependency table (matting-cli)			
Name	**Description**	**Version**	**Features**
clap	CLI parameters	4.0.29	derive
csscolorparser	CSS color parsing	0.6.2	
chrono	Timestamps	0.4.23	
matting-lib	Matting library	(local)	

Dependency table (matting-web)			
Name	**Description**	**Version**	**Features**
warp	Web server	0.3.3	
futures	Multi threading	0.3.25	
tokio	Multi threading	1	full
csscolorparser	CSS color parsing	0.6.2	
clap	CLI parameters	0.4.29	derive
lazy_static	Lazily evaluated statics	1.4.0	
tempfile	Temporary files	3.4.0	
matting-lib	Matting library	(local)	

7.8 Logging system

7.8.1 Environmental variables

The logging level and logging style can be set by exporting the `MATTING_LOG` and `MATTING_LOG_STYLE` environmental variables. The variables represent the log level and the log style respectively.

The `MATTING_LOG` variable may assume the following values

- `error` : Designates very serious errors.
- `warn` : Designates hazardous situations.
- `info` : Designates useful information.
- `debug` : Designates lower priority information.
- `trace` : Designates very lower priority, often extremely verbose, information.

See [2] for more information and more options.

The `MATTING_LOG_STYLE` variable may assume the following values

- `auto` : Will attempt to print style characters if possible.
- `always` : Will always print style characters.
- `never` : Will never print style characters.

See [3] for more information.

7.8.2 matting-cli

This program only uses the `info` , `warn` and `error` log levels.

The `--verbose` flag automatically sets `MATTING_LOG` to `info` .

7.8.3 matting-web

This program logs to the `info` , `debug` , `warn` and `error` levels.

The default level is set to `info` .

7.9 Flux Diagrams and Error handling

7.9.1 Matting CLI

The following diagram shows the flux of the `matting-cli` executable.

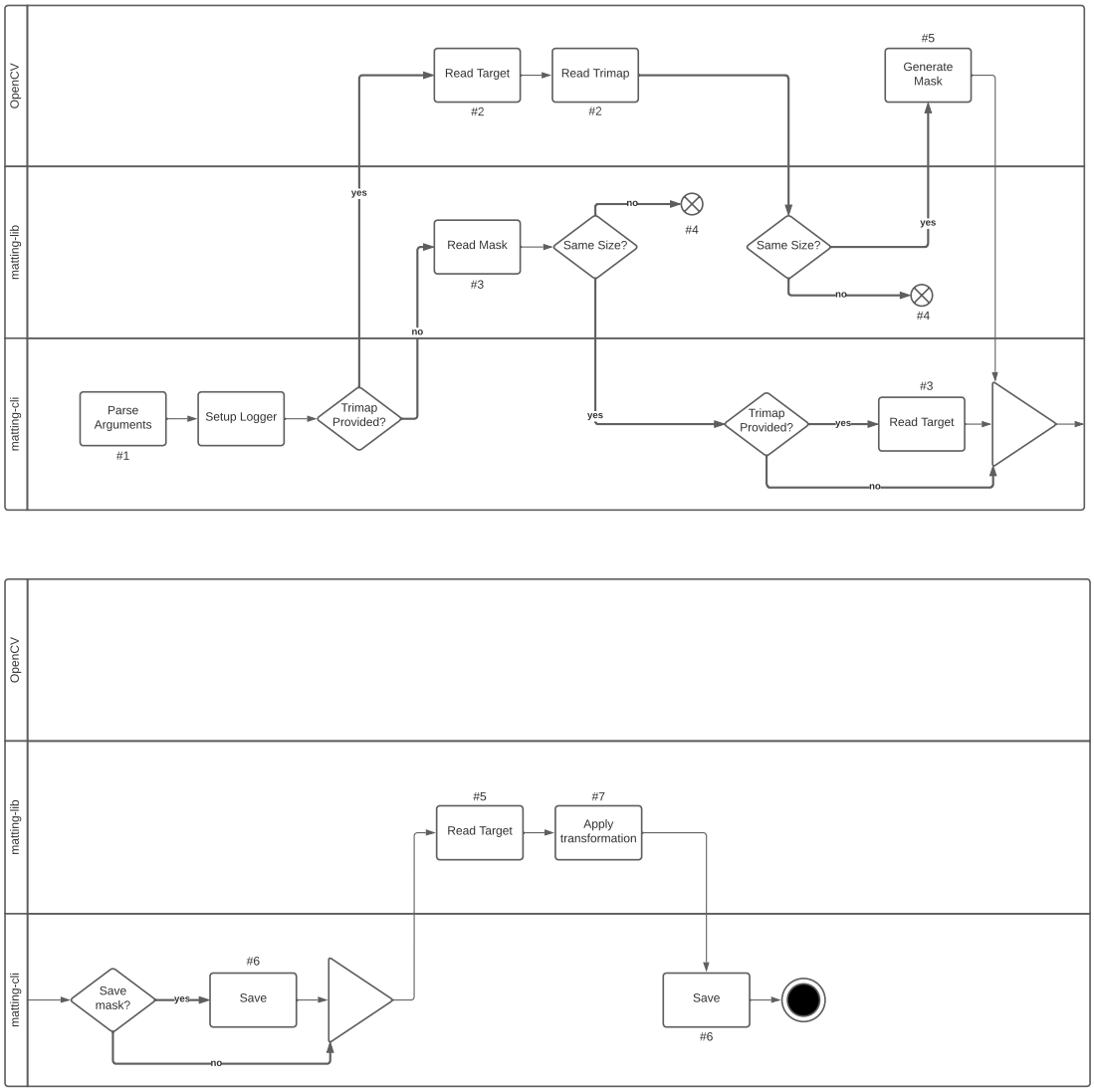


Figure 11: CLI Swimlane

The numbered labels (#) represent the possible errors at the given stage.

#1 Wrong CLI parameters or invalid combination.

#2 (`OpenCV`) The image file is corrupt or could not be read. The file may not exist or the executable may not have the required permission.

#3 (`matting-lib`) The image file is corrupt or could not be read. The file may not exist or the executable may not have the required permission.

#4 The two images are not of the same size.

#5 The mask could not be generated. This is most likely a hard core dump.

#6 The image could not be saved. The path may not exist or the executable may not have the required permission.

#7 The color is invalid and could not be parsed.

7.9.2 Matting WEB

Thw following diagram shows the flux of the `matting-web` executable.

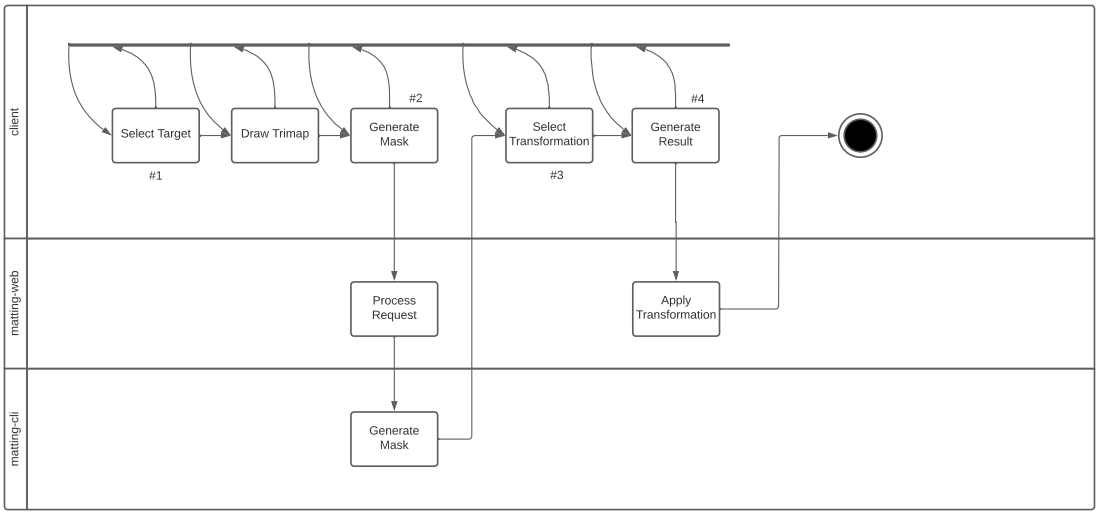


Figure 12: WEB Swimlane

It is important to note that at every stage, the user is able to go back to any previous stage.

The numbered labels (#) represent the possible errors at the given stage.

#1 The target is not a valid image, or it is bigger than 2MiB.

#2 The mask could not be generated. This is most likely a BAD_REQUEST or INTERNAL_SERVER_ERROR.

#3 The image size is not the same as the target.

#4 The result could not be generated. This is most likely a BAD_REQUEST or INTERNAL_SERVER_ERROR

8 Structure

8.1 mandate

The `mandate` folder contains all the documents regarding the project (documentation, abstract and diaries).

8.2 matting-lib

The `matting-lib` folder.

8.3 matting-cli

The `matting-cli` folder.

8.4 matting-web

The `matting-web` folder.

8.5 assets

The `assets` folder contains example images and resources to use the project. These files can be used for the testing protocol.

9 Tests

9.1 Testing protocol

9.1.1 CLI Tests

Requirements: The following tests must be executed are satisfying the following requirements. The resources can be found in the project source code. Tests must be executed in order.

1. The compiled executable of the matting-cli software must be placed in a folder specified in the PATH environmental variable (e.g. `/usr/local/bin` or `/usr/bin`).
2. The matting-cli file must be an executable. It should be enough to run chmod +x <file>.
3. The user from which the tests will be executed must have permission to execute the program.
4. A valid `target` image (`target.jpg`).
5. A valid `trimap` image with the same size of the target image (`trimap.png`).
6. A valid `background` image with the same size of the target image (`background.jpg`).
7. The user must have the permission to read each resource used and write files to the specified locations.
8. All the resources needed must be in the same directory as the one where the caller executed the commands.
9. An image viewer with resizing capabilities along with a graphical environment.
10. The command `file`.

Tests:

Test-00 | Verbose output	
Reference	Req-05
Steps	1. Execute the following command: `matting-cli -i target.jpg --trimap trimap.png --save-mask mask.png -o out.png --fill red --verbose`

Result	The output written to the STDOUT must look like this:
	`[<time> INFO matting_cli] Reading target image...`
	`[<time> INFO matting_cli] Done! [<time>]`
	`[<time> INFO matting_cli] Reading trimap image...`
	`[<time> INFO matting_cli] Done! [<time>]`
	`[<time> INFO matting_cli] Generating soft mask...`
	`[<time> INFO matting_cli] Done! [<time>]`
	`[<time> INFO matting_cli] Reading target image...`
	`[<time> INFO matting_cli] Done! [<time>]`
	`[<time> INFO matting_cli] Saving soft mask to mask.png`
	`...`
	`[<time> INFO matting_cli] Done! [<time>]`
	`[<time> INFO matting_cli] Filling background with color`
	`...`
	`[<time> INFO matting_cli] Done! [<time>]`
	`[<time> INFO matting_cli] Saving output to out.png...`
	`[<time> INFO matting_cli] Done! [<time>]`

Test-01 | Help message	
Reference	Req-00

Steps	1. Execute the following command:
	`matting-cli -h`
	`matting-cli --help`

Result	The output written to the STDOUT must look like this:
	`Matting CLI`
	
	`Usage: matting-cli [OPTIONS] --target <TARGET>`
	`<--mask <MASK>|--trimap <TRIMAP>>`
	
	`Options:`
	`-i, --target <TARGET> Target image`
	`--mask <MASK> Background mask image`
	`--trimap <TRIMAP> Trimap image`
	`--save-mask <SAVE_MASK> Save mask path`
	`-o, --output <OUTPUT> Output image`
	`-f, --fill <FILL> Fill background action`
	`-t, --transparent Transparent background action`
	`-r, --replace <REPLACE> Replace background action`
	`--verbose Verbose flag`
	`-h, --help Print help information`
	`-V, --version Print version information`

Test-02 | Empty parameters	
Reference	Req-00_0, Req-00_1, Req-00_2, Req-00_3, Req-00_4
Steps	1. Execute the following command: `matting-cli`

Result	The output written to the STDOUT must look like this:
	`error: The following required arguments were not:`
	`provided:`
	`--target <TARGET>`
	`<--mask <MASK>|--trimap <TRIMAP>>`
	
	`Usage: matting-cli --target <TARGET>`
	`<--mask <MASK>|--trimap <TRIMAP>>`
	
	`For more information try '--help'`

Test-03 | Empty action	
Reference	Req-00
Steps	1. Execute the following command: `matting-cli --target target.jpg --trimap trimap.png`
Result	The output written to the STDOUT must be empty and no file must be generated.

Test-04 | Mask generation	
Reference	Req-00_4

Steps	1. Delete the previously generated mask:
	`rm mask.png`
	2. Execute the following command:
	`matting-cli --target target.jpg --trimap trimap.png --save-mask mask. png`
	3. List the files in the current directory:
	`ls .`

Result	The output written to the STDOUT must be empty and a file `mask.png` must appear in the directory.

Test-05 | Replace background feature	
Reference	Req-00_5

Steps	1. Execute the following commands:
	`matting-cli -i target.jpg --trimap trimap.png --replace background.jpg -o res1.jpg`
	`matting-cli -i target.jpg --mask mask.png --replace background.jpg -o res2.jpg`
	2. Open `res1.jpg` and `res2.jpg` with an image viewer.

Result	Both images must present the background image behind the subject.

Test-06 | Fill background feature	
Reference	Req-00_6

Steps	1. Execute the following commands:
	`matting-cli -i target.jpg --mask mask.png --fill "red" -o res1.jpg`
	`matting-cli -i target.jpg --mask mask.png --fill "#FF0000" -o res2.jpg`
	`matting-cli -i target.jpg --mask mask.png --fill "rgb(255,0,0)" -o res3.jpg`
	2. Open `res1.jpg`, `res2.jpg` and `res4.jpg` with an image viewer.

Result	Both images must present a red colored background behind the subject.

Test-07 | Transparent feature	
Reference	Req-00_7

Steps	1. Execute the following commands:
	`matting-cli -i target.jpg --trimap trimap.png --transparent -o res1.png`
	`matting-cli -i target.jpg --mask mask.png --transparent -o res2.png`
	2. Open `res1.png` and `res2.png` with an image viewer.

Result	Both images must present a transparent background behind the subject.

Test-08 | Output with format	
Reference	Req-01

Steps	1. Execute the following commands:
	`matting-cli -i target.jpg --mask mask.png --fill "red" -o format.webp`
	`matting-cli -i target.jpg --mask mask.png --fill "red" -o format.jpg`
	`matting-cli -i target.jpg --mask mask.png --fill "red" -o format.png`
	2. Print information about each of the resulting images:
	`file format.webp`
	`file format.jpg`
	`file format.png`

Result	Each information line must specificy the respective image format. (e.g. `.jpg` → JPG)

Test-09 | Transparency and JPG format	
Reference	None
Steps	1. Execute the following command: `matting-cli -i target.jpg --mask mask.png --transparent -o res.jpg` 2. Open `res.jpg` with an image viewer.

Result	A message to the STDOUT must warn about the usage of the JPG format with transparency. The image must have a black background behind the subject.
	`[<date> WARN matting_cli] The image has an alpha channel but the format is not PNG.`
	`[<date> WARN matting_cli] It is advised to change the output format to ".png"`

Test-10 | Mask default name	
Reference	Req-04
Steps	1. Execute the following command: `matting-cli -i target.jpg --trimap trimap.png --save-mask` 2. List the files in the directory
Result	The mask must have been saved in the current directory with the name `mask_[timestamp].png`.

Test-11 | Output default name	
Reference	Req-04

Steps	1. Execute the following command:
	`matting-cli -i target.jpg --mask mask.png --transparent`
	2. List the files in the current directory:
	`ls .`

Result	The mask must have been saved in the current directory with the name `target_[timestamp].png`.

Test-12 | Wrong size images	
Reference	Req-02

Steps	1. Copy `trimap.png`, `mask.png` and `background.jpg` to `trimap2.png`, `mask2.png` and `background2.jpg` respectively.
	2. Open `trimap2.png`, `mask2.png` and `background2.jpg` and resize them to a smaller dimension.
	3. Save the files.
	4. Execute the following commands:
	`matting-cli -i target.jpg --mask mask2.png`
	`matting-cli -i target.jpg --trimap trimap2.png`
	`matting-cli -i target.jpg --mask mask.png --replace background2.jpg`

Result	Each of the matting commands must print en error indicating that the sizes of the images are not the same.
	`[<time> ERROR matting_cli] An error occured: The mask and target image must be of the same size.`
	`[<time> ERROR matting_cli] An error occured: The trimap and target image must be of the same size.`
	`[<time> ERROR matting_cli] An error occured: The replacement and target image must be of the same size.`

9.1.2 Website Tests

Requirements: The following tests must be executed are satisfying the following requirements. The resources can be found in the project source code. Tests must be executed in order.

1. The compiled executable of the matting-cli software must be placed in a folder specified in the PATH environmental variable (e.g. `/usr/local/bin` or `/usr/bin`).
2. The matting-cli file must be an executable. It should be enough to run chmod +x <file>.
3. A browser.
4. The user from which the tests will be executed must have permission to execute the program.
5. A valid `target` image (`target.jpg`).
6. A valid `trimap` image with the same size of the target image (`trimap.png`).
7. A valid `background` image with the same size of the target image (`background.jpg`).
8. A running server instance of the web application.
9. The server must have the permission to read each resource used and write files to the specified locations.
10. The server must have read and write permissions to `/tmp`.

Tests:

Test-13 | Trimap mask generation	
Reference	Req-03
Steps	1. Upload the target image to the website 2. Draw the trimap with the given tools 3. Press the generate mask button
Result	A loading animation must appear on the website. After some time the corresponding alpha matte mask must appear, replacing the loading animation.

Test-14 | Save mask feature	
Reference	Req-03
Steps	1. Press the download button under the generated mask image 2. Open the image from the file system with a viewer
Result	The image must be saved and contain the correct image.

Test-15 | Transparency feature	
Reference	Req-03
Steps	1. Select the transparent transformation option 2. Click the apply transformation button
Result	A loading animation must appear on the website. After some time the corresponding alpha matte mask must appear, replacing the loading animation. The result must be the target image with a transparent background.

Test-16 | Fill background feature	
Reference	Req-03
Steps	1. Select the fill background transformation option 2. Select a color with the color input 3. Click the apply transformation button
Result	A loading animation must appear on the website. After some time the corresponding alpha matte mask must appear, replacing the loading animation. The result must be the target image with the selected color as a background.

Test-17 | Replace background feature	
Reference	Req-03
Steps	1. Select the transparent transformation option 2. Select the background image with the file input 3. Click the apply transformation button
Result	A loading animation must appear on the website. After some time the corresponding alpha matte mask must appear, replacing the loading animation. The result must be the target image with the selected image as a background.

Test-18 | Save result feature	
Reference	Req-03
Steps	1. Press the download button under the generated image 2. Open the image from the file system with a viewer
Result	The image must be saved and contain the correct image.

Test-19 | Wrong size replacement	
Reference	Req-03
Steps	1. Select the replacement background option as a transformation 2. Upload the `background2.jpg` image
Result	An alert message must appear saying that the background must be of the same size as the target image.

Test-20 | Core dump	
Reference	Req-03
Steps	1. Draw the trimap such that it all black 2. Press the generate mask button
Result	The server must not crash and keep running

Test-21 | Image size cap	
Reference	Req-03
Steps	1. Upload an image with size bigger than 2MiB as the target image
Result	An alert must appear saying that the image size cannot exceed 2MiB of size.

Test-22 | Empty parameters	
Reference	Req-03
Steps	1. Execute the following command: `matting-web`

Result	The output written to the STDOUT must look like this:
	`error: the following required arguments were not provided:`
	`--www <WWW>`
	
	`Usage: matting-web --www <WWW>`
	
	`For more information, try '--help'.`

Test-23 | Logging	
Reference	Req-05
Steps	1. Execute the following command: `matting-web -w <path/to/www>`

Result	The output must look like this and keep logging.
	`[<time> INFO matting_web::handler] Server starting`
	`[<time> INFO warp::server] Server::run; addr =0.0.0.0:8080`
	`[<time> INFO warp::server] listening on http ://0.0.0.0:8080`

9.2 Test results

ID	Result	Note
Test-00	Passed	The output looks like the given one.
Test-01	Passed	The output looks like the given one.
Test-02	Passed	The output looks like the given one.
Test-03	Passed	Nothing was done.
Test-04	Passed	Empty STDOUT and correct mask generated.
Test-05	Passed	The background is correct.
Test-06	Passed	The background is red.
Test-07	Passed	The background is transparent.
Test-08	Passed	Every format is correct.
Test-09	Passed	A warning message is given.
Test-10	Passed	The mask is saved with the correct name.
Test-11	Passed	The mask is saved with the correct name.
Test-12	Passed	Error messages are given.
Test-13	Passed	The mask is correctly generated.
Test-14	Passed	The image is correctly saved.
Test-15	Passed	The result has a transparent background.
Test-16	Passed	The result has a colored background.
Test-17	Passed	The result has an image as a background.
Test-18	Passed	The image is correctly saved.
Test-19	Passed	An alert message is displayed.
Test-20	Passed	The server does not crash.
Test-21	Passed	An alert message is displayed.
Test-22	Passed	The output looks like the given one.
Test-23	Passed	The output looks like the given one.

10 Conclusion

10.1 Future development

There are lots of features that could be added to the application and improvements that could be made.

1. **GPU Acceleration**: The code remove, replace or fill the background of an image given its mask is executed pixel-by-pixel by the CPU. It would be incredibly faster if it could use the GPU instead, by using a library like *Vulkan*. The implementation would require writing a simple compute shader which reads the mask and the original image from a buffer.
2. **Better website**: The website design could be improved. It is not fully responsive and it certainly could have a better design overall.
3. **Trimap upload**: It could be useful if the user was able to directly upload a trimap for a given image, rather than having to always draw it.
4. **Automatic cropping**: When a user upload an image replacement for the background. It would be useful if it could be cropped to the correct size directly within the website.
5. **Website format handling**: Unlike the CLI version, the web application does not handle different formats. Everything is converted into a PNG and saved as one. The user has to manually convert the resulting image into another format.
6. **Optimized undo**: The undo feature gets slow as more flood fills are made.

10.2 Personal conclusions

I really enjoyed this project for its semplicity and usefulness.

List of Figures

References

[1] Paolo Bettelini. *trimap-matting*. 2022. URL: https://github.com/paolobettelini/trimap-matting.
[2] Docs.rs. *Crate env_logger*. URL: https://docs.rs/env_logger/latest/env_logger/#enabling-logging.
[3] Docs.rs. *Crate env_logger*. URL: https://docs.rs/env_logger/latest/env_logger/#disabling-colors.
[4] Mozilla. *<color>*. 2022. URL: https://developer.mozilla.org/en-US/docs/Web/CSS/color_value#format_syntax.
[5] OpenCV. *Alpha Matting*. 2022. URL: https://docs.opencv.org/4.x/d4/d40/group__alphamat.html.
[6] OpenCV. *cv::Mat Class Reference*. 2022. URL: https://docs.opencv.org/4.x/d3/d63/classcv_1_1Mat.html.
[7] OpenCV. *OpenCV modules*. 2022. URL: https://docs.opencv.org/4.x.
[8] Mozilla Research. *Rust*. 2010. URL: https://www.rust-lang.org/.
[9] Wikipedia contributors. *Matte (filmmaking) — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Matte_(filmmaking)&oldid=1127451547. [Online; accessed 18-January-2023]. 2022.

Glossary

FFI Foreign Function Interface. 15

trait A trait in Rust defined shared behavior among structures. 15