Trimap Matting

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

Paolo Bettelini

Contents

1 Introduction

1.1 Abstract

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**: 2022-09-29
- **Deadline**: 2022-12-07

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

3 CLI

3.1 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/debug`. In order to make this executable globally available we can move it into a folder in the $PATH enviroment variable, such as `/usr/bin`. We may also modify the executable file name to change its invokation name.

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

We executable can now be invoked by just writing

```
$ matting-cli
```

3.2 Usage

The followings 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.

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 timestmap.

Note: the argument of `--color` can be any valid CSS color. See [2] 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.

3.3 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 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 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]
```

4 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[6] 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

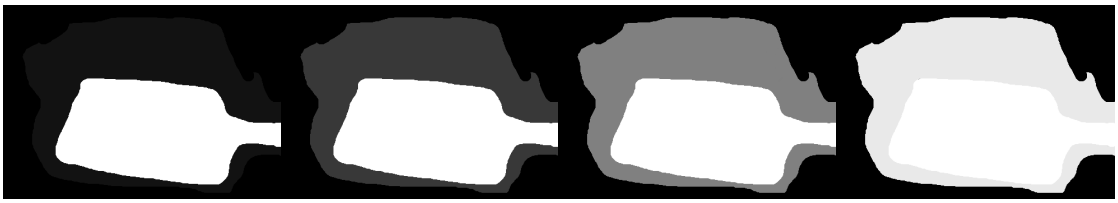
Figure 6: Replacement

5 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[5]. The library also contains a module about *alpha matting*, which contains a function to generate alpha mattes given a trimap [3].

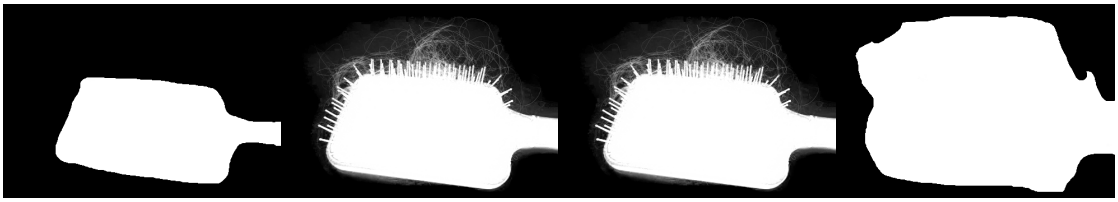
5.1 Trimap colors

As stated in the last section, the gray color represents the unknown pixels. The library however, accepts a greyscale trimap. The question is wheter 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

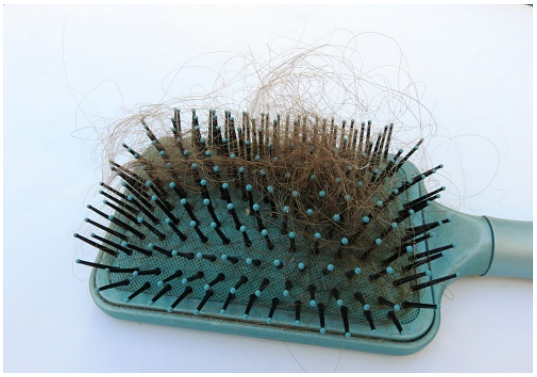


Figure 9: Hairy brush

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.

6 Implementation

6.1 OpenCV fundamental types

The fundamental OpenCV type used in this project is the **Mat**[4] 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.

6.2 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 pratices, 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.

6.3 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**.

6.4 Matting library

The core of this project is certinaly the alpha matting algorithm. In order to develope 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
```

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

The `error` module exports error management utils

```rust
#[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;
```

6.5 Website

List of website features

The images uploaded cannot be bigger than 2MiB

If a transformation is applies whilst the mask is loading, the site will wait until the mask is ready.

The trimap panting tool aren't enabled until an image is uploaded

The mask download button and the transformation selector aren't enabled until a mask has been generated or uploaded.

The website will check wheter the image and the mask are of the same size

The website will check wheter the image, mask and replacement image are of the same size

The website will handle error responses from the server with an alert message

The header dropdown is responsive and needs to be toggled when the page is small.

While drawing you can press ctrl+z to undo

6.5.1 Painting mechanics

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

6.5.2 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
$\text{stack} \leftarrow []$
while ¬stack.empty() **do**
    $\text{point} \leftarrow \text{stack.pop}()$
    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
    $\text{pixels}_{x,y} \leftarrow \text{fill}$
    $\text{stack.push}(\{x : \text{point}_x + 1, y : \text{point}_y\})$
    $\text{stack.push}(\{x : \text{point}_x - 1, y : \text{point}_y\})$
    $\text{stack.push}(\{x : \text{point}_x, y : \text{point}_y + 1\})$
    $\text{stack.push}(\{x : \text{point}_x, y : \text{point}_y - 1\})$

6.5.3 Undo feature

Everything drawn onto the canvas, meaning each line and fill operations are stored in a stack data structure.

List of Figures

References

[1] Paolo Bettelini. *trimap-matting*. 2022. URL: https://github.com/paolobettelini/trimap-matting.
[2] Mozilla. *<color>*. 2022. URL: https://developer.mozilla.org/en-US/docs/Web/CSS/color_value#format_syntax.
[3] OpenCV. *Alpha Matting*. 2022. URL: https://docs.opencv.org/4.x/d4/d40/group__alphamat.html.
[4] OpenCV. *cv::Mat Class Reference*. 2022. URL: https://docs.opencv.org/4.x/d3/d63/classcv_1_1Mat.html.
[5] OpenCV. *OpenCV modules*. 2022. URL: https://docs.opencv.org/4.x.
[6] 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. 9

trait A trait in Rust defined shared behavior among structures. 9