Master’s thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Painting hand-drawn images using mobile devices

Jan Brejcha
Open Informatics, Software Engineering

2014
Supervisor: Ing. Daniel Sýkora, Ph.D.

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačové grafiky a interakce

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Jan Brejcha**

Studijní program: Otevřená informatika (magisterský)
Obor: Softwarové inženýrství

Název tématu: **Obarvování ručních kreseb na mobilních zařízeních**

Pokyny pro vypracování:

Navrhněte a implementujte uživatelsky přívětivou aplikaci, která umožní běžnému uživateli iPadu vybarvit obrázek nakreslený rukou na papíře a výsledek vytisknout na barevné tiskárně. Aplikace umožní uživateli pořídit fotografii jím vytvořené předlohy, kterou následně natočí do vodorovné polohy a provede potřebné jasové korekce. Dále aplikace uživateli nabídne přednastavenou paletu barev, z níž bude možné vybrat požadovaný odstín a pomocí hrubých tahů prstem po dotykovém displeji specifikovat, na která místa kresby zvolenou barvu nanést. Finálním krokem bude vlastní obarvení kresby s možností uložení výsledku do paměti nebo tisku na bezdrátově připojené barevné tiskárně. Pro výpočet vodorovné polohy využijte gyroskopických senzorů a pro barvení implementuje algoritmus LazyBrush [1] s využitím knihovny GridCut [2]. Výslednou aplikaci otestujte v reálných podmínkách na několika uživatelích a pokuste se ji vyladit tak, aby byla připravena k prodeji na platformě Apple store.

Seznam odborné literatury:

[1] Sýkora et al.: LazyBrush: Flexible Painting Tool for Hand drawn Cartoons, Computer Graphics Forum 28(2):599-608, 2009.

[2] Jamriška et al.: Cache-efficient Graph Cuts on Structured Grids, Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 3673–3680, 2012.

Vedoucí: Ing. Daniel Sýkora, Ph.D.

Platnost zadání: do konce letního semestru 2014/2015

České vysoké učení technické v Praze
Fakulta elektrotechnická
L.S.

[illegible]
prof. Ing. Jiří Žára, CSc.
vedoucí katedry

[illegible]
prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 19. 2. 2014

/ Declaration

Abstrakt / Abstract

Tato diplomová práce se zabývá implementací iOS aplikace pro obarvování ručně kreslených kreseb vyfocených vestavěným fotoaparátem. Pro obarvování kreseb byl studován, implementován a testován algoritmus LazyBrush. Analýza požadavků, návrh uživatelského rozhraní a testování použitelnosti byly provedeny ve spolupráci s plánovanou cílovou uživatelskou skupinou–dětmi.

Pro dosažení lepších vizuálních výsledků obarvení bylo implementováno předzpracování obrazu. Obraz vyfocený vestavěným fotoaparátem není vhodný pro okamžité obarvování–obsahuje pozadí, obraz může být otočen a perspektivně zkreslen. Tyto jevy jsou odstraněny pomocí rozpoznání hran papíru, zpětné perspektivní transformace užitím homografie, ořezu a filtrace obrazu. Tímto je náročnost pracovního postupu zjednodušena pouze na vyfotografování–obarvení–uložení, díky čemuž je aplikace dobře použitelná i dětmi a neodbornými uživateli.

Překlad titulu: Obarvování ručních kreseb na mobilních zařízeních

This thesis deals with the implementation of iOS application for colorization of hand-drawn images captured with the built-in camera. For image colorization the LazyBrush algorithm has been studied, implemented and tested. Requirements analysis, user interface design and usability testing was made in collaboration with intended target group of users—children.

To obtain better visual results of colorization image preprocessing of captured image was implemented. Image captured with the built-in camera is not ideal for immediate colorization—it contains background, the picture may be rotated and perspectively distorted. These phenomena are removed through registration of paper edges, reverse perspective transformation using homography, image cropping and image filtration. This reduces the complexity of the workflow just to capture—colorize—save which makes the application usable easilly for children and unprofessional users.

Contents

Tables / Figures

Chapter 1
Introduction

Interactive digital image manipulation is becoming more and more popular. There is an increasing demand for improving or retouching photos not only on personal computers but on tablets and smartphones as well. As hardware performance of tablets and smartphones is increasing, new applications of computer graphics and interactive image manipulation are possible. Since tablets are controlled by touching the screen with finger or stylus, some applications seems to be more natural on tablet than on PC.

Mobile devices like tablets and smartphones are mainly used for content consumption or entertainment. For children there is a variety of applications allowing them to colorize simple colorings — which are in many cases unchangeable.

Example of such application for iOS can be Coloring Book by Toy Maker on Figure 1.1. Children can colorize prepared vector cartoons with a bucket or a brush tool. Application Coloring for Kids on Figure 1.2 has a story with song for each coloring. Children can use only the brush tool with several predefined sizes and colors; no bucket tool is present in this app. In practice it is difficult to color the coloring precisely. Another application called Coloring Book by TabTale on Figure 1.3 works on similar principle like the previous ones. Children can choose from prepared images, choose colors and touch inside the regions in the image that are then colored with the chosen color.

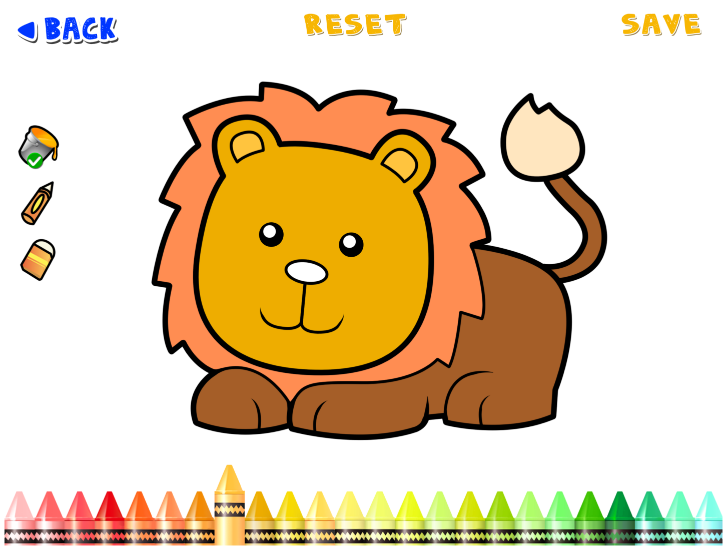


Figure 1.1. Coloring Book by ToyMaker for kids

The fact that the colorings in many applications are unchangeable makes the application boring after all images were colorized several times. Natural idea to solve this problem is to allow children to load their own hand-drawn images to be colorized. This possibility engages children creativity more. This approach can be also a new method for easy bitmap image colorization which is still in common tools neither really quick to use nor user-friendly.

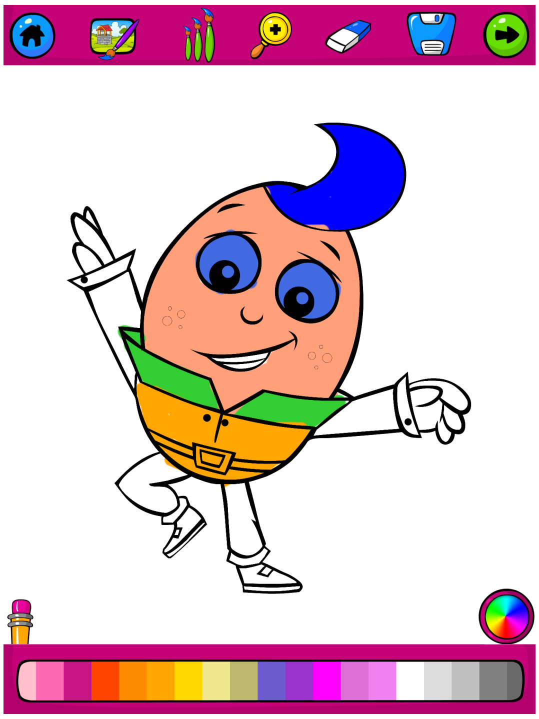


Figure 1.2. Coloring for Kids by Internet Design Zone

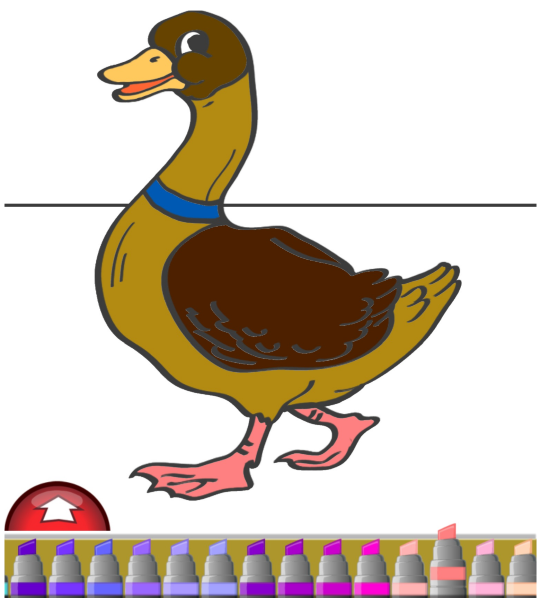


Figure 1.3. Kids Coloring Book by Tab-Tale

Chapter 2
Our Goal

2.1 LazyBrush Application on iOS

The main intention is to design and implement iOS application capable of colorization of photographed drawings. Main features of the applications are following. With the application, a new photo can be taken or an existing one can be used. Then the brush strokes (called scribbles) on top of the image are drawn. Each scribble consists of one or several brush strokes with particular color; each scribble has only one color. The scribble defines the region which should be colored with the color of that scribble. As the user draws the scribbles, the system colors the image interactively with particular colors. The final colored image can be saved into image library or printed on AirPrint friendly printer directly from the application.

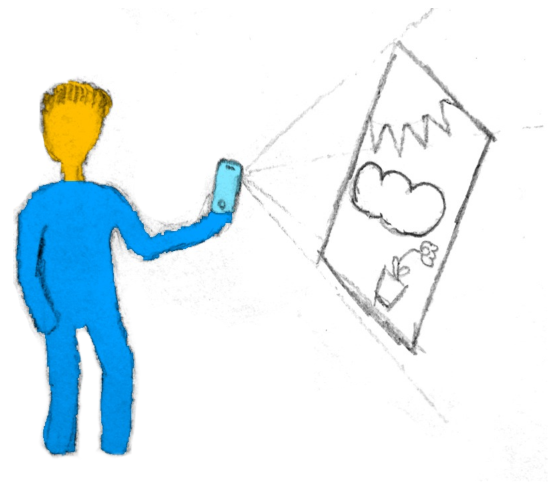
To make the process of taking a photo and colorizing a picture as simple as possible, the image shall be preprocessed, cropped, perspective distortion shall be removed, the outlines shall be highlighted and the grey background shall be removed to obtain white background of the drawing. This application is though considered to be colorization application only — no other advanced image editing techniques shall be implemented. For future work, there will be a lot of opportunities to implement other tools for image editing to provide comprehensive image editing application that takes advantage of sophisticated image coloring not only for kids but for advanced users as well.

2.2 Application Workflow

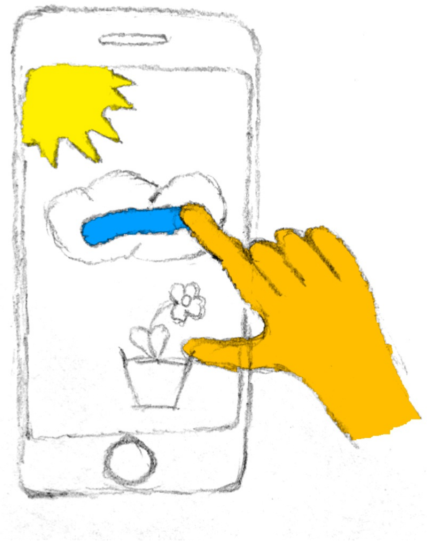
The application is desired to be coloring application for children. The desired workflow is illustrated as a storyboard drawn by hand with soft pencil and colorized with Lazy-Brush iOS application on Figure 2.1. The child draws an image with his pencil or pen on paper (in school, at home or during his free time, see Figure 2.1 a), then he takes a photo of the drawing (preferably with our application, see Figure 2.1 b) and then uses the application to color the drawing (on Figure 2.1 c) and to save (or send by email or message to his relatives or friends) or to print on an AirPrint friendly printer (see Figure 2.1 d). Children are not required to be able to read, but it is assumed that the users of the application are not entirely new to iOS and understand basic navigation in iOS — they are able to run any application and know their possibilities of device controlling with gestures.

Nevertheless the main planned target user group are children, other users than children are also possible. We just do not design the application for their concrete needs. For example, these users can be people who like to draw greyscale drawings and want to see how it looks like when the images are colored — people who like drawing and art, or people who could use it for doing their work — like designers or architects to color quick sketches. For them this application could be considered as a proof-of-concept and, in case it is beneficial for them, other application based on this one can be created to suit their needs better.

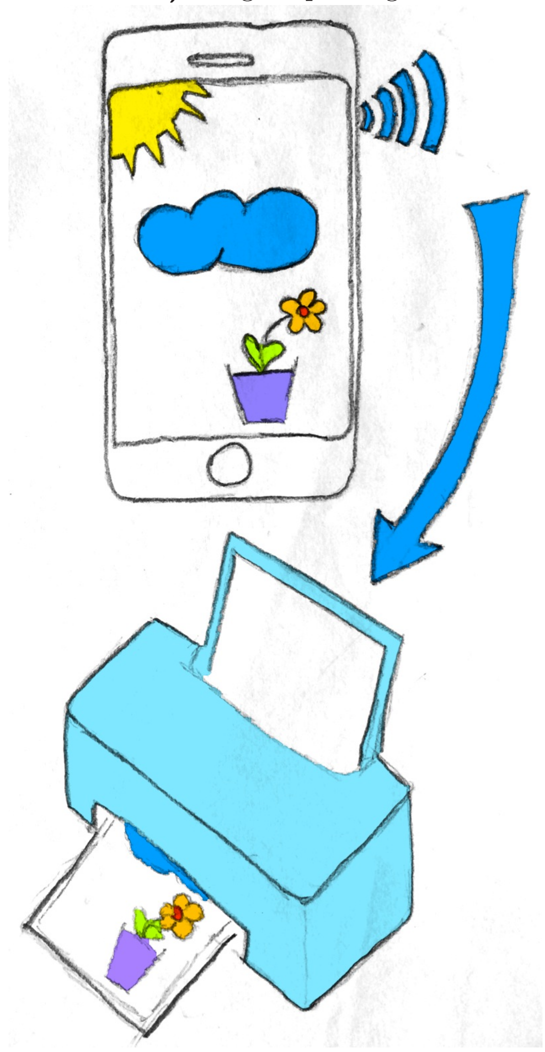
a) Image drawing



b) Image capturing



c) Scribble drawing, colorization



d) Image printing

Figure 2.1. Storyboard of application workflow

Chapter 3
Related Work

Related work is divided into two sections. Related research about the image segmentation and the automated image colorization was studied. An overview of these research topics can be found in Section 3.1. Since real application is going to be implemented, existing applications in the field of applications for drawing and sketching, coloring books for children and other image processing applications were searched. An overview of these applications on the iOS platform is available in Section 3.2.

3.1 Related Research

One of the first algorithms developed for colorizing black and white pictures was described by Lieberman [1] and was based on flood-fill algorithm. Smith [2] described the tint fill algorithm for colorizing areas with shaded boundaries and Fishkin and Barsky [3] presented three algorithms for filling antialiased regions. These methods are limited with the initial idea of flood fill on which they are based. Problematic for these methods is the inability to color complicated patterns or drawings with gappy outlines.

Coloring with usage of color seeds was first described by Horiuchi [4] who used probability relaxation method for improving the search of an RGB vector from an intensity value. Levin et al. [5] proposed method for coloring greyscale photos. Luan et al. [6] proposed technique that significantly lowers the number of user-defined constraints to do the image colorization which is based on repetitive texture patterns. This technique is ideal for coloring photos but is unsuitable for coloring greyscale hand-drawn images because there is a lack of similar repetitive patterns. Very fast and efficient framework for photo and video colorization was described in [7]. This method is based on computing chrominance of the pixels in the image according to its distance from a scribble with known chrominance. It is very efficient and therefore ideal for coloring greyscale photo or video "on the fly", but it is unsuitable for coloring hand-drawn images because they tend to contain black on edges and white other areas — there is a lack of other smooth greyscale shades. Sýkora et al. [8] proposed method for colorization of black and white animations created with cell or paper based technology. However this method is suitable for colorization of old greyscale material, it fails on classic hand-drawn cartoons because it makes an assumption of strong dark closed outlines which are not always present in hand-drawn images. Manga colorization method was proposed by Qu et al. [9]. Their framework analyses both pattern and intensity continuities so that regions with open boundaries or similar patterns can be separated with a single scribble. Segmented regions can be then colored in various ways. This colorization method is ideal for complicated black and white drawings that contains a lot of continuous patterns.

Many image colorization algorithms use image segmentation to obtain areas in the image that are then colored with desired colors. Grady [10] described method using random walker probability calculation to obtain high quality segmentation. Another approach to solve image segmentation uses maxflow/mincut problem formulation from combinatorial optimization. With connection to computer vision, the maxflow/mincut problem was first introduced by Greig et al. [11] to reconstruct original image from noisy image. This problem formulation helped to solve the image segmentation problem which was introduced by Boykov and Jolly [12]. The efficiency of graph cuts was also studied by Boykov et al. [13] who introduced approximate energy minimization algorithm called α-expansion for graph cuts.

Sýkora et al. [14] introduced LazyBrush algorithm which is built on the idea of approximate energy minimization through graph cuts. This algorithm was developed to satisfy various conditions to be able to colorize hand-drawn images quickly and effectively. It avoids problems typical for previously mentioned algorithms like color leakage through gappy outlines so it is ideal for the purpose of the application being implemented.

3.2 Existing Applications

On iOS there is a variety of applications for image manipulation or creation. Only some of current most successful iOS applications for creating or manipulating with images were chosen for this review. There are two main branches of applications we are interested in — the image manipulation applications and the image drawing applications.

Photoshop Touch application from Adobe has several options to select a part of the image and then do some image processing on this part (like bucket coloring, etc.). Lasso, magic wand and the scribble selection (see Figure 3.1) can be used — which is very similar to the selection we use for LazyBrush algorithm. It is a complex application with many other features like working with layers, adjustments and filters.

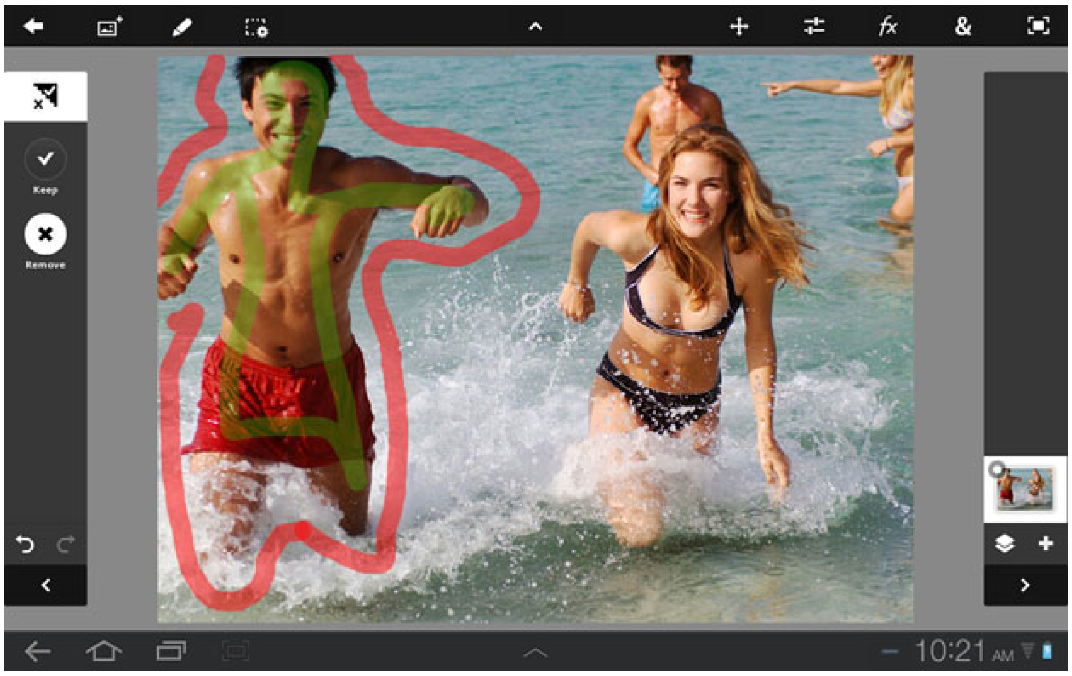


Figure 3.1. Photoshop Touch scribble selection tool. Image source credit: © Future Publishing Limited

Apple's iPhoto is aimed mainly for improving and retouching photos. It cannot select part of a photo and for example color it with specific color. Though user can resize, crop the photo, adjust exposition, color saturation and much more.

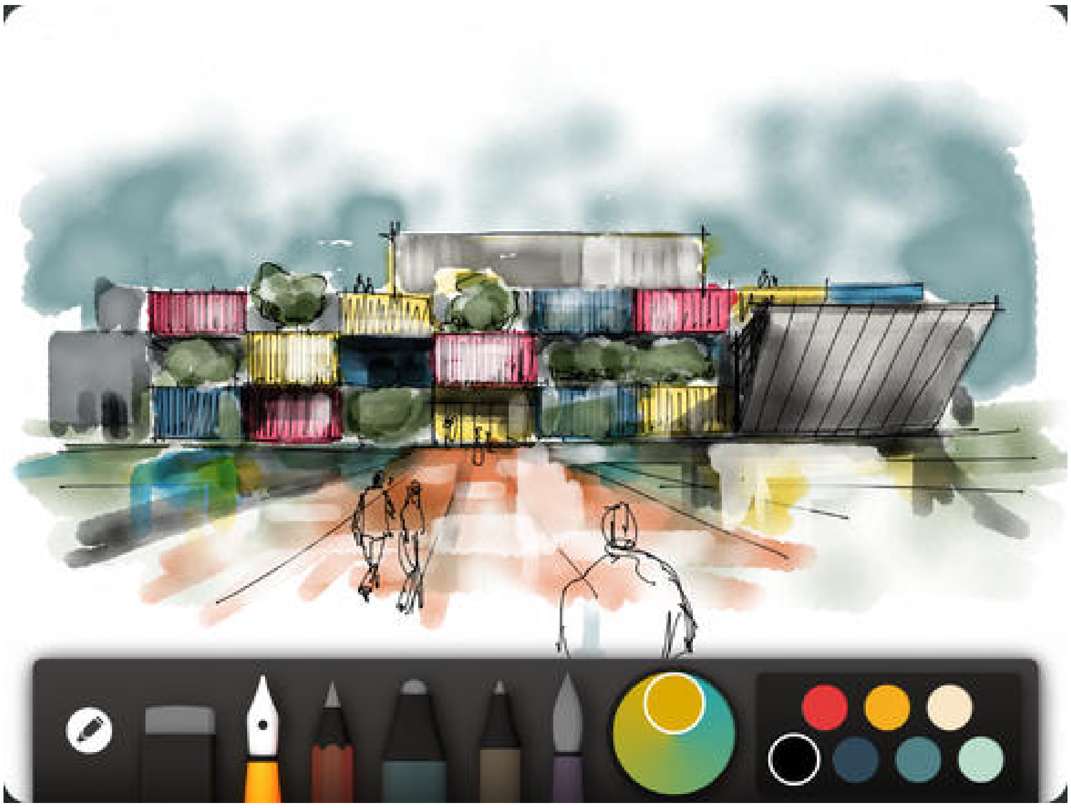


Figure 3.2. Paper from FiftyThree. Image source credit: © FiftyThree

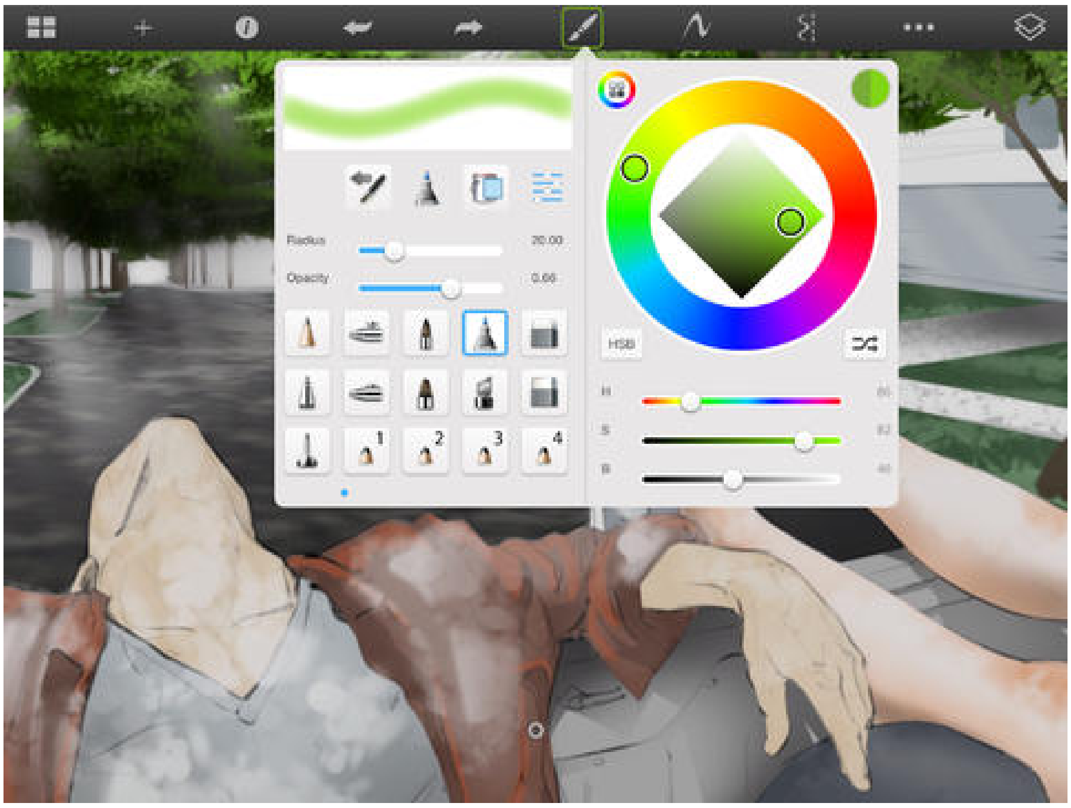


Figure 3.3. SketchBook Pro from Adobe. Image source credit: © Adobe

Besides the applications combining painting and image processing like the Photoshop Touch, there are a lot of applications for drawing with a finger only. One of those applications is the Paper application from FiftyThree. User can draw with his finger very smoothly. The application offers several tools for drawing like fountain pen, a pencil and an outliner. It brings the idea how the painting tool should be implemented — very smooth, responsive with no lags at all.

Another sketching application is SketchBook Pro for iPad created by Adobe. It is an application for people who want to do their sketches on iPad. User draws the sketch with his fingers, and can use some advanced features like layers and layer transformations — rotation, scale, opacity, etc. Besides Paper application, it has more tools — basic primitives such as rectangle, ellipse and straight line. On the other side, this application is not so smooth and responsive during drawing as Paper application. User Interfaces of both Paper and SketchBook Pro application can be compared on Figure 3.2 and 3.3.

There are also smaller single task applications that are capable of editing photos. For example, with the ColorSplash application the photo can be decolorized and then the original colors can be placed back with a brush tool to only some parts of the image.

Application name	painting	editing	target group	price
Photoshop Touch	yes	yes	common users	9.99$
iPhoto	no	yes	common users	4.99$
Paper	yes	no	common users, artists	free
SketchBook Pro	yes	no	artists, designers	4.99$
ColorSplash	no	yes	common users	0.99$

Table 3.1. Comparison of iOS applications

For better understanding what each application is capable of, the table 3.1 was prepared. The only application capable of coloring particular area of a photo is the Photoshop Touch. The other applications mentioned here are useful either for drawing images or image manipulation. None of the application are primarily designed to color bitmap images, though with Photoshop Touch it is possible to do this, thanks to the scribble selection.

Chapter 4
LazyBrush Algorithm

According to analysis in Section 3.1 LazyBrush algorithm by Sýkora, Dingliana and Collins [14] was found to be the most suitable for the needs of our application. This algorithm is designed to satisfy a set of properties which were developed in cooperation with professional ink-and-paint illustrators. The definition of the algorithm with all its properties was taken from [14].

4.1 Ideal Painting Tool

Ideal painting tool is a set of properties defined in [14] to form a tool that will be highly effective for coloring bitmap images. The set of properties for an ideal painting tool is intended as following:

- **Color leakage:** For standard flood-fill algorithms, the color leakage through unclosed outlines is a serious problem. Unclosed outline gaps must be closed manually or automatically with automatic outline joining algorithms. The problem is that these algorithms close all the gaps and thus can accidentally create many small areas which makes colorization more time consuming for the artist. Ideal painting tool has to avoid this problem. See section a) on Figure 4.1.
- **Optimal boundary:** Ideal painting tool finds an optimal boundary according to the previous rule and then fills as much area as possible according to found boundaries. See section b) on Figure 4.1.
- **Connected labelling:** The area that is labelled to be filled with color should be connected (there should not be any disconnected parts). See section c) on Figure 4.1.
- **Soft scribble:** For the best usability of the tool, the soft scribble feature was proposed. The scribbles can be placed imprecisely (users can drag the brush outside the area they are labelling). The majority of the scribble determines the area which is going to be colored. See section d) on Figure 4.1.
- **Anti-aliasing:** Edges in captured images are anti-aliased. This anti-aliasing has to be present in the colored image as well. See section e) on Figure 4.1.

4.2 Energy Function

The framework that satisfies properties defined in Section 4.1 is defined through energy function. To obtain the result colorization the function is going to be minimized.

The input is a grey-scale image I consisting of a set of pixels P in a 4-connected neighborhood system N and a set of scribbles (brush strokes) S (provided by user) with a particular set of colors C. The result is considered the labelling (color-pixel assignment) c that minimizes energy function (1).

$$E(c) = \sum_{\{p,q\}\in N} V_{p,q}(c_p, c_q) + \sum_{p\in P} D_p(c_p), \tag{1}$$

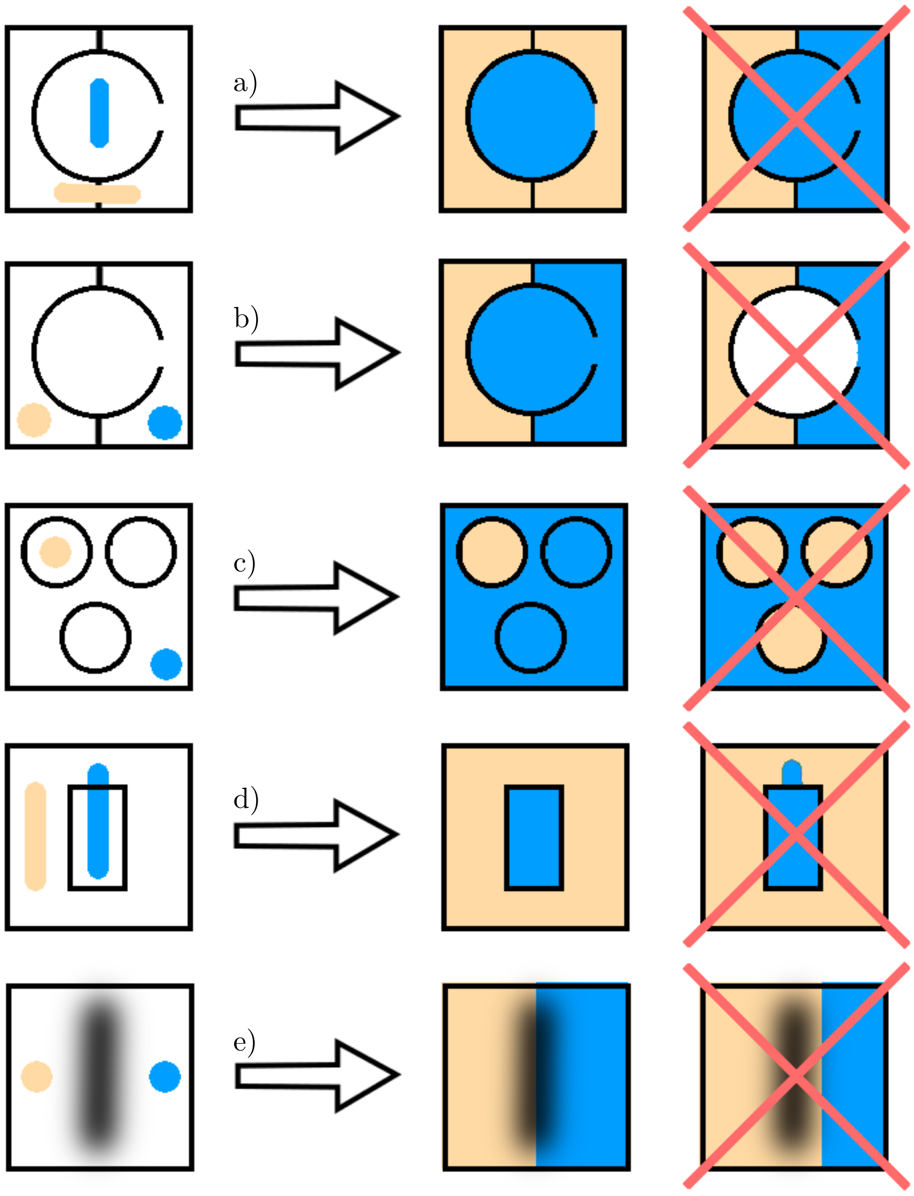


Figure 4.1. Properties of Ideal painting tool: a) Color leakage b) Optimal boundary, c) Connected labelling d) Soft scribbles, e) Anti-aliasing preservation

where p, q are two neighboring pixels, c_p is a color c assigned to pixel p, c_q is a color c assigned to pixel q, $V_{p,q}$ is a smoothness term which defines the energy of intensity differences between pixel p and pixel q according to the assigned color c_p and c_q. $D_p(c_p)$ is a data term defining energy of pixel p according to assignment of the color c_p. For better understanding of meaning of these variables, please refer to Figure 4.2.

4.2.1 Smoothness Term

The first part of formula (1) is called the smoothness term. During colorization, each pixel is multiplied with the scribble's color. Because of this, the best place for color borders is in dark outlines (the black color remains black after multiplication so the color

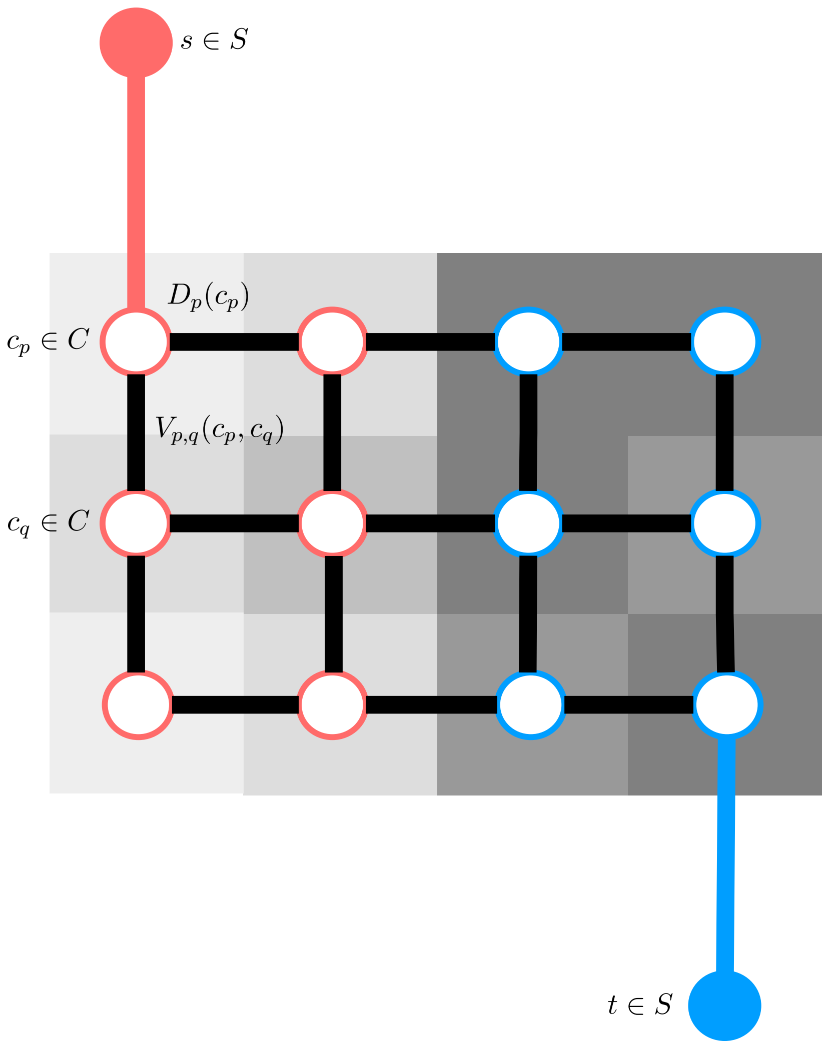


Figure 4.2. Energy function visualization

borders are “hidden” in the outline). Therefore, the smoothness term is formulated in Formula (2).

$$V_{p,q}(c_p, c_q) \propto \begin{cases} min(I_p, I_q) & \text{for } c_p \neq c_q \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

where I_p is intensity of the pixel p, I_q is intensity of the pixel q. For practical usage, there are further requirements on the smoothness term. To obtain regions without holes, nonzero smoothness term in case $c_p \neq c_q$ is required. Otherwise the places where $c_p \neq c_q$ can be disconnected from the network which can create holes. The second problem is that the smoothness term in this form can cause the segmentation to go through white areas. An illustration of this phenomenon can be seen on Figure 4.3. To fix this issue, high-energy values for outlines between the white pixels must be set. For this purpose, the pixel intensities are remapped. As our application colors the greyscale image created with soft pencil, a sufficient choice for intensity mapping function is non-linear mapping (see [14] for more details) that enhances the contrast in formula (3).

$$I'_p = K \cdot {I_p}^2 + 1, \tag{3}$$

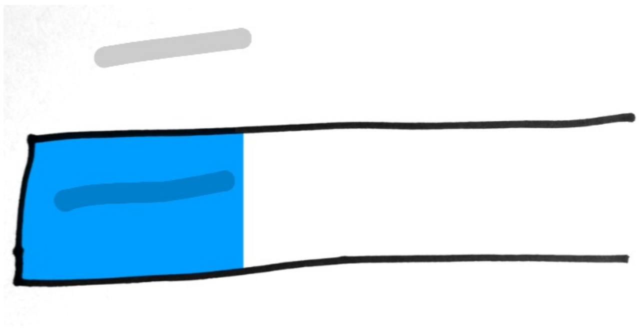


Figure 4.3. Illustration of shortcut through white area

$$K = 2 \cdot (w + h), \tag{4}$$

where w is width of the image I, h is height of the image I and I'_p is the remapped intensity of the pixel p.

4.2.2 Data Term

The second part of the formula (1) is called the data term; it defines hard constraints in the minimization function. In case of LazyBrush not all user-defined scribbles are necessarily hard constraints, which allows us to enable the soft scribbles feature. The labelling is therefore modified by user as follows in formula (5).

$$D_p(c_p) = \lambda \cdot K, \tag{5}$$

where $\lambda \in \langle 0, 1 \rangle$ is a constant and K is a coefficient defined in formula (4). The value λ modifies the meaning of the data term. The value λ is set to $\lambda = 1$ for pixels without any scribble and $\lambda = 0$ for pixels that obtained hard scribble. For soft scribbles the main idea is that a fragment of area $|R|$ with a constant smoothness term should be colored with the color of the scribble with biggest area. Therefore for soft scribbles we set $\lambda > 1 - \partial S/|S|$, where ∂S is the perimeter of the scribble S and $|S|$ is the area of the scribble. For detailed inference of this formula, please refer to [14]. As for most scribbles the $1 - \partial S/|S| < 0.95$ holds, in practice we can set $\lambda = 0.95$ for all scribbles instead of measuring the ∂S and $|S|$ values for each scribble separately.

4.2.3 Minimization

Because the energy function (1) is dependent on the pixel intensities only and not on the color labels, it satisfies the Potts model [15]. According to [16] minimization of a function that satisfies Potts model can be transformed to multiway cut problem on a certain undirected graph $G = \{V, E\}$, where $V = \{P, C\}$ is a set of vertices of particular pixels P, and color terminals C and $E = \{E_p, E_c\}$ is a set of edges. Each pixel $p \in P$ is connected with its 4 neighbors that orthogonally surrounds pixel p via edges E_p having weight $w_{p,q} = V_{p,q}$ for case $c_p \neq c_q$. Edges E_c connect color terminals with pixels labelled by scribble having weight $w_{p,c} = K - D_p(c)$ — where for hard scribbles $w_{p,c} = K - 0 \cdot K = K$ holds and for soft scribbles $w_{p,c} = K - \lambda \cdot K = K \cdot (1 - \lambda)$ holds. The whole situation can be seen before segmentation on Figure 4.4 and after segmentation on Figure 4.5. The intensities in this case have three levels (white, grey, black) and from these intensities the capacities of edges are deducted. The more thick the edge, the more capacity it has.

According to [17], the multiway cut with 3 or more terminals is the NP-hard problem. To solve this problem an approximate α-expansion algorithm could be used. The

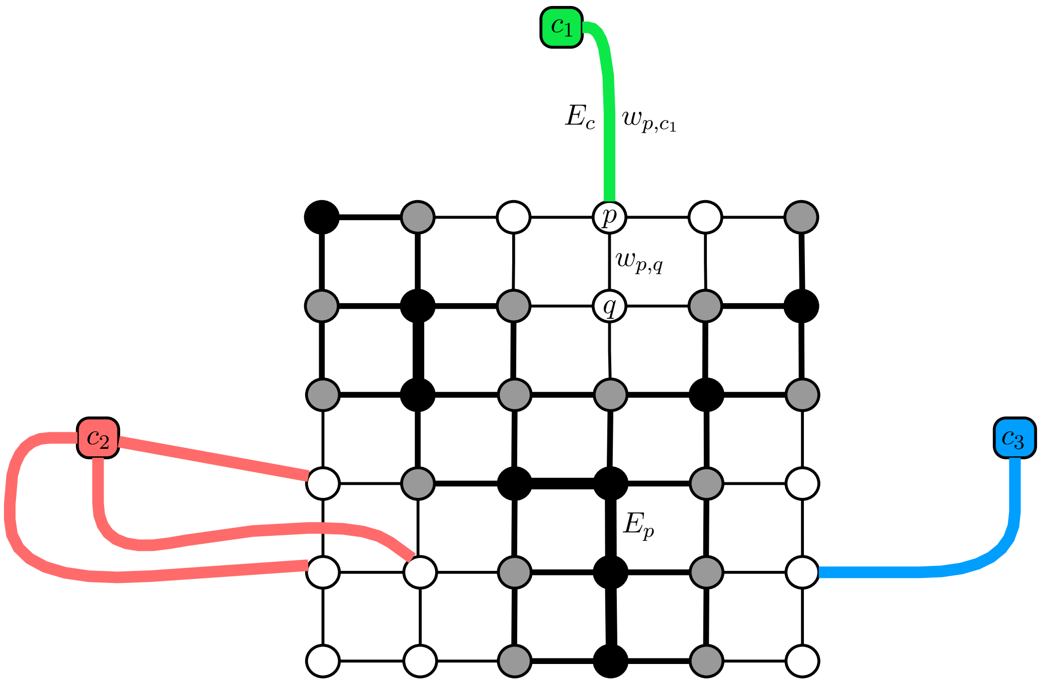


Figure 4.4. Graph of multiway cut

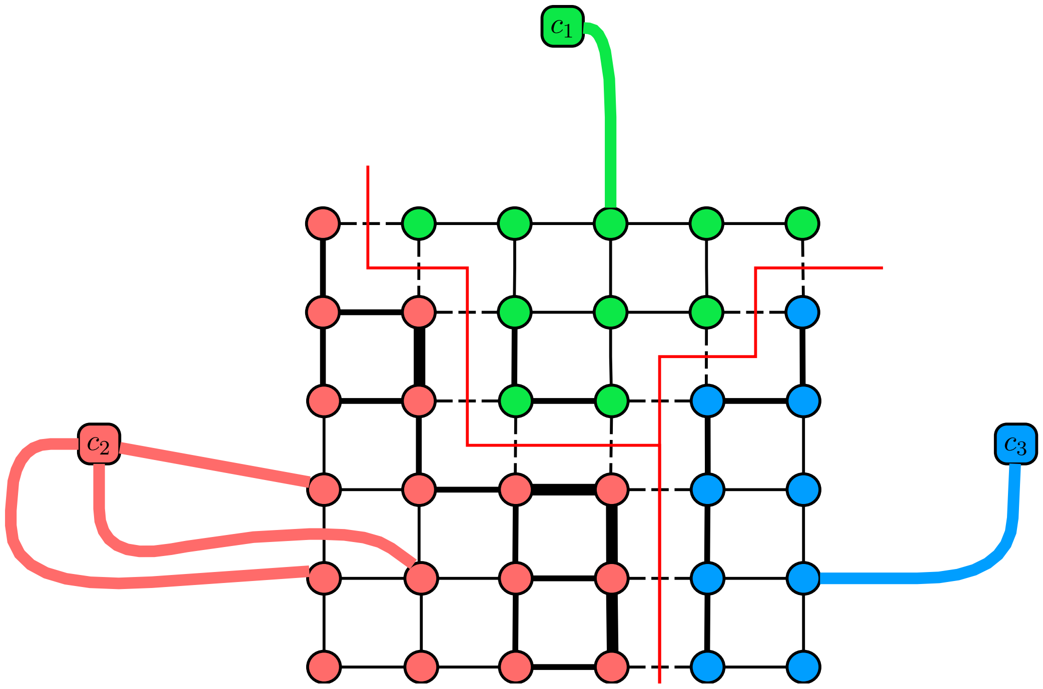


Figure 4.5. Solved and colored graph of multiway cut

problem is that the α-expansion algorithm is still slow for the purpose of the interactive image colorization. To address this issue the LazyBrush algorithm comes with greedy multiway cut algorithm defined below. This algorithm does not guarantee the optimality, but provides comparable results to the mentioned α-expansion and is several times faster.

- Input: Image I, set of user-defined scribbles S.
- Output: Colored Image I'.

1. Initialize the set of background color labels C. Each label $c_i \in C$ corresponds to one particular scribble $s_i \in S$.
2. Create the set M of all uncolored pixels in image I.
3. Choose one arbitrary label $c_f \in C$ and remove it from set C.
4. Create graph $G = \{V, E\}$ from set M. Connect pixels with label c_f to terminal S and pixels with all labels $c_i \in C$ to terminal T.
5. Solve the maxflow/mincut problem [18] above graph G from terminal S to terminal T.
6. Color pixels corresponding with terminal S with the color of label c_f. Remove these pixels from the set M.
7. If the set C is not empty, go to (3), otherwise set output image $I' = I$, end.

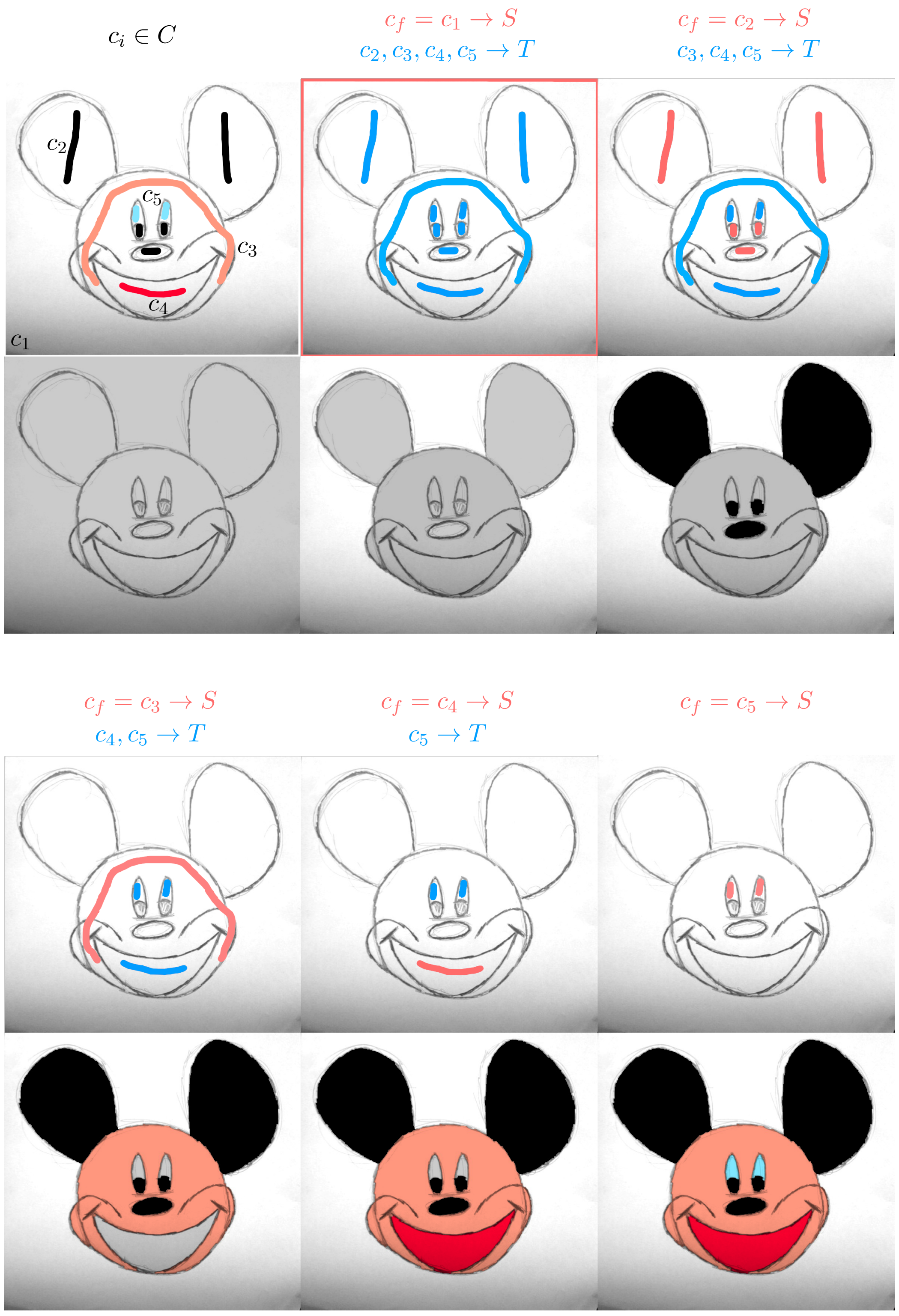


Figure 4.6. Work of LazyBrush algorithm. In every iteration one label c_i is set as the foreground c_f and is removed from the set of labels C. On images in the second and fourth line the colorization after each iteration is shown. Grey color means uncolored pixels that remain in the set M for the next iteration.

Chapter 5
Requirements Analysis

The requirements analysis consists of two main parts—functional and nonfunctional requirements. Functional requirements are defined through use cases and use case scenarios which define detailed user interaction with the application. The use case scenarios are based on the style that was presented by Cockburn [19]. According to this requirements analysis, the actual implementation of the LazyBrush application for iOS is created.

5.1 Functional Requirements

In the SketchColor application two entities come into the play—the user and the system. There is only one user role—the artist. No other roles are considered since neither system administrators nor other staff for this project are needed.

Interaction between the user and the system is defined on use case diagram on Figure 5.1. This diagram represents the functional requirements that the application shall fulfill. Each functional requirement is described in detail in use case scenarios.

Each scenario describes certain user activity. It consists of main scenario and alternative scenarios. Field "trigger" defines which scenario is called after the current one finished. Pre-conditions are conditions that must be fulfilled before the scenario; post-conditions are conditions that must be fulfilled after the scenario in case the scenario was successful. Field "level" defines the granularity of the use case. According to Cockburn [19], there are three levels. Those are: summary, user goal and sub-function. The most interesting for us is user goal, as it describes the tasks of the user in the system.

Use Case: Obtain Drawing.
ID: 1
Description: User chooses how to obtain the picture to be colored. There are two possible methods — Take Photo (ID: 1.1) or Choose Photo From Library (ID: 1.2).
Level: User Goal
Pre-Conditions: User opened the application.
Post-Conditions: An image is obtained and prepared for colorization.
Trigger: None.
Main Scenario:

1. User selects one option—either Take Photo or Choose Photo From Library.
2. System triggers Take Photo (ID: 1.1) or choose image from image library (ID: 1.2) according to the user's selection.
3. The system obtains an image.
4. The system filters the image to enhance the edges.
5. The system opens the obtained image to color it.

Special Requirements:
In the step 4 of the main scenario, the system filters the image. This is needed because for the colorization process is much better if the image is more black and white than greyscale—e.g. it is needed to have dark (ideally black) edges and light (ideally white) background (as the background is mostly the white paper).

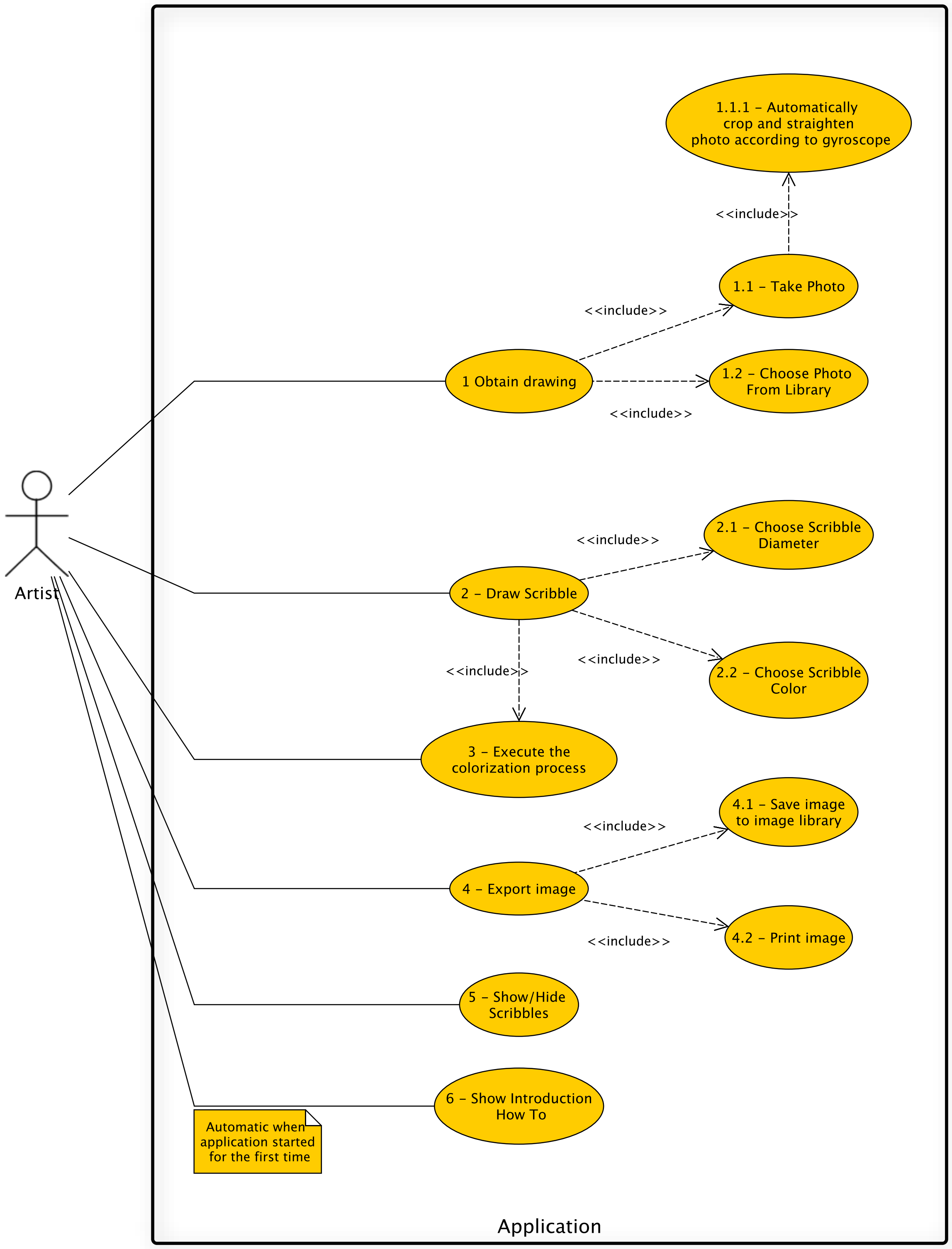

1.1.1 – Automatically crop and straighten photo according to gyroscope
<<include>>
1.1 – Take Photo
<<include>>
1 Obtain drawing
1.2 – Choose Photo From Library
<<include>>
Artist
<<include>>
2.1 – Choose Scribble Diameter
2 – Draw Scribble
<<include>>
<<include>>
2.2 – Choose Scribble Color
3 – Execute the colorization process
4.1 – Save image to image library
<<include>>
4 – Export image
4.2 – Print image
<<include>>
5 – Show/Hide Scribbles
6 – Show Introduction How To
Automatic when application started for the first time
Application

Figure 5.1. Use case diagram

Use Case: Take Photo.
ID: 1.1
Description: The live camera preview is presented and the user takes the desired photo of the drawing to be colored.
Level: Sub-Function
Pre-Conditions: User selected the option to take a photo in a scenario (ID: 1).
Post-Conditions: An image is captured through the camera. The image is cropped to contain the white paper with the drawing only, the image disortion is compensated with data from gyroscope sensor.
Trigger: Automatically crop and straighten photo (ID: 1.1.1) in case of main scenario. In case of Alternative Scenario 1 trigger (ID: 1).
Main Scenario:

1. The system opens dialog for image capture. Choose Photo From Library.
2. The user moves the camera to see desired image.
3. The user instructs the system to capture the photo.
4. The system captures the photo.

Alternative Scenarios:

Alternative Scenario 1

1. The system opens dialog for image capture.
2. The user cancels the image capture, no photo is captured.
3. The system goes to scenario (ID: 1).

Use Case: Automatically crop and straighten photo.
ID: 1.1.1
Description: The captured image is cropped to contain only the area of the paper with the drawing and the image disortion is compensated with data from gyroscope sensor.
Level: Sub-Function
Pre-Conditions: The photo in scenario (ID: 1.1) was captured.
Post-Conditions: The photo was cropped and the disortion was compensated from data from the gyroscope sensor — the images should look like it was captured with camera perpendicular to the plane on which the paper lies even if the image was captured from another than perpendicular angle.
Trigger: None.
Main Scenario:

1. The system obtains the image to be modified.
2. The system compensates the photo with data from gyroscope to look like it was captured with camera perpendicular to the plane on which the captured image lies.
3. The system crops the image to contain only the paper with the drawing.

Use Case: Choose Photo from Library
ID: 1.2
Description: The user selects the image stored in the image library to be colored.
Level: Sub-Function
Pre-Conditions: The user selected the option to select image from device image library.
Post-Conditions: An image is chosen.

Trigger: In case of Alternative Scenario 1 trigger scenario (ID: 1).
Main Scenario:
1. The system opens dialog for image selection from the image library.
2. The user selects the desired image.

Alternative Scenarios:

Alternative Scenario 1

1. The system opens dialog for image selection from the image library.
2. The user cancels the selection, no image is selected.
3. The system goes to scenario (ID: 1).

Use Case: Draw Scribble
ID: 2
Description: The user touches the screen and drags his finger over it. The system draws the scribble above the image.
Level: User Goal
Pre-Conditions: An image is opened and prepared for coloring. System knows the color and thickness of the scribble being created.
Post-Conditions: The scribble is drawn.
Trigger: Execute the colorization process (ID: 3)
Main Scenario:
1. The user drags his finger above the area where the scribble shall be placed.
2. As the user drags his finger, the system records points where the user touched during the drag. The system automatically draws the spline that goes through the recorded points as fast as possible—the user shall not observe any lag. The spline has predefined color and thickness.
3. When the scribble is finished, the user will release his finger.
4. The system ends the scribble.

Use Case: Choose Scribble Diameter (Color)
ID: 2.1 (diameter), 2.2 (color)
Description: The user chooses the diameter (color) for the scribbles that are going to be created.
Level: Sub-Function
Pre-Conditions: The image is open and prepared for coloring.
Post-Conditions: The diameter (color) of the scribble was chosen.
Trigger: None.
Main Scenario:
1. The user instructs the system to change the diameter (color).
2. The system presents the view for changing diameter (color).
3. The user chooses the diameter (color).
4. The system sets the diameter (color) for future scribbles.

Use Case: Execute the Colorization Process
ID: 3
Description: The system will calculate the colorization of the image according to the created scribbles.

Level: User Goal
Pre-Conditions: One or more scribbles were created in scenario (ID: 2).
Post-Conditions: The image is colorized and presented on screen.
Trigger: None.
Main Scenario:

1. The system is instructed by user or other scenario to start the colorization process.
2. The system colorizes the image.
3. The system presents the colored image.

Use Case: Export Image
ID: 4
Description: Exports the image. Image can be saved to device image library or printed on AirPrint printer.
Level: User Goal
Pre-Conditions: The image is open.
Post-Conditions: The images are saved to the device image library or printed on the printer.
Trigger: None.
Main Scenario:

1. The system is instructed to save the current image into device image library or print the image on the printer.
2. The system saves/prints the image.
3. The user is notified that the images were saved or sent to printer.

Alternative Scenarios:

Alternative scenario 1

1. The system is instructed to print the image.
2. The system did not find any printer.
3. The user is notified that no printer was found.

Use Case: Show/Hide Scribbles
ID: 5
Description: For better experience, user can turn show or hide the scribbles. The scribbles are hidden on default. This setting does not affect the scribbles that are being drawn, these scribbles are always visible.
Level: Sub-Function
Pre-Conditions: The image is open.
Post-Conditions: Drawn scribbles were shown/hidden.
Trigger: None.
Main Scenario:

1. The user instructs the system to show or hide drawn scribbles.
2. The system shows or hides the scribbles.

Use Case: Show Introduction How To
ID: 6
Description: When the application is run for the first time, the introduction how to is shown to teach the user what is the application about and what he can do.

Level: User Goal
Pre-Conditions: The application is run for the first time, or the user selected an option to see the introduction how to again.
Post-Conditions: None.
Trigger: Obtain drawing (ID: 1)
Main Scenario:

1. The user opens the application for the first time.
2. The system presents the introduction—sample prepared image is opened. The system instructs the user how to draw a scribble.
3. The user draws a scribble as instructed by the system.
4. The system colorizes the image.
5. The system instructs the user how to select color and scribble thickness. The system instructs the user to select new color and to make another scribble to another area.
6. The user selects new color and makes new scribble.
7. The system colorizes the image.
8. The system instructs the user how to show or hide scribbles. The system also shows how to export the image.
9. The system welcomes the user to the application and ends the introduction how to.

5.2 Non-functional Requirements

Non-functional requirements are here defined with emphasis of usability. Usage of mobile applications differs rapidly from desktop ones. The main difference is that the user wants to wake his mobile or tablet, do some quick action and then asleep the device again. On the other side, the desktop is typically used when the user has more time so the usage is slower.

- **Memory budget**: To solve the maxflow/mincut problem the considerable memory space is needed. According to unofficial experiments, around 45% of memory of the device is usable until the system shuts the application down, so it is crucial to lower the memory consumption of other components of the application as much as possible.
- **Responsiveness**: From usability point of view it is very important that the user interface stays responsible without lags especially during scribble drawing. During drawing of the scribbles, the user has to concentrate and must not be disturbed by non-smooth screen updates.
- **Usability**: Since the application is intended mainly for children, it should be easy to use for them. However, we do not want to discourage adult users from using the application—this must be kept in mind during the UI design phase.

5.3 User Research and User Requirements

To design the User Interface of the application properly, user research with children has been made. Because children behavior differs greatly according to their age, the target group was sampled to three main age groups and for each user group important questions regarding children activities connected to drawing and coloring pictures were answered.

5.3.1 Target Group of Users

Target group is very important in this stage. Our target group is children that like to play with colors and mobile devices, children that like to draw and colorize pictures. These tasks can be done at home, at school during art lessons or with friend on an afternoon party. Children can draw and want to colorize pictures for various reasons — as an entertainment, because they have to (school homework) or because they want to create a gift for mom or relatives. All children doing theese or similar tasks are our target user group.

These users are divided into five smaller groups according to their age: in the first one there are children below the age of 7 years, in the second there are children aged 7 - 9, in the third 10 - 12 years old ones, in the fourth 13 - 15 years old and in the fifth one children above 15 years. The secondary target group of users is other people that like drawing, colorizing—or art at all. We design the application with emphasis to children between 7 and 15 years. Younger children have very specific requirements (small hands, inability to read, etc.) and thus special application for this category would be needed. The same situation is with the group of users older than 15 years — probably the feature set of this application will be too small to be ideal for them.

5.3.2 User Research

The whole procedure of User Research is inspired by Chapter 2 and 3 from [20]. During user research various users were asked to provide answers for the questions regarding drawing, colorizing, drawing on tablets and smartphones, etc. The main questions we wanted to answer are following:

- Why children draw? What are the objects or actions they like the most to draw?
- Do children like more to draw with a pencil, crayons, chalks or a pen? Do children like more to draw greyscale or color pictures?
- How easy is for children to use state of the art applications for colorizing like ColoringBook from the Chapter 1? What are the most common problems they have using these applications?
- Do children use tablets or smartphones — generally touch devices? What are the most interesting applications for them?

These questions were answered during interviews with children. These questions were not asked directly—the users told us about how they are actually drawing, why they use this tool, etc. From each category of interest there was at least one respondent interviewed. Answers from these questions were generalized into table 5.1. Some findings cannot be generalized into the table. These findings are generally valid for all subgroups of our users:

- They like more landscape paper orientation.
- They do not like to read.
- They behave quickly — they make quick and irresponsible decisions.

All interviewed children like to use touchscreen device. They use it mainly for games; the older ones use it also for writing emails, sms or writing down notes in school. None of interviewed children has broad experience with applications for colorization. This imply that our approach of a bitmap images colorization is really new for them so it may be problematic for users to understand for what the application is good for, which features it has and where are the limitations. To identify the finger motoric skills an existing iOS application for colorization was used. This application

User characteristic	age: 7 — 9	age: 10 — 12	age: 13 — 15
Sex	male/female	male/female	male/female
Physical limitations	worse finger motoric skills, probably better for girls, reading word or two is not a problem	finger motor skills can be precise as well as imprecise, probably better for girls	finger motoric skills are better but some users may still have problems using touch device precisely
IT experience	rare usage of touch devices, probably does not have his/her own, games only	more frequent usage of touchscreen devices, sms, games	heavy usage of touchscreen devices, writing emails, notes in school, sms, taking photos and retouching them, games
Motivation	entertainment, interested in drawing, school, like using touchscreen devices	entertainment, interested in drawing, school, like even more using touchscreen devices	entertainment, interested in drawing, school, using touchscreen devices is everyday life, it is “in”
Attitude	like more to draw color pictures, colorization is entertainment	greyscaled pictures seems to be more entertaining — for both drawing and observing	does not like colorization, drawing is entertainment both color and greyscaled
Prefferable tool	crayons, watercolors, tempera, color fix	soft pencil, chalk	soft pencil, pen, tempera

Table 5.1. User research summary

does not automatically colorize particular areas of the picture (i. e. the bucket tool), so children had to colorize the object manually. Some children were able to do it almost without a mistake, other were not able to place the brush strokes precisely. This research revealed interesting fact that even 12 years old child can have the same problems with the precision of placing brush strokes as the 7 years old child.

The user characteristics from table 5.1 are transformed into user requirements on system being designed in table 5.2. The three age groups were merged into one, because only one application for whole group is going to be designed and implemented. Thus all the findings must be merged to satisfy the majority of users.

User characteristic	Children aged from 7 and above
Can have bad motoric skills of fingers or hands.	The UI elements must be big enough to hit them easily. Also there should be a lot of free space between UI elements for better orientation.
May have problems with reading.	All textual information should be accompanied by image or icon to understand the meaning easilly.
Can have little experience with platform/application of this kind	All features shall be self-explaining or explained by some other form like tutorial.
Make quick irresponsible decisions	All changes should be revertible.

Table 5.2. Conclusion of user requirements from user research

From the user research it is obvious that the UI design has to be created with the emphasis on simplicity and easy intuitive navigation. Important is the fact that not all children are able to read so all possible textual information has to be supplied with descriptive icon so there is obvious what the label is saying. Usage of bigger buttons and sufficient amount of free space cannot be underestimated as well. Free space helps the user to identify and hit the correct UI element in shorter time. As the proposed colorization technique is entirely new to touchscreen devices, the user shall be informed at the first launch of the application about the application features, for example in a form of a short interactive tutorial.

Chapter 6
UI Design

The user interface design was created according to the Chapter 5 and is inspired by the user research (see Section 5.3). The creation of the UI design of the final product was though iterative process. Thanks to the iOS platform, no dummy prototypes were created. Implementing user interface on iOS is so quick that it is better to prototype it on real application, reuse it and improve it in next iterations. The application was tested informally with users through the whole development. In final stage the application was tested more formally with several children (see Chapter 8.2). These findings were then used for improving the UI design being presented in this chapter. The UI design is divided into two main parts. The first part talks about the tutorial and the second part talks about the actual image colorization.

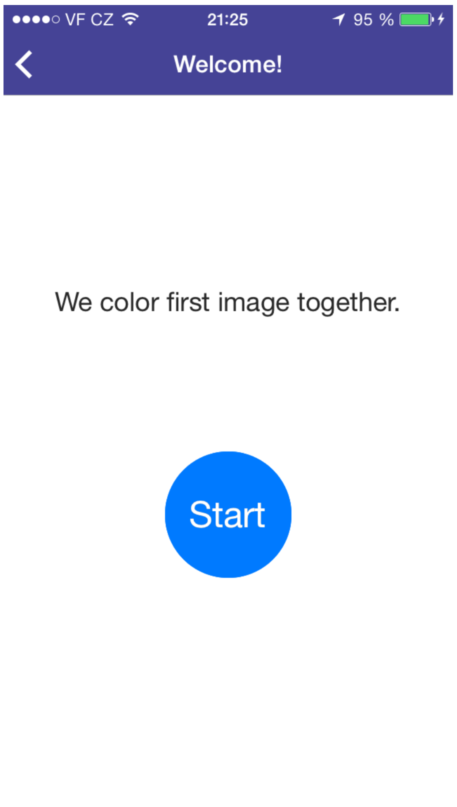
6.1 Tutorial

After the application is launched for the first time, the tutorial is started automatically. The welcome page with start button can be seen on Figure 6.1. All buttons in the application that start some important activity are big circular buttons. The colors were chosen to have big contrast against the white text. In the tutoriali, the user shall learn mainly four important things.

- Learn that the pinch zoom is available.
- Learn that scrolling is possible with two fingers.
- Learn that the application colors automatically the areas where a scribble was placed.
- Learn that any brush stroke can be removed with the shake gesture (undo feature).

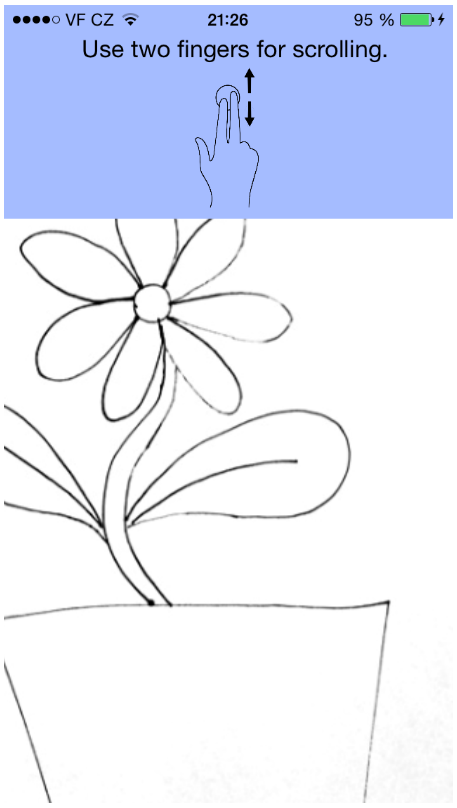
These are the most important things in the application and they must be mentioned in the tutorial because there are no visible UI elements to let the user know that these features exist. In contrast the color can be changed by tapping on the button with color bucket which is pretty straightforward and hence does not need to be presented in the tutorial. The less actions are in the tutorial, the better—the features learned have better chance to be remembered if there is only few information.

The instructions how to zoom and how to scroll are very similar to each other; the scrolling part of the tutorial can be seen on Figure 6.2. Instructions for drawing scribbles are presented via animation. The user is asked to draw the scribble into the flowerpot and the leaf according to the brush strokes that are drawn interactively on these places. After the user places each brush stroke, the application colors the image so the user sees the result immediately. The user can learn how to use the undo/redo feature on the Figure 6.3. Undo/redo is implemented as a standard iOS gesture for undo and redo—shake gesture. After the user completes all the tasks given by the tutorial, the tutorial ends and the main application dashboard (see Figure 6.4) is brought on the screen.



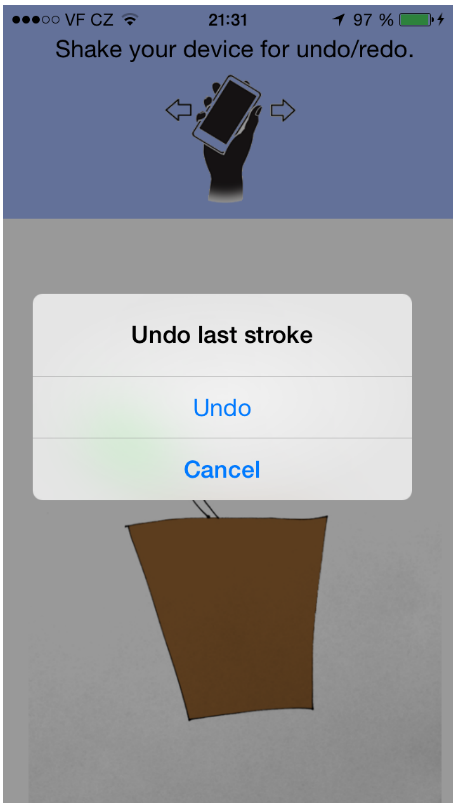
We color first image together.

Figure 6.1. Tutorial welcome page



Use two fingers for scrolling.

Figure 6.2. Tutorial—scrolling



Shake your device for undo/redo.

Figure 6.3. Tutorial—undo

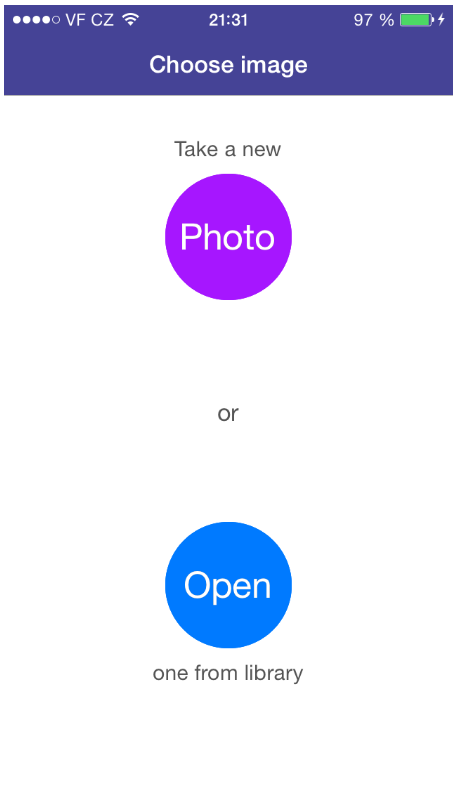


Figure 6.4. Main application dashboard

6.2 Image Colorization

In case the tutorial has been completed earlier, the main application dashboard (on Figure 6.4) is opened on application launch. There are two main options to load a photo into the application. The first option is to take a photo directly from the application. The advantage of taking photo directly from application is that the application crops the image automatically according to found edges of the paper and removes the perspective disortion. Thanks to this it is possible to take nice photos without drop shadows even when the light source is directly above the paper—we can take the photo from side and thus do not let the drop shadows to appear on the paper. The second option is to open an existing picture from the image library. Such a picture could be captured earlier, or downloaded from the internet. The dashboard thus contains two buttons, one for each option. Again, these buttons are big circular to encourage the user to tap on them. The buttons can be quickly recognized against each other thanks to the different colors that comply the color scheme of this application—tones of blue and violet.

The camera view was created to look similar to the standard iOS photo application—mainly the shutter button which is very important in camera application. Therefore, users know instantly what is the button for. The camera view has only two options—to take a photo with shutter button or to cancel it with cancel button. No other UI elements are needed; all remaining tasks like focus the camera view provides automatically.

When an image is opened, it can be colored. The color can be changed by tapping on the color bucket button which is located on the bottom toolbar on the right side. The color picker is a standard picker on the iOS. From this color picker, on Figure 6.6, it can be seen that it has one major disadvantage. Only the current color is fully visible, all other color are behind a fog. Therefore the user recognizes the right shade of the color only after he scrolls on it, which is inconvenient. There are plans to create custom color picker but this has not been done yet due to the lack of time. The color bucket changes its color according to currently selected color to visualize it. On the Figure 6.7, a change of brush size can be seen. Again, a standard iOS component named Slider was used to change the brush size. The preview of the brush width is visible below the slider.

Both color picker and brush size picker are placed on the half screen popover (Figures 6.6, 6.7). For iPhone platform, in contrast to iPad platform, the popover is not in standard library so it had to be implemented. Without such popover normal full screen view would be pushed on the current view with components for color picking or a brush size. The popover has main advantage against standard approach with full screen view—the user sees the image being colored and its current colors. Therefore, the color picking is much easier because the user does not lose the context for which he chooses the color. When the user picks the brush size, again the part of the picture, for which the new brush size is being picked, should be visible so the user does not need to recall the needed brush width. Another advantage of the half screen popover is that the animation which brings the popover is visually nice so the user enjoys it more.

On Figure 6.8, there are scribbles shown on top of the partly colored image. This functionality is needed when more complex images are colored so the user needs to know which parts obtained a scribble.

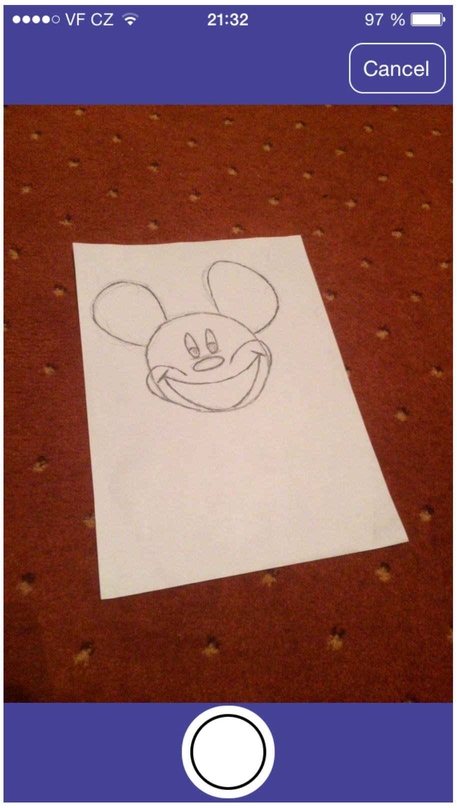


Figure 6.5. Camera UI

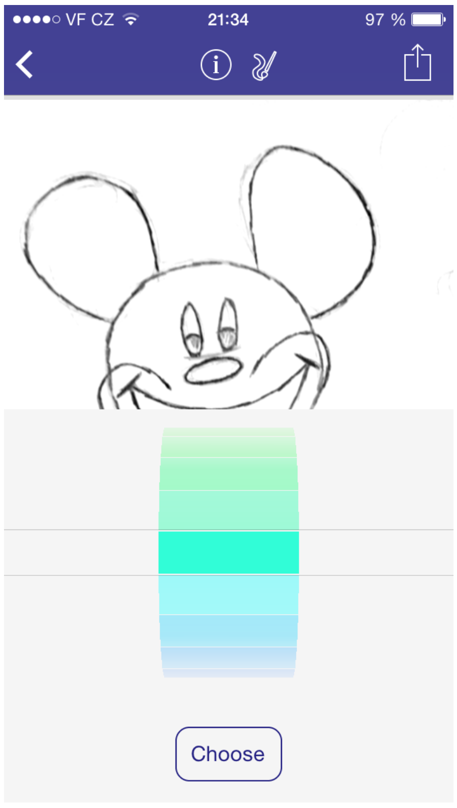


Figure 6.6. Color picker UI

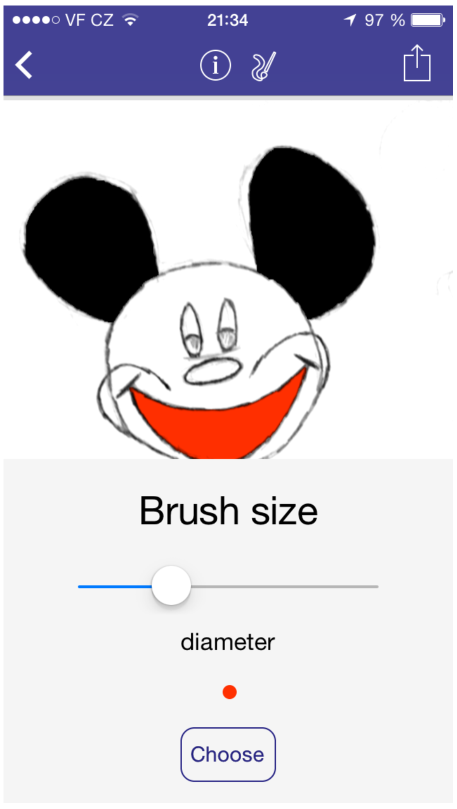


Figure 6.7. Brush size chooser

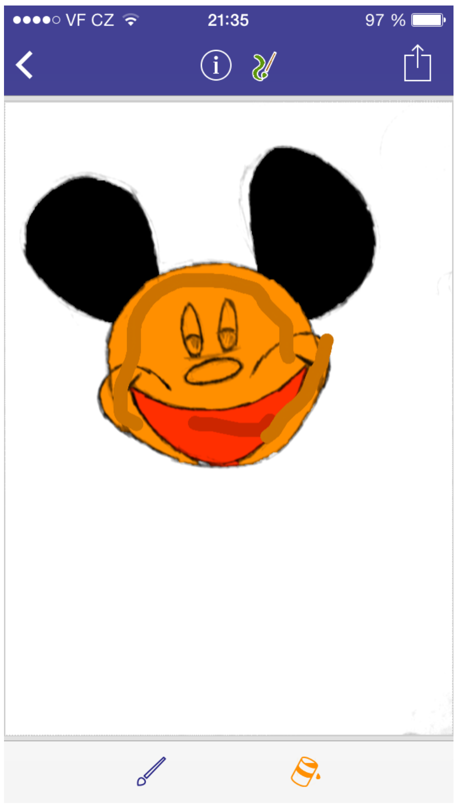


Figure 6.8. Scribbles shown

Chapter 7
Implementation

The entire application is implemented for iOS and is runnable on iPhone and iPad. The application is restricted to iOS 7.0 or higher as it uses new components to present the content. Applications for iOS can be implemented in Objective-C or Objective-C++. Other languages like plain C++ are possible to use as well with following restriction—the implementation of UI must be implemented in Objective-C(++). There are some RAD tools that allow to implement the application in pure C++ (C++Builder) or Object Pascal (Delphi), etc. These tools allow the programmer to write one application for more mobile platforms at once. For this application, the Objective-C++ was chosen because of following reasons. The code for this application mostly implements the user interface. There is of course the implementation of LazyBrush algorithm that does not deal with UI but this is not the majority of the code. The reason for using Objective-C++ prior to Objective-C is that the application uses the C++ GridCut [18] library for the segmentation problem and thus it must be compiled as Objective-C++.

7.1 Application Architecture

The architecture of the application is designed according to the requirements analysis in Chapter 5. The overview of the workflow can be seen on Figure 7.1. The user takes a greyscale photo which is automatically cropped according to the boundaries of the paper. The user can also open a photo from image library—in this case, the system does not crop the image. Then the source image is filtered to enhance edges and suppress the shade gradients and dropped shadows. Then the user draws the scribbles on top of the image. Each time a scribble is added, the system colorizes the image so the steps 5 and 6 can be repeated several times. The colorized image can be printed on an AirPrint friendly printer or saved to the image library of the device.

The whole application consists of several main components visible on Figure 7.2. The entry point for the application is the ImageSelector component which lets the user choose between two options—take a photo or open an image from the gallery. These two actions are provided by the ImageSource component. Capturing a photo is provided by the Camera component which is built on top of the iOS AVFoundation framework. Opening a photo from the image library is done by the ImagePicker component. After the image is provided by the ImageSource component, the ImageSelector makes further preprocessing of the image. The preprocessed picture is taken by ColorizationController which manages the colorization process. Scribbles are created through ScribbleManager and rendered by ScribbleRenderer. The scribbles are used by Solver to colorize the image which is provided by ColorizationController at any time.

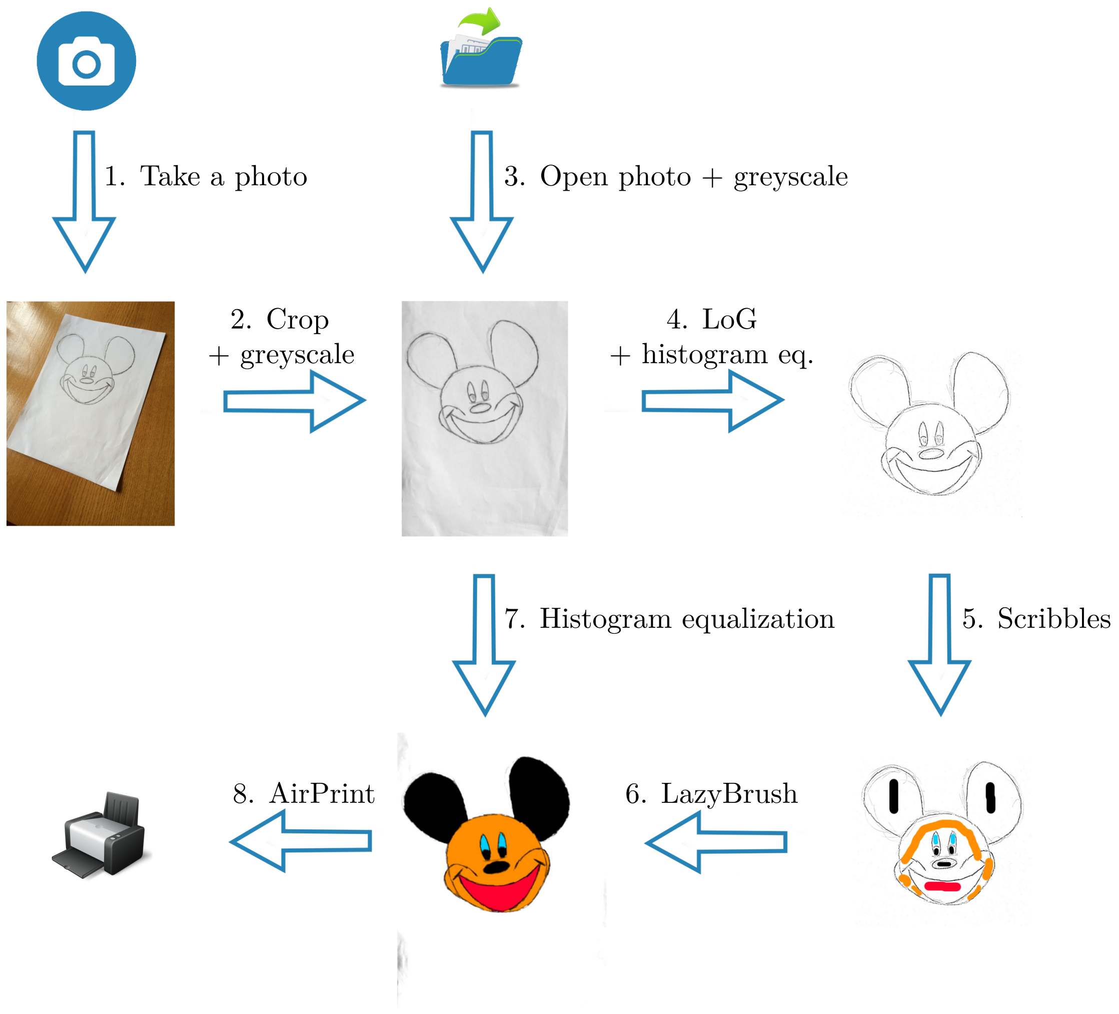


Figure 7.1. Application workflow overview

7.1.1 ImageSelector Component

This is the entry point component for the application. It provides the image to be colored and it requires an image either from the camera or the system image library. Before the image is provided to other components it is preprocessed. In case the image was taken by the camera, the paper edges are registered and the image is cropped (see Section 7.3) with the CropImage component. After cropping is done or in case the image was opened from the system image library, the image is filtered to enhance outlines and to clear the background by the ImageFilter component as described in Section 7.2. The CropImage component is implemented by the CropImage class and ImageFilter is implemented in class Filter. The actual implementation of the ImageSelector component is provided by several classes because it is closely coupled with the views that are presented on screen. The intention is to start image picking or the capturing process that shows the actual view for the selected action. Once the action finishes, the result is passed onto another view which shows the image and allows the user to add other inputs like scribbles, etc. On iOS, each main view is managed by UIViewController or its subclass. The initial view controller presenting the initial view is RootViewController. When the user makes decision whether to choose the image from the library or whether to take a new one, the controller calls the custom

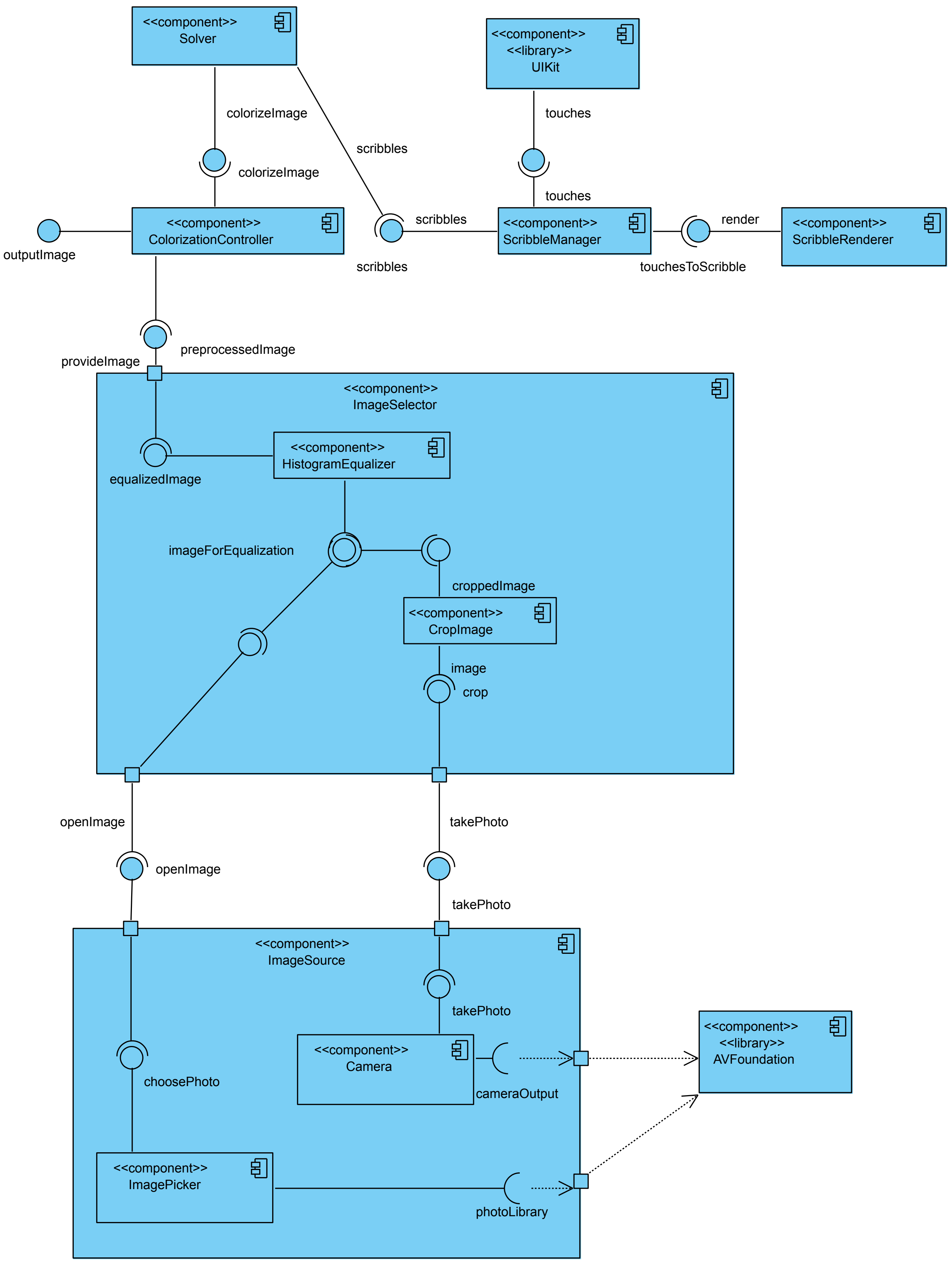


Figure 7.2. Application architecture component diagram

segue. The segue is a mechanism in iOS to trigger an action while a new view controller is presented on top of the previous one. In this case, the OpenPhotoSegue resp. TakePhotoSegue calls UIImagePickerController or TakePhotoViewController which provide an image. The result image is then preprocessed by the CropImage class or the Filter class. The final image is then passed to LazyBrushViewController which is the actual implementation of ColorizationController. In other words, the ImageSelector component is implemented by the RootViewController, the TakePhotoSegue and the OpenPhotoSegue classes.

7.1.2 ImageSource Component

The ImageSource component is created by two other components. The ImagePicker component is already implemented in the UIKit framework as UIImagePickerController. This component is capable of taking photographs as well but, because the application needs custom gyroscope data from the time a new photo is captured, this Camera component had to be implemented from scratch. The implementation of Camera component is TakePhotoViewController. It is implemented on top of the AVFoundation framework that allows to access raw camera data in real time.

7.1.3 ColorizationController Component

ColorizationController is in fact implemented in the LazyBrushViewController class. This controller presents all the options that the user can choose when drawing new scribbles on the screen. ColorizationController needs an image that is prepared for coloring and Solver that provides the actual colorization of the image. This component provides the colored image any time for a further manipulation (saving or printing).

7.1.4 Solver Component

The Solver component is implemented with the SolverManager class. It manages the execution of the LazyBrush algorithm. It requires an image and scribbles as an input and provides the colored image. For more details about the SolverManager implementation, refer to Section 7.4.

7.1.5 ScribbleManager Component

The implementation of this component is the SelectionImageView class. It responds to all touches events and creates new scribbles accordingly.

7.1.6 ScribbleRenderer Component

The ScribbleRenderer component is implemented by two classes — the Scribble class and the GLQuadraticPath class. The rendering is important to be able to visualize the scribbles and to calculate which pixels are labeled by which scribble. For details about the implementation of scribbles, see the 7.5.

7.2 Image Filtration

According to [14], the LazyBrush algorithm is more accurate and faster on images with dark outlines and a white background (i.e. the intensity difference between an outline and an area that will be colored is maximal). When a photo of a hand-drawn image is taken, the result is not ideal for coloring with LazyBrush due to a small contrast of outlines and the background. The captured image has a lot of greyscale gradients even if drawing on a white paper was photographed. Moreover, such image can contain unintended shadows dropped by user's hand holding the device. This is dependent on illumination conditions, the device position, etc.

To obtain better image, the source photo is filtered to enhance the edges in the image and to get rid of the unintended shades of the background or drop shadows. Since the edge detection is needed by many applications, this field of the image manipulation and the computer vision was studied largely. At first, simple filtering methods like the Roberts cross operator [21], the Sobel operator [22] and the Prewitts operator [23] were invented to enhance the edges in the image. Marr and Hildreth [24] presented a Laplacian of Gaussian (LoG) operator for the edge detection at zero-crossings. While these filters are effective in detecting edges, the output can be disturbed by a noise. To address this issue, the Canny edge detection filter and algorithm [25] with a non-maximum suppression and hysteresis tresholding for detecting step edges was presented. On top of these ideas, more powerful techniques for the edge extraction were developed. To obtain a reasonable set of edges from pictures captured in various environments with various light conditions, the automatic scale of the filter operators has to be estimated. A method for such estimation is presented in [26].

The method described in [26] is usable mainly if the domain of the objects on the images is unknown. Since the domain of the interrest for our application are hand-drawn images with a soft pencil or a pen, our task is simpler, therefore there is no need to implement such a robust technique. In [14], the LoG filter with linear rescale of pixel intensities was used. Inspired by this technique, similar filtration was implemented. The main advantage of the LoG filtration is that the edge center is located in local maxima of the filtered image. Therefore, we obtain a single outline per edge unlike of other techniques that locate edge borders.

7.2.1 LoG Filter

The LoG filter uses the second derivative of Gaussian kernel (see Table 7.2). Since the filtration with this kernel is sensitive to noise in the signal, the noise is at first filtered out by the small Gaussian kernel visible in Table 7.1. The image is convoluted with the Gaussian filter at first, and then with the second derivative of the Gaussian kernel. Thanks to the associativity of the convolution, we can convolute the second derivative of the Gaussian kernel with the Gaussian kernel and then use this new kernel to filter the image. After the convolution, the image must be negated to obtain black edges and white background. Since naive implementations of convolution are slow, the iOS Accelerate low-level C framework was used in the implementation. More specifically, the Accelerate framework has function (*vDSP_f3x3()*) for convolution of the 1D or 2D discrete signals with the arbitrary 3×3 kernel.

Another possibility to implement the convolution quickly is to use some third party library for the fast-fourier transformation. The image and the kernel are transformed with the fourier transformation, then according to the convolution theorem, the image is multiplied with the kernel and the result in transformed back with the reverse fourier transformation. The result is the convolution of the image. Most probably, this is the way the Accelerate framework really does the convolution itself, because it allows the user to use the arbitrary kernel.

To speedup the convolution process even more, the separability of the Gaussian kernel can be used. Thanks to this, it is possible to convolute the image with horizontal filter $[1, -4, 1]$ at first and then once more with the same filter rotated by 90 degrees. The result is the same as the convolution with the kernel in Table 7.2 with a smaller computational cost.

1	2	1
2	4	2
1	2	1

Table 7.1. Discrete Gaussian convolution kernel

0	1	0
1	-4	1
0	1	0

Table 7.2. Discrete Laplacian of Gaussian convolution kernel

7.2.2 Histogram Equalization

Second approach to image filtration in our application is histogram equalization. The source image for storing the result of the colorization is filtered with this filter, as can be seen on Figure 7.1. Also histogram equalization is used to darken the output of the LoG filter.

The goal is to remap intensities in the image so that the darkest intensity is black and the lightest intensity is white. For this purpose, lower and upper bound is found and then the intensities of the image are linearly remapped between these two bounds. The histogram $H(I)$ of the greyscale image I is calculated. Then the darkest intensity (upper bound) p with the highest histogram value is retrieved. The lower bound l is the lowest intensity which appears in the image.

Each pixel intensity $I(x, y)$ is recalculated the same way as on the Figure 7.4 a. Here we see histogram of the picture on Figure 7.4 a. All pixels lying between lower bound l and upper bound p are linearly rescaled. All pixel intensities below the lower bound are clamped to 0; all pixels above the upper bound are clamped to 255. The result can be seen in Figure 7.4 b.

The output of the LoG filter applied to the source image can be seen on Figure 7.4 c. It can be easily seen that the outlines are very bright, so the histogram equalization is used to darken these outlines. The result of the histogram equalization of this image can be seen on Figure 7.4 d. This image is used for LazyBrush colorization calculation, while the image on Figure 7.4 b is colorized and presented as the result image. The image on Figure 7.4 d cannot be presented as a result, because the edges detected with LoG filter look unnatural.

7.3 Automatic Image Cropping

As was defined in Chapter 5, in use-case (ID: 1.1.1) the photographed image shall be automatically cropped and straightened to compensate the inaccurate position of the camera during photo capture. The intention is to obtain an undistorted paper with the drawing without the background of the paper.

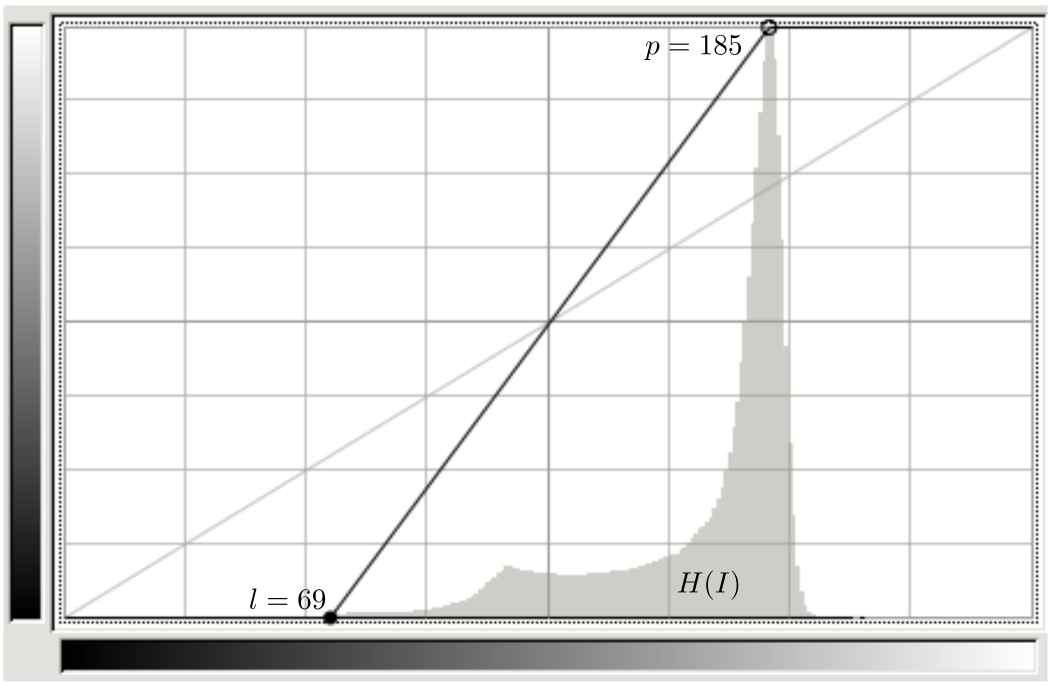


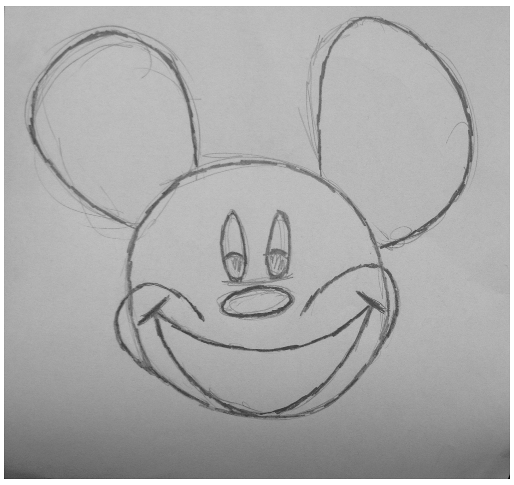
Figure 7.3. Visualisation of linear rescale of image intensities

To crop the image, a two-phase approach was used. Firstly, the edges in the image are detected with the edge detection filter. The edges of the paper should be the longest straight edges in the image assuming the white paper on a relatively dark surface. Secondly, the edges are recognized using the Hough transformation [27]. The Hough transformation is a robust method for recognition of objects that can be expressed by an analytic equation. In our case, the Hough transformation is used for the recognition of lines that create paper boundaries. The method is independent of the edge orientation so the paper on the photo can be rotated and perspectively distorted. After the lines creating the borders of the paper are recognized, the corners of the paper are calculated and the back perspective transformation is applied on the image to obtain undistorted image. The whole process is built with following assumptions.

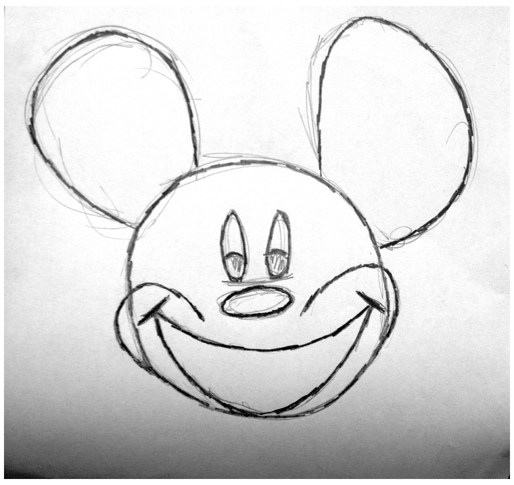
- The paper lies on surface that ensures that the paper boundaries are well visible (i.e. a white paper on a dark table). The surface is plain, thus the paper is not bended or deformed.
- The whole boundary of the paper is visible (i.e. all four edges of the paper can be registered). If not, the image is not cropped.
- The paper that is being cropped and straightened has the same aspect ratio as the A4 paper has—that is 210 : 297.

7.3.1 Edge Detection

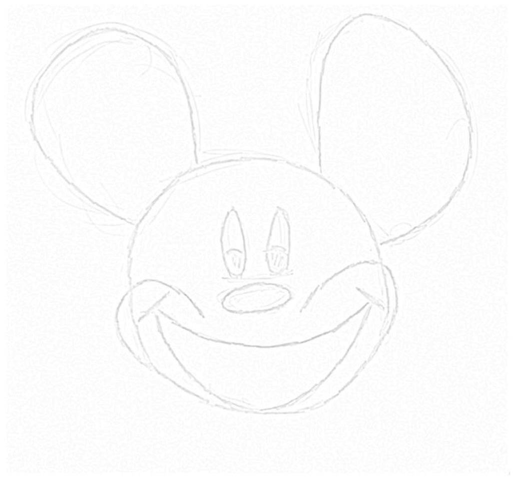
Prior to the registration of the paper edges, the edges must be enhanced to make the registration more reliable. This is done with LoG filter the same way as was defined in Section 7.2.



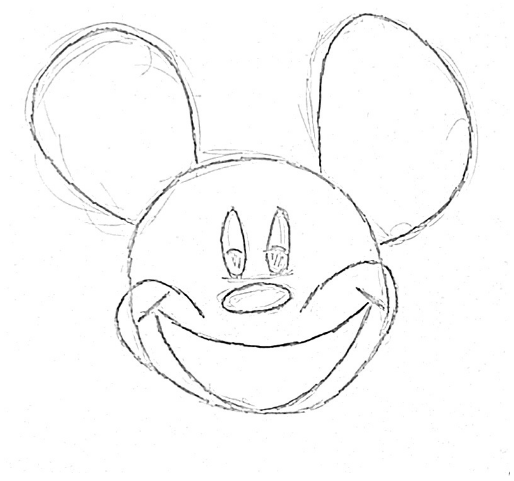
a) Source photo



b) Source photo after histogram equalization



c) Source photo filtered with of LoG filter



d) Source photo after application of LoG filter and histogram equalization

Figure 7.4. LoG filter and histogram equalization applied on source photo

7.3.2 Cropping

When the edges are enhanced by the LoG filteri, the boundaries of the paper must be recognized and the image is cropped. For this purpose, the Hough transformation is used. The image is transformed to Hough space to detect lines. Four strongest lines that are reasonably far away each other are then considered as edges of the paper. Finally, the lines are sorted to counter-clockwise order and intersections of these lines are found. These intersections are considered as corners of the paper.

For the Hough space transformation the polar representation of lines is used. Each line in $\mathbb{R}^2$ space is therefore represented by angle θ and radius ρ from origin $O \in [0, 0]$ (see Figure 7.5), so $x \cdot \cos(\theta) + y \cdot \sin(\theta) = \rho$. The accumulator array $A \in \mathbb{R}^{m,n}$ of the Hough space has the same dimension as the source image I, therefore $m = w, n = h$, where m is the width of A, n is the height of A, w is the width I and h is the height of I. Each bin $i \in \{0, 1, ..., m\}$ on the x-axis of A corresponds to the angle θ_i and each bin $j \in \{0, 1, ..., n\}$ on the y-axis of A corresponds to the radius ρ_j. To define a line uniquely in $\mathbb{R}^2$ space, the angle $\theta \in \langle 0, \Pi \rangle$. Therefore the sample $\Delta\theta$ of θ is defined as $\Delta\theta = \frac{\Pi}{m}$. Since radius $\rho \in \langle -\sqrt{m^2+n^2}, \sqrt{m^2+n^2} \rangle$, the sample $\Delta\rho$ of ρ is defined as: $\Delta\rho = \frac{2 \cdot \sqrt{m^2+n^2}}{n}$. The algorithm is listed in Figure 7.6. The variable *dtheta* is equal to $\Delta\theta$, the variable *dr* is equal to $\Delta\rho$.

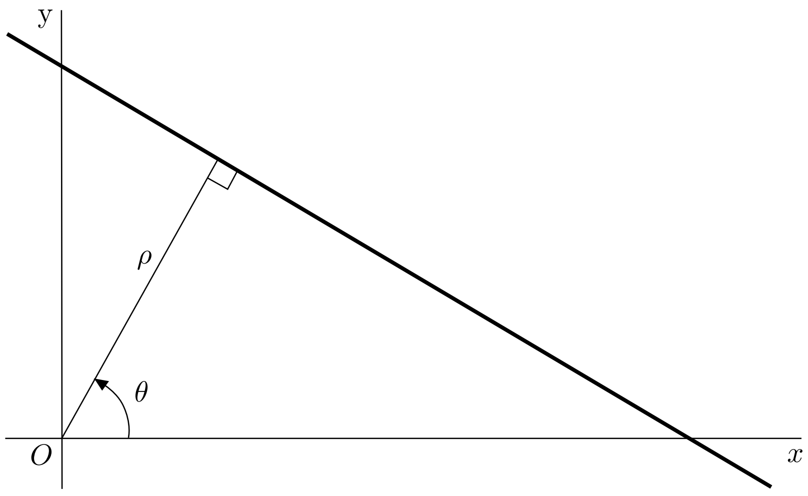


Figure 7.5. Polar representation of a line

After the edge pixels of the image I are transformed via Hough transformation into the accumulation array A, four strongest lines that are reasonably far away from each other and have reasonably big angle between each other to form a page boundary are extracted repeating following two steps:

1. The maximum value is found at indices i, j in the array A. The i, j indices denotes the angle $\theta_i = i \cdot \Delta\theta$ and the radius $\rho_j = j \cdot \Delta\rho$. The angle θ_i and the radius ρ_j define a line with respect to the origin O.
2. The array A is nulled for radiuses near ρ_j and angles near θ_i. For our purposes:

$$\forall p \in \{i - m/6, ..., i + m/6\}, \forall q \in \{j - n/6, ..., j + n/6\} : A(p, q) = 0.$$

To find four edges, steps 1. and 2. are repeated four times.

```
for (y = 0; y < n; ++y){
    for (x = 0; x < m; ++x){
        if (I[x, y] is Edge pixel){
            for (i = 0; i < m; ++i){
                theta = dtheta * i;
                r = x * cos(theta) + y * sin(theta);
                j = (r / dr) + (n / 2);
                ++A[i, j];
            }
        }
    }
}
```

Figure 7.6. Hough transformation pseudocode

After the angle θ_i and the radius ρ_j that define the boundary line with respect to the origin O are found, the line is transformed to a parametric form (i. e. two points defining the line are found). This is needed for easier calculations with boundary lines to sort them and to find intersections of these lines that define the corners of the paper being registered. To obtain two points $[x_1, y_1], [x_2, y_2]$ from the polar line representation, we choose values $x_1 = 0, x_2 = m$ in case the $\theta_i \geq \Pi/4 \wedge \theta_i < 3/4 \cdot \Pi$. Otherwise, we choose $y_1 = 0, y_2 = n$. Corresponding variables y_1, y_2, or x_1, x_2 are then calculated by following equations:

$$y_k = \frac{\rho_j - x_k \cdot \cos(\theta_i)}{\sin(\theta_i)}, k = \{1, 2\},$$

or:

$$x_k = \frac{\rho_j - y_k \cdot \sin(\theta_i)}{\cos(\theta_i)}, k = \{1, 2\}.$$

Then the edges are sorted counter-clockwise, starting by the most left edge. For this purpose, centroid $m[m_x, m_y]$ of each edge is calculated: $m_x = (x_1 + x_2)/2$, $m_y = (y_1 + y_2)/2$. The most left edge e_l has a centroid with minimal m_x, the most right edge e_r has a centroid with maximal m_x, the most bottom edge e_b has a centroid with maximal m_y and the top edge e_t has a centroid with minimal m_y. Then four intersection points are calculated. Because the edges are sorted counter-clockwise, we calculate the intersection i_1 between edges e_l and e_t, intersection i_2 between edges e_l and e_b, intersection i_3 between edges e_b and e_r and intersection i_4 between edges e_r and e_t. Thus the counter-clockwise oriented points defining corners of the paper were obtained.

Now the inverse perspective transformation is computed and the image is transformed to obtain an undistorted image. With the assumption that the paper with the same aspect ratio as the A4 paper is being recognized, we set destination points of corners of the desired image for the transformation. The destination points are simply the corners of the biggest rectangle that fits the source image with the aspect ratio of the A4 paper. In case the paper being registered is portrait oriented, the destination points in the counter-clockwise orientation beginning with the top left one are: $a_1 = [0, 0], a_2 = [0, t], a_3 = [w, t], a_4 = [w, 0]$ where w is the width of the source image and $t = 297 \cdot (w/210)$ (to conform to the aspect ratio of the A4 paper). In case the paper being registered is landscape oriented, the $t = 210 \cdot (w/297)$. Now the homography matrix H between corresponding points $i_k => a_k, k = \{1, 2, 3, 4\}$ must be calculated. For such calculation, the OpenCV implementation of RANSAC was used. After the homography is found, the image is transformed with the inverse perspective transformation with the homography matrix H. The result image is then cropped — the pixels inside the rectangle denoted by the destination points of its corners $a_k, k = \{1, 2, 3, 4\}$ are the final image.

Results of the whole process—the edge recognition, the back perspective transformation, the crop and the image filtration with pixel intensity rescale can be seen on Figure 7.7, resp. on Figure 7.8.

7.4 LazyBrush Algorithm Implementation

The application performance is of a great importance. Therefore the LazyBrush algorithm has to be as fast as possible. The most time consuming task in the LazyBrush algorithm is the image segmentation. During every colorization, the image

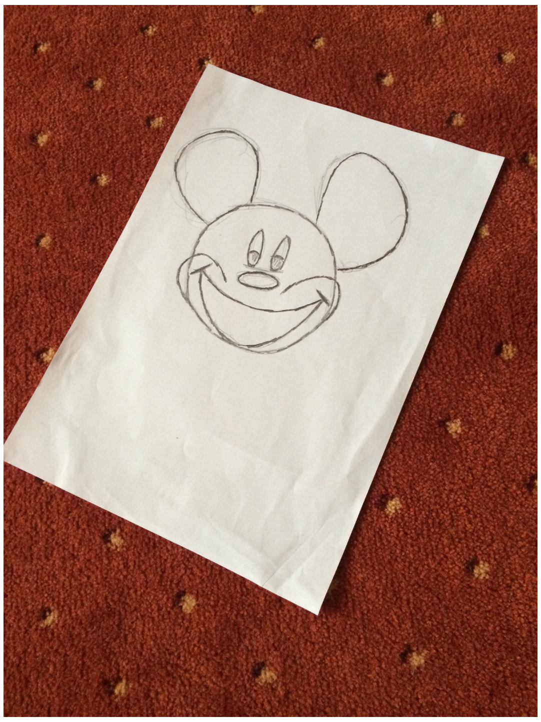


Figure 7.7. Original image to be cropped

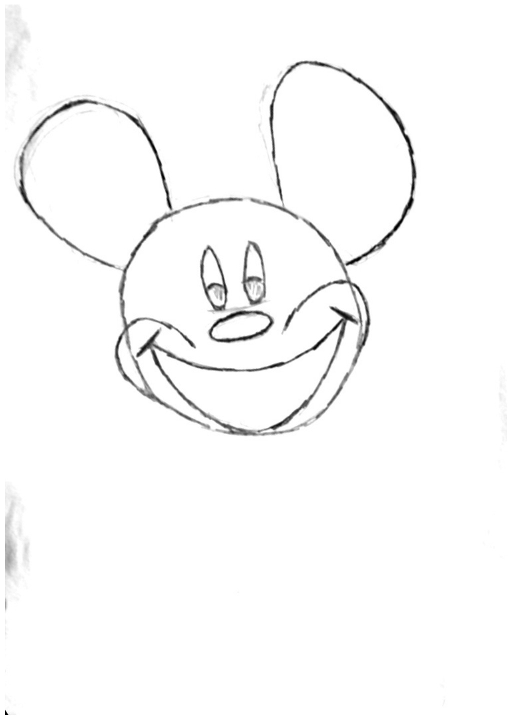


Figure 7.8. Image transformed with homography matrix H, cropped according to the destination points a_k, greyscaled and filtered with the histogram equalization (the intensity rescale).

segmentation is called as many times as there is the number of different colors (scribbles). Therefore, the very efficient implementation of this task is needed. The image segmentation problem is solved by the maxflow/mincut problem as was stated in Chapter 4.

One of the most known algorithms for the maxflow/mincut problem is the Ford-Fulkerson algorithm [28]. Advantage of this algorithm is that is general — it can be used on arbitrary graphs; disadvantage is that it is slow. For computer vision, further faster algorithms were addressed. Boykov and Kolmogorov [29] defined a new algorithm for maxflow, much faster than the Ford-Fulkerson. The improvement was made mainly by improving the method for searching augmenting paths. Kolmogorov provided free implementation of the algorithm[1]) for academic purposes. Another implementation of this algorithm can be found in boost library[2]) which is open source and free to use — but to our knowledge much slower than Kolmogorov implementation. Implementation with further improvements was provided by Jamriška and Sýkora known as GridCut library[3]). This library states that it is up to 8 times faster than the Boykov-Kolmogorov implementation. As the requirements on library performance are high, the benchmark comparing the Boykov-Kolmogorov implementation and the GridCut library on iPad 3 and iPhone 5s has been done in Section 8.3. The GridCut library performed much better than the Boykov-Kolmogorov implementation which implies the GridCut library was used in the implementation of the LazyBrush algorithm on iOS.

1) `http://pub.ist.ac.at/~vnk/software.html`
2) `http://www.boost.org/doc/libs/1_54_0/libs/graph/doc/boykov_kolmogorov_max_flow.html`
3) `http://gridcut.com`

The LazyBrush algorithm was implemented as a single class named SolverManager. The class collaboration diagram can be found on Figure 7.9. The workflow to use this class to colorize a greyscale image is following:

- Create a new instance of SolverManager. Pass the greyscale image and the array of scribbles into the constructor.
- Run the solve method.
- New colorized image is returned when the solution is found.

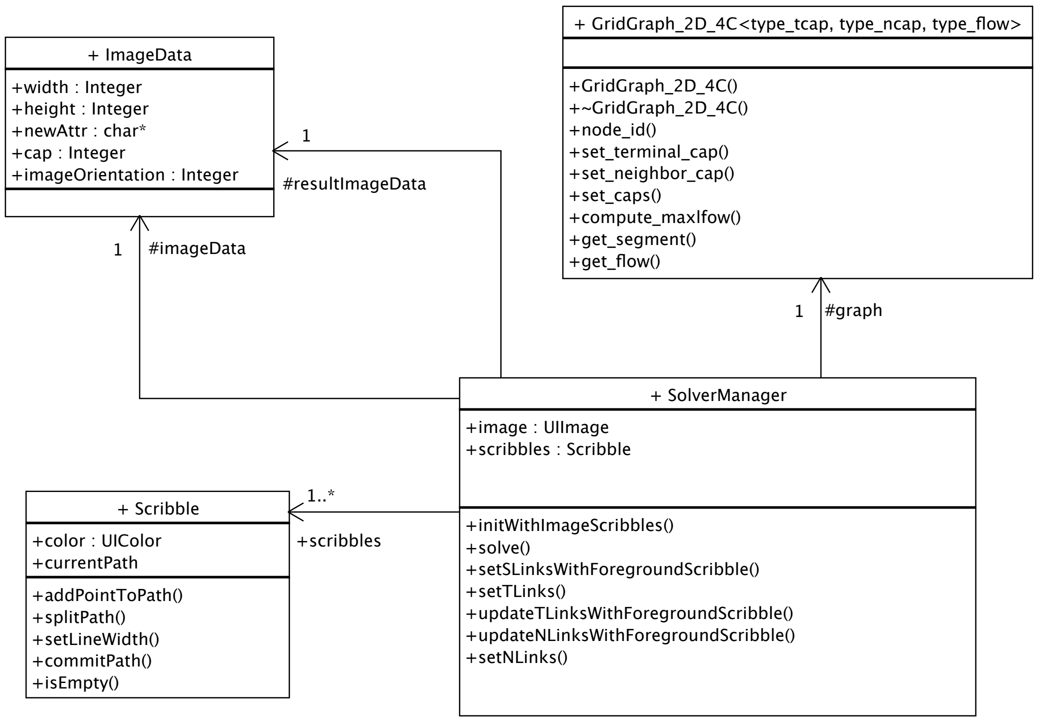


Figure 7.9. Solver manager class diagram

For simple usage of SolverManager, the type of image passed into the solver and the image returned by the solver is always UIImage. This is a high level image on iOS that can be easily shown on any view. Raw image data (e.g. the pixel values) cannot be accessed directly from UIImage. To address this problem, a function for creating raw image data (ImageData structure on Figure 7.9) was implemented. It creates a new CoreGraphics context with a C array of unsigned chars (the image pixels), draws the UIImage into this context and packs the raw data into the ImageData structure with additional information about the image like width, height, orientation, etc. Also, after the image was colored, there is a need to pack the raw data back into an UIImage, so respective function for this functionality was implemented as well. The conversion of the input UIImage to ImageData is done in the constructor of the SolverManager. The constructor also makes a deep copy of the array of scribbles. During the solve method, the scribbles are deleted one by one. This implies that the SolverManager instance can be used only once, then must be deleted. The capacities for the GridCut graph are stored in six capacity C arrays. First four arrays are for all directions (top, bottom, left, right) and two arrays are for storing capacities from the start node and to the terminal node. Instead of allocating six arrays separatelyi, only one is allocated and respective pointers are calculated to this one array.

After SolverManager is constructed, the solve method is called. At first, capacities between pixels in 4-neighbourhood (N-links) by method *setNLinks* are calculated. In facti, for the best efficiency, the capacities are precalculated for each intensity level and then we lookup the capacity for the current pixel intensity in an array. The capacity C_i for an intensity level i is calculated according to the formula (1).

$$\forall i \in \{0, 1, 2, ..., 255\} : C_i = 1 + K \cdot \left(\frac{i}{256}\right)^2, \tag{1}$$

where $K = 2 \cdot 800 \cdot 600 = 960000$. This setting is based on the described setting in Chapter 4. The constant K is based on predicted width and height $800 \times 600px$ for all images. The capacity $w_{i,j}$ for an N-link between pixels with intensities $p_i, p_j \in \{0, 1, 2, ..., 255\}$ is set according to the formula (2).

$$w_{i,j} = \min(C_{p_i}, C_{p_j}). \tag{2}$$

Then capacities of all edges connecting all pixels with a terminal node are calculated in the *setTLinks* method . For each scribble, the pixels labeled by this scribble are obtained. Each capacity of the edge connecting labeled pixels and the terminal node (the edge is called T-link) is set to the constant $S = K \cdot (1 - \lambda) \approx K \cdot (1 - 0.95) = 960000 \cdot 0.05 = 48000$. Other edges are by default set to zero. Then the loopi, until there are no more scribbles to process, is run. In the beginning of each iterationi, one scribble as a foreground scribble is chosen and new graph from the GridCut library is created. Capacities of edges between pixels labeled by the foreground scribble and the start node (edges called S-links) are set to the constant S in the *setSLinksWithForegroundScribble* method. T-links of pixels labeled by the foreground scribble are set to zero in the *updateTLinksWithForegroundScribble* method. The capacity arrays are then set to the graph and the maxflow is calculated on this graph by the GridCut library. At the end of the iteration, the segmentation is obtained from the graph. In the *updateNLinksWithForegroundScribble* method, the pixels belonging to the start node are colored with the color of the foreground scribble and N-links connecting the colored pixels with the pixels without a color are set to zero (disconnected from the graph). Also, T-links going to and from the colored pixels are set to zero. At last, the foreground scribble is removed from the array of scribbles and the loop starts again in case there is at least one scribble in the array of scribbles.

When the array of scribbles is empty, the image is fully colored. A new UIImage is created from the colored ImageData structure and is returned from the SolverManager. After the solve method finished, SolverManager is destroyed and the colored image is shown on the device screen.

7.5 Drawing Scribbles

To label pixels with arbitrary colors for the LazyBrush algorithm, the scribbles are drawn on top of the picture. Single scribble consists of one or more brush strokes. These strokes are drawn by the user with touch gestures. As the scribbles influence the final colorization of the imagei, the user experience of drawing scribbles is essential. The implementation must deal with two main problems. The first one is the problem of the interpolation of the points given by the touch gesture to create a smooth spline. The second problem is how to draw the scribbles interactively, but smooth as possible without any lags. This seems to be easy thing – nowadays devices like iPhone 5S have so much power that poor implementations does not affect the visual drawing performance much. The problem can be seen on older devices like iPad 3. Naive implementations using the CoreGraphics iOS framework cannot provide smooth drawing without lags.

7.5.1 UIBezierPath Scribble Implementation

At firsti, an attempt to implement the scribble rendering with high level UIBezierPath has been made. The class diagram on Figure 7.10 describes the architecture of this part of the application. The scribble consists of zero-or-more paths. Each path was UIBezierPath which contains its leading points, color, width and other attributes and can be rendered. The problem with the usage of UIBezierPath is the rendering performance. The UIBezierPath can be rendered partly into defined rectangle or as a whole. Therefore, when the user draws a part of the stroke, the UIBezierPath has to be divided into smaller segments (of type UIBezierPath). Then only the last added segment can be rendered without the whole UIBezierPath. Another problem is that when we want to divide bezier paths into segments which should be rendered only once we have to render those segments into bitmap instead of drawing them directly into the screen context. The reason is that when the view is going to be redrawn the context is cleared automatically so without drawing the scribbles into the bitmap only the last segment would be visible. The fact is that if we render the bezier path into bitmap and then the bitmap is drawn onto the screen it takes more time than if only the last segment could be drawn directly into the screen context.

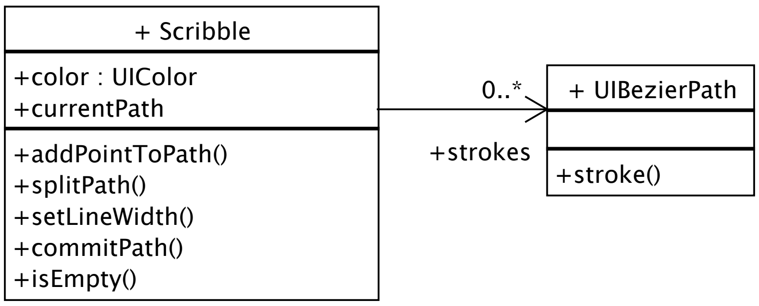


Figure 7.10. Scribble class diagram with UIBezierPath

The rendering speed of CGPath as a type from the CoreGraphics framework was tested as well. The usage of CGPath is very similar to UIBezierPath — again the paths have to be rendered into bitmap to avoid re-rendering the whole path over again. According to the performance tests made through an Instruments profiling application, the application really spends 95% of the time with the CoreGraphics rendering routines. The visual performance was not different from the implementation that used UIBezierPath since UIBezierPath is an Objective-C wrapper for CGPath.

7.5.2 OpenGL ES 2 Scribble Implementation

Due to these facts, the rendering was reimplemented to use OpenGL ES 2.0. The difference is well observable on older devices like iPad 3. The main difference is that since OpenGL uses the GPU chip for rendering the CPU utilization is much lower than with UIBezierPath. The CPU utilization for an implementation with UIBezierPath is around 70% and for an implementation with OpenGL it is around 12%. This is the reason why the implementation using UIBezierPath is lagging.

When drawing a spline, CPU must do mainly two things. The first one is to handle the touch events and to calculate the spline leading points, the second one is the drawing. As the rendering through CoreGraphics is CPU intensive and takes a big amount of time, the application cannot handle so many touch events. Therefore, the leading points are further apart and the spline is created from bigger blocks. This makes the lag behavior. On the other side, the OpenGL implementation handles the rendering on the GPU chip which is faster a lot. The application samples more touch events and the spline is created from very small blocks. This makes the spline being drawn smoothly.

For OpenGL ES scribble custom class GLQuadraticPath has been implemented. This class interpolates leading points with quadratic curve and renders the calculated spline with OpenGL ES 2. As the points are added into the GLQuadraticPath, new points of spline segment are calculated. The calculation was derived from the formula of the quadratic function (3).

$$f(x) = ax^2 + bx + c \tag{3}$$

Consider points $\{q, w, z\} \in \mathbb{R}^2$ obtained from the touchscreen in this order. The task is to approximate a quadratic spline from these points. The method is defined in (4) and (5).

$$\begin{aligned} m_1 &= \frac{w-q}{2} \\ m_2 &= \frac{z-w}{2} \end{aligned} \tag{4}$$

$$\begin{aligned} \forall t \in \langle 0, 1 \rangle : p(t) &= (m_2 - w)t^2 + (w - m_1)t + m_1 \\ &= m_2t^2 - wt^2 + wt - m_1t + m_1 \\ &= m_2t^2 + wt(1-t) + m_1(1-t) \\ &= m_2t^2 + (1-t)(wt + m_1) \end{aligned} \tag{5}$$

The situation can be seen on Figure 7.11. At first, we calculate the centroid m_1 of the first and the second point and the centroid m_2 of the second and the third point. Then for every parameter t, the equation (5) is evaluated. In practice, the interval $\langle 0, 1 \rangle$ is sampled by number of points on the segment which is calculated according to distance between first and second and between second and third point. The calculated points are passed to the OpenGL renderer. Each point is rendered as a single singular texture with the thickness of the stroke.

The helper class GLInfo holds the initialized OpenGL objects and the OpenGL context so the GLQuadraticPath can render its contents via OpenGL. The Scribble class diagram using GLQuadraticPath can be found on Figure 7.12. Each time new point is added to the GLQuadraticPath with an *addCurveToPoint* method, a new spline segment is created from the points calculated by formula (4) and (5). These points are stored in the vertices array of floats. This array is then stored into the graphics memory via buffer object and rendered. The big advantage of the OpenGL framebuffer is that it must be explicitly cleared when needed. This means that only new segments are drawn because the ones that were already drawn remain in the framebuffer. The GLQuadraticPath also has a method *render* that renders the whole path. This is usable when the screen is cleared and the scribbles has to be all rendered to be visible.

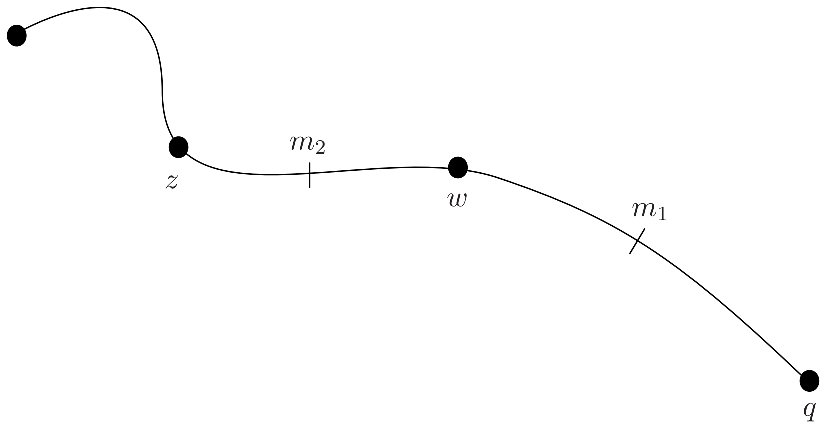


Figure 7.11. Quadratic Spline Interpolation

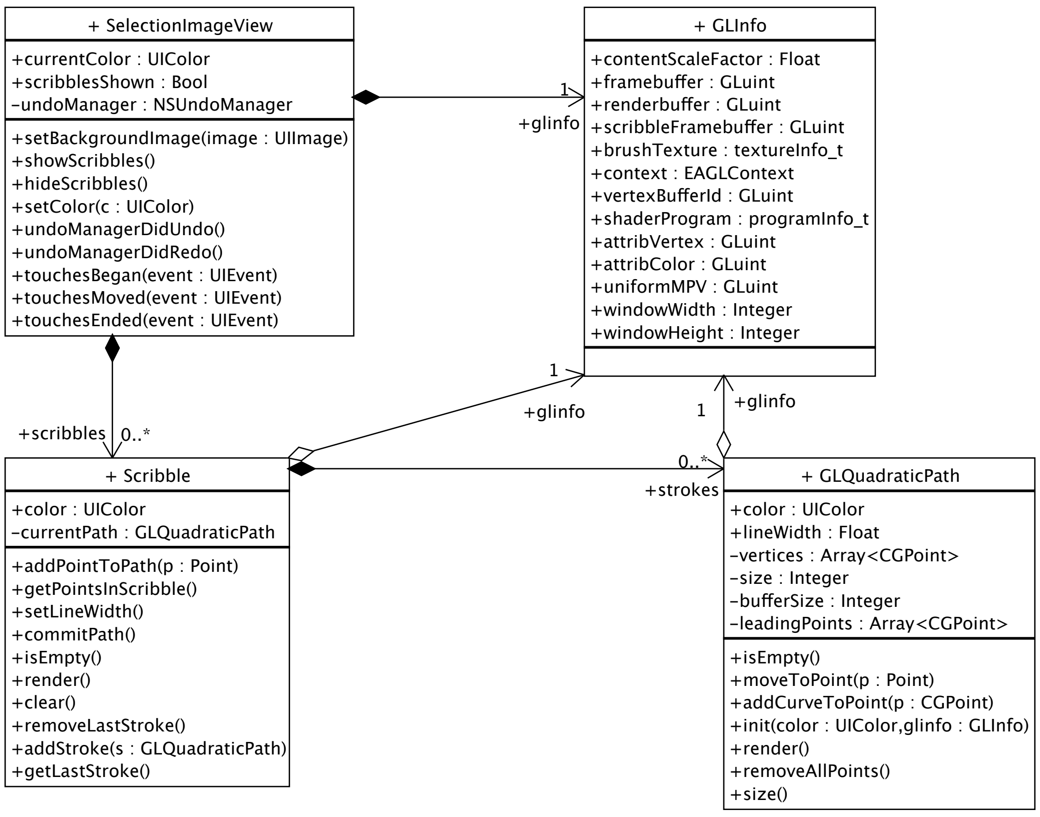


Figure 7.12. Scribble class diagram with GLQuadraticPath

The scribble manages all strokes (GLQuadraticPath) that have the same color. Therefore, the scribble consists of zero or many GLQuadraticPath objects. Besides the stroke management, the scribble object is also responsible for finding all points (pixel coordinates) in the scribble. This means that all rendered points from all brush strokes has to be found. This functionality is implemented in the *getPointsInScribble* method. Basically, the implementation iterates over all the GLQuadraticPath objects in the scribble and renders each in the off-screen renderbuffer. Then it gets the rendered image for each stroke, iterates over the image and stores the pixels which have different color than the background in the result array. This method is called from the SolverManager mentioned earlier when the hard constraints of the LazyBrush algorithm must be set.

OpenGL is initialized in a SelectionImageView which holds collection of Scribbles. This view initializes the GLInfo object and passes it to Scribble which passes it to newly created paths. SelectionImageView responds to touch events and manages scribble creation when needed. It also handles the undo manager — it removes last stroke from scribble when undo event occurs and adds back the stroke to the scribble when redo event occurs. It can also show or hide all the scribbles by clearing the framebuffer or rendering all the scribbles again.

Chapter 8
Results

In this work the LazyBrush application for iOS has been implemented. Through this effort, it is shown that such application for the colorization is able to run with an interactive response on touchscreen devices. The colorization application on smartphone or tablet has its advantages — it can be always with the user, the drawing that the user wants to color can be simply captured by the device's camera so no downloading or sending of the material is needed. The use of LazyBrush algorithm is advantageous as well, because the LazyBrush algorithm is independent on a drawing style or a drawing tool. Therefore, a variety of images on variety of occasions can be colorized. The application was intended and designed mainly for kids since many kids like drawing a lot so there is plenty of potential users of this application.

8.1 LazyBrush Algorithm Testing

To test if the implemented LazyBrush algorithm runs properly, original images 8.2, 8.4, 8.6 8.8, 8.10 and 8.12 from [14] were colorized. As we obtained the images from the author of the article [14] in digital form, the images were only opened in the iOS application, greyscaled, linearly rescaled with lower and upper bound as any other image and colorized. The input for LazyBrush of these images can be seen on Figures on the left side: 8.1, 8.3, 8.5, 8.7, 8.9 and 8.11. The bordering rectangles on Figures with input are automatically generated scribbles to obtain white background. From these images, it can easily be seen that the LazyBrush implementation works as expected.

8.2 Usability Testing

After all main functionalities according to requirements analysis in Chapter 5 were implemented, the application has been tested formally with several children. The main design flaws were addressed in the final user interface presented in Chapter 6.

The tutorial was found as the most problematic part of the application. While older users usually go through the tutorial without a problem, children usually make some mistake because they do not read the instructions carefully and make quick irresponsible decisions. Second main problem was identified with soft scribbles — sometimes children make too small brush strokes that does not affect the final colorization because they are too small to gain majority weight and cause some part of the image to be colored, and sometimes they make the brush strokes unreasonably big so the color fills a larger area, than they actually want ed to. Despite these errors, children were able to colorize their image in reasonable amount of time and almost all of them stated that the colorization with this application was fun.

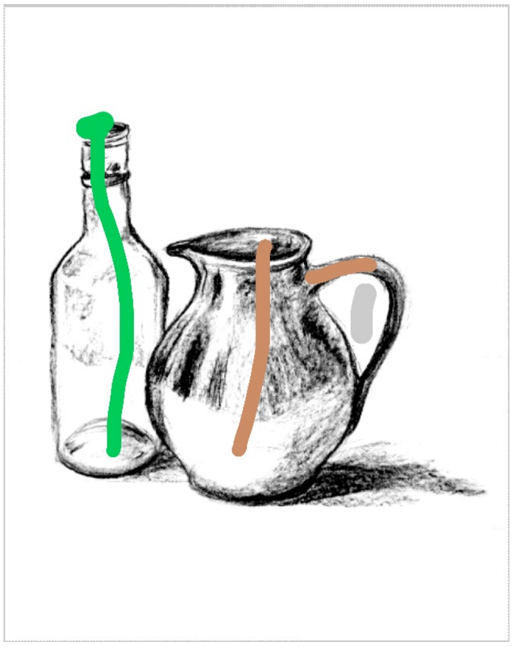


Figure 8.1. Bottle with scribbles

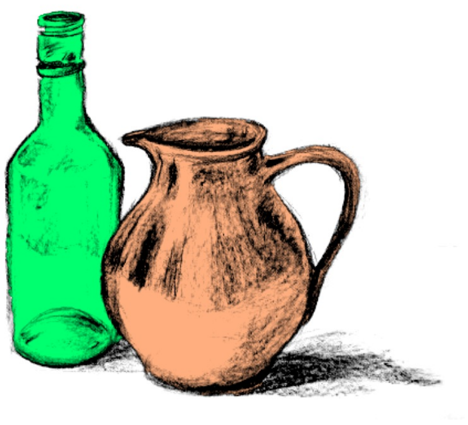


Figure 8.2. Bottle

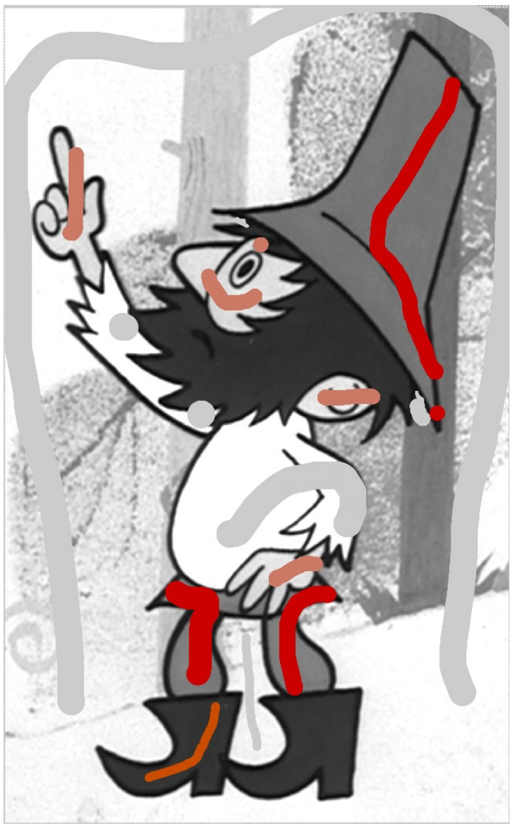


Figure 8.3. Robber with scribbles

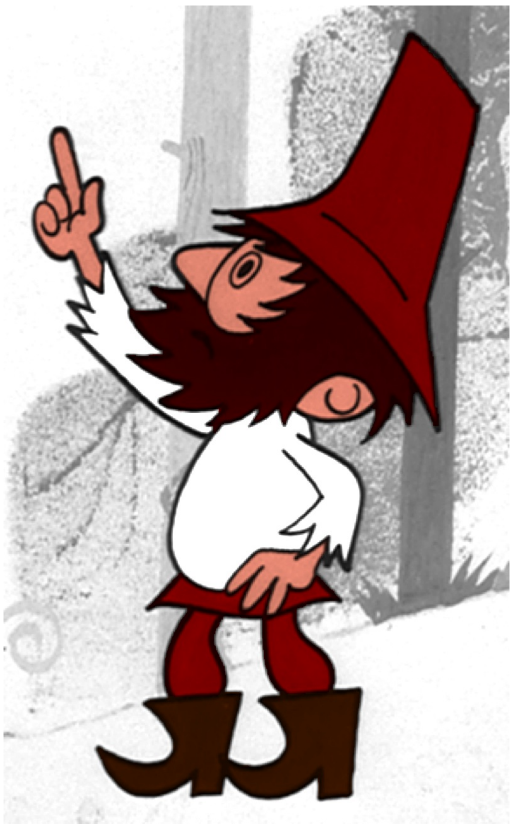


Figure 8.4. Robber

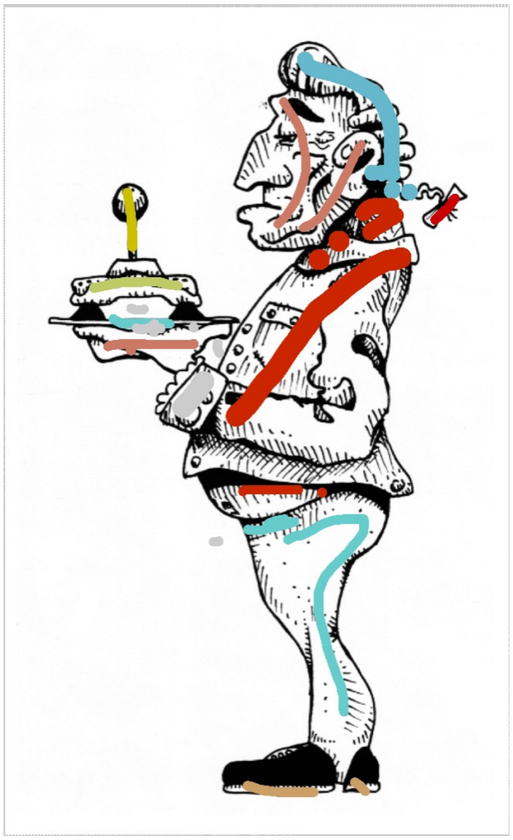


Figure 8.5. Footman with scribbles

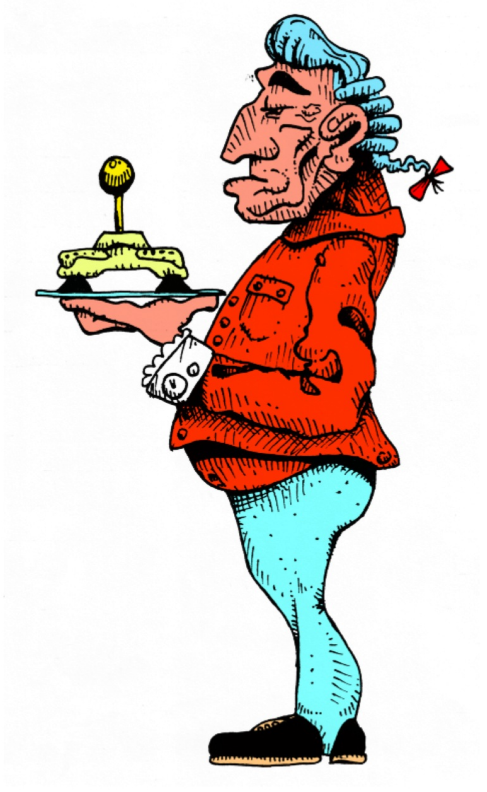


Figure 8.6. Footman

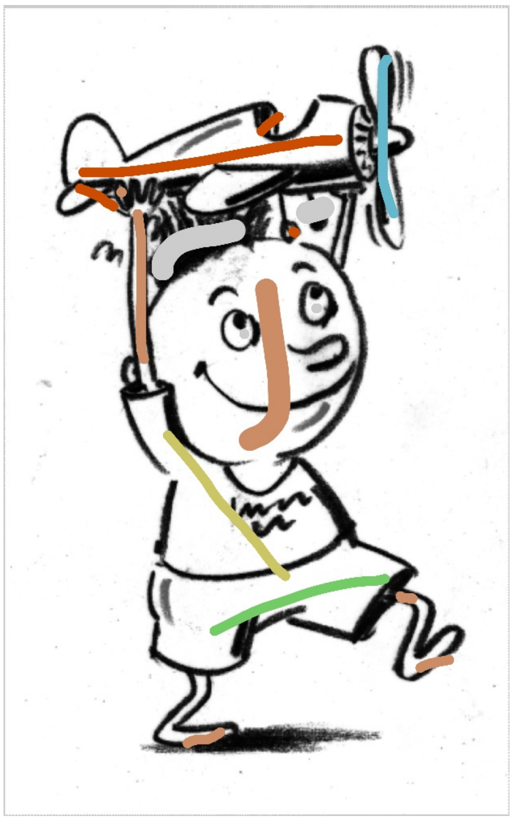


Figure 8.7. Boy with scribbles

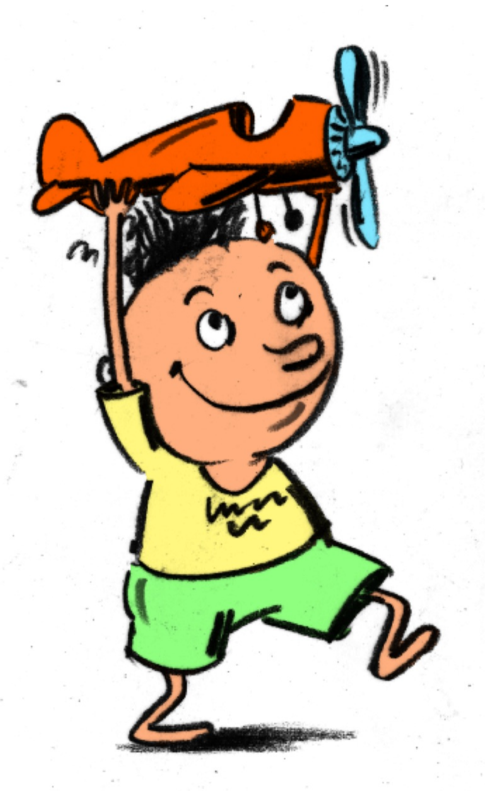


Figure 8.8. Boy

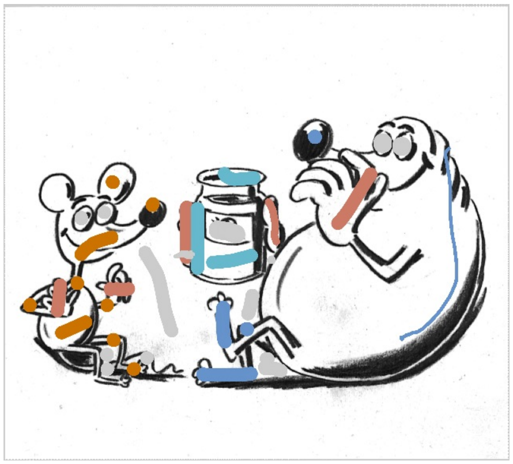


Figure 8.9. Picnic with scribbles

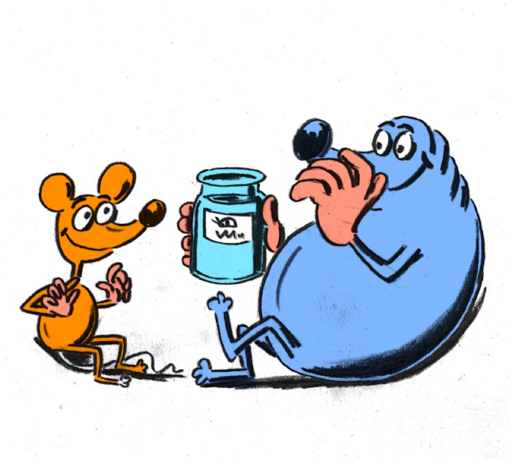


Figure 8.10. Picnic

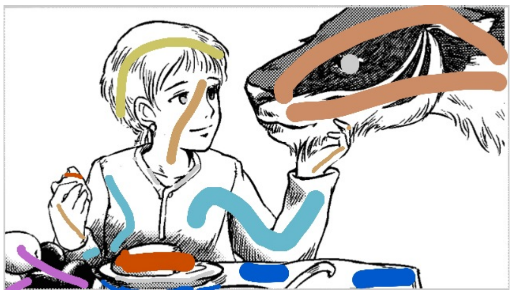


Figure 8.11. Manga with scribbles

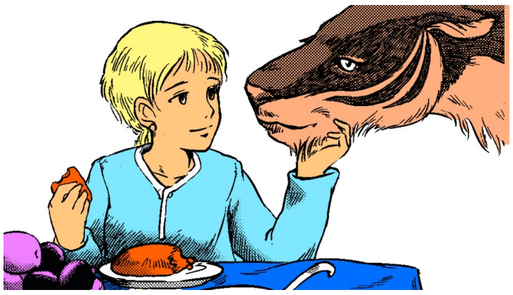


Figure 8.12. Manga

8.2.1 Target User Group

The target group for testing was the same as for designing the application. The target group is therefore children 7 - 15 years old who like to draw or colorize pictures and who like to use touchscreen devices like smartphones or tablets. The secondary target group is all other people who enjoy to draw images, or who need to draw images and colorize them for their work. These people shall also have a positive attitude to touchscreen devices and use them on an everyday basis.

8.2.2 Test Setup

The test was done with iPhone 5s device with installed release version of the application. All tests took part in afternoon/evening hours of late spring so the artificial lighting had to be used. This made the test more difficult because the user had to take the photo of his picture that way that no drop shadow waas visible on the paper (of which the users were aware by their common sense automatically, without the need to give them an advice). Participants were chosen according to the screener which was made verbally not to distract the children with writing. The children need to fit one of age groups: 7 - 9, 10 - 12, 12 - 15, they should have a positive attitude to drawing pictures and should have a minor experience with using iOS/Android touchscreen device. The process of the test was captured on a video with an iPad 3 device. The camera captured the hands with the screen only, not the entire user. All the users voluntarily agreed that they take part in the test and that they will be captured on a video which will not be presented to third parties.

8.2.3 Participants

The overview of participants that took part in the test is available in Table 8.1. There was a problem to find respondents that would cover the whole target group. There was no girl as a child participant, and generally, there is very few children that have experience with iOS devices — Android devices are more common in the Czech republic.

Number	Age	Sex	Platform experience
1	9	male	little experience with iOS
2	9	male	little experience with Android
3	9	male	no experience with touchscreen devices
4	11	male	little experience with Android
5	12	male	daily experience with Android
6	12	male	no experience with Android

Table 8.1. Overview of participants that took a part in the test

8.2.4 Test Assignment

Each participant was told to draw a picture; the tool selection (pen or pencil) was up to the participant. The picture should be as big as possible. After the picture was drawn, the participant was given an iPhone 5s phone with opened application. At first, the tutorial was shown and the user was expected to go through the tutorial. After the tutorial was finished, the user was told that at that moment he should be prepared to take a photo of his/her drawing and colorize it with the application. As the colorization is finished, the picture should be saved to the device image gallery.

8.2.5 Findings

The findings are sorted according to the priority. The most important problems are listed at first. The ordering is based on how serious the finding is with a connection to the usability. At each finding the participant who experienced the problem is given. Also, the information if the problem was fixed and how is given.

- **Priority:** High.
- **Problem:** Because of his little fingers, he has difficulty to hold the phone while taking a photo of his picture. Therefore, the image was not positioned exactly over the paper with the drawing and the application did not cropped the photo.
- **Proposed solution:** The application has to be able to remove big perspective disortion and rotation, so it has to be able to recognize the paper on the photo in arbitrary position (i. e. rotated and taken from a side under the big angle).
- **Found by participant:** 1.
- **Fixed:** Yes. The paper boundary recognition has been implemented with the Hough transformation (see Section 7.3) which is robust and thus able to recognize rotated and perspective disorted edges of the paper.

- **Priority:** High.
- **Problem:** While trying to take photo on landscape orientation, the view rotates but the camera preview does not rotate. Therefore, only part of the preview is visible.
- **Proposed solution:** Force the preview to rotate as well or disable rotation of the screen while taking photo.
- **Found by participant:** 5
- **Fixed:** Yes, the rotation was disabled for this view.

- **Priority:** High.
- **Problem:** When the application shows the animation of drawing brush stroke, the user does not understand what he shall do. He is confused that the example brush stroke made by the application during animation does not disappear and expects further animations.
- **Proposed solution:** The brush stroke drawn by the application as an example stroke shall dissappear as soon as it is drawn completely.
- **Found by participant:** 3
- **Fixed:** Yes, after the brush stroke is drawn, the application waits for 0.3 seconds and then removes it.

- **Priority:** High.
- **Problem:** The user accidentally swipes from the left edge of the screen and thus make the gesture that goes to the previous screen (the main application dashboard) so his work is lost.
- **Proposed solution:** Disallow the swipe gesture that goes to the previous screen.
- **Found by participant:** 6
- **Fixed:** Yes, swipe gesture to go to the previous screen was disallowed on this view.

- **Priority:** High.
- **Problem:** The user did not know that the whole paper with its borders should be visible so the system was not able to crop the photo. He was not aware of this feature.
- **Proposed solution:** The user should be noticed how to position the paper with the drawing on the photo so the system can crop the image appropriately. Also, the user should know that the application will automatically enhance the image to make better contrast between the background and outlines.
- **Found by participant:** 1, 2, 3, 4, 5, 6.
- **Fixed:** No.

- **Priority:** High.
- **Problem:** The user went through the tutorial so quickly he almost could not espy what he is really doing. Therefore, he touched the screen many times and the application was too slow to present all the instructions. The user touched the screen even before the instruction appeared so the instruction disappeared in fact before it fully appeared. Also, the user accidentally draw a brush stroke while he should zoom.

- **Proposed solution:** User interaction must be enabled only when the instruction was fully shown. Also only the type of user interaction according to the actual instruction can be allowed — i. e. when the user is supposed to zoom, he cannot draw brush strokes.
- **Found by participant:** 3
- **Fixed:** No.

- **Priority:** High.
- **Problem:** The user cannot recall some instructions from the tutorial—for example, he does not know he can zoom. Further, he does not recall that he needs to use two fingers to scroll instead of only one which is used for drawing.
- **Proposed solution:** Repeat the most important information from the tutorial multiple times. Also the scrolling could be redesigned to be able to scroll with only one finger as the users are used to.
- **Found by participant:** 1, 2, 3, 5
- **Fixed:** No.

- **Priority:** High.
- **Problem:** The user does not understand the label of undo button in the Czech language (it says "revert action" instead of "undo").
- **Proposed solution:** Change the label to something simpler such as "undo"[1]).
- **Found by participant:** 1, 2
- **Fixed:** No.

- **Priority:** Medium.
- **Problem:** The user has problems to understand how soft scribbles work. He knows that small scribbles can be redrawn by bigger scribbles but he does not understand that very big scribble redrawing with another big scribble sometimes does not help. This is due to the greedy nature of the LazyBrush algorithm.
- **Proposed solution:** Add the possibility to erase the brush strokes.
- **Found by participant:** 1
- **Fixed:** No.

- **Priority:** Medium.
- **Problem:** The user in the tutorial does not understand that the goal of the tutorial is not to colorize the whole image, but only to learn main important features of the application.
- **Proposed solution:** Highlight the information that the tutorial is there to teach the user basic principles of using the application.
- **Found by participant:** 3
- **Fixed:** No.

[1]) The czech label "revert action" is actually the standard system label and it can be problematic to change it.

- **Priority:** Low.
- **Problem:** When the user was choosing the color, he accidentally tapped into the picture and colorized its part. He was little bit confused.
- **Proposed solution:** Disallow user interaction with the picture while choosing color or brush size.
- **Found by participant:** 1
- **Fixed:** No.

- **Priority:** Low.
- **Problem:** When the user was asked to save the colorized image, he was searching the right button for this action for a while (10 seconds). Afterwards he recognized the button correctly.
- **Proposed solution:** Highlight the possibility to save the image.
- **Found by participant:** 4, 5
- **Fixed:** No.

- **Priority:** Low.
- **Problem:** The user tried to colorize very small rays from the sun that had variety of holes in them and the application did not colorize them at all.
- **Proposed solution:** Improve the filtration of the image so it is more sensitive to edges. This will result that the application will be able to colorize more objects.
- **Found by participant:** 6
- **Fixed:** No.

8.2.6 Colorized Images

During the usability testing, the children colorized their own images visible on Figures 8.13, 8.14, 8.15, 8.16 and 8.17. Figure 8.13 was drawn with a pen and the result is filtered with LoG and equalized with histogram linear rescale. From this, it can be seen that the LoG filter is not ideal for the colorized image output since it detects the edges and therefore the edges look unnatural. Because of this, the LoG filter is not used any-more for colorized image outputs, the image for output is filtered with the histogram equalization explained in Section 7.2. Figures 8.14, 8.15 and 8.16 were linearly rescaled between the lower and upper bound. While the upper bound removes the grey background, the lower bound makes the image contents more dark. This makes acceptable results on images with dark outlines which were captured at good lighting conditions. The big problem for this method remains images with bright edges. Although the linear rescale with the lower bound makes the edges darker, it also creates dark noisy artifacts in the background since the background intensities does not have sufficient distance from the edge intensities. Therefore, image on Figure 8.17 was linearly rescaled with the upper bound only.

Images which were drawn with a pen or a thin fix like those on Figures 8.18 and 8.19 have naturally dark edges and thus the linear rescale with the lower and upper bound gives good results. From all these presented figures, we can see that the children were capable of using the application successfully to colorize their own image.

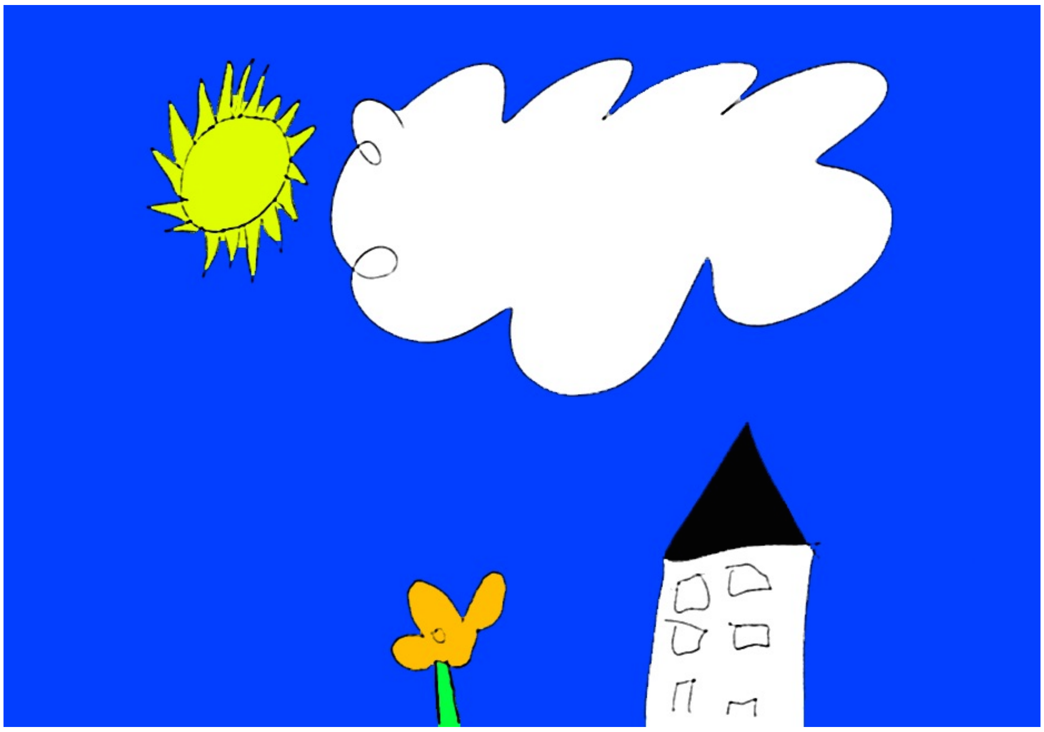


Figure 8.13. Image drawn by respondent 1

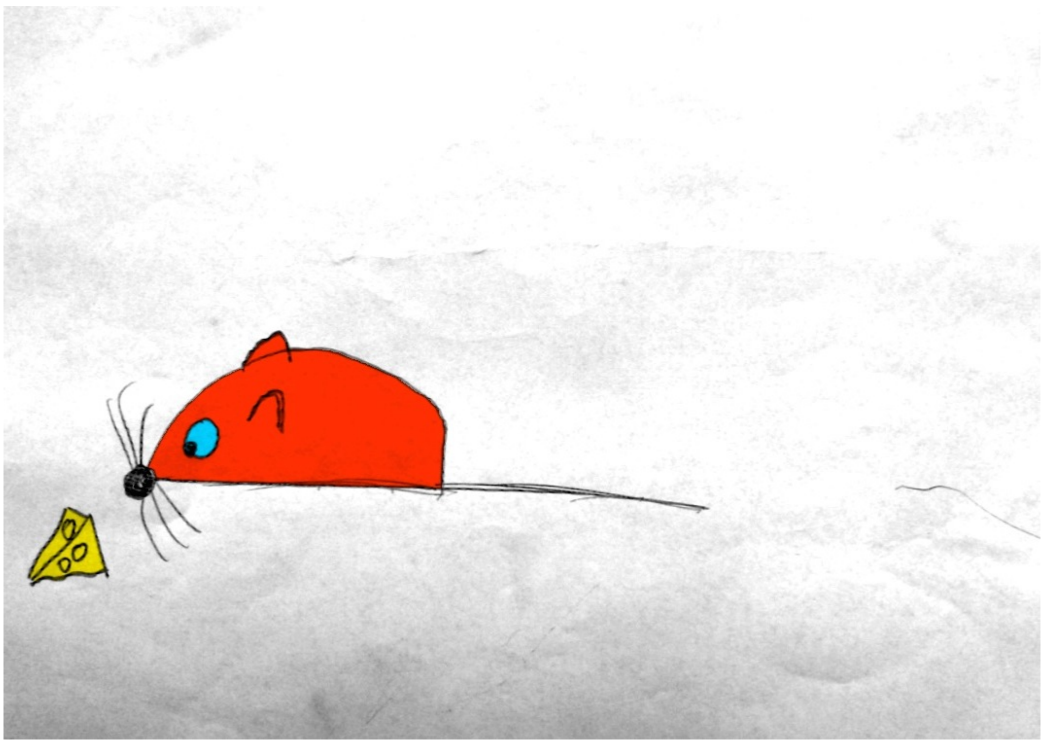


Figure 8.14. Image drawn by respondent 2

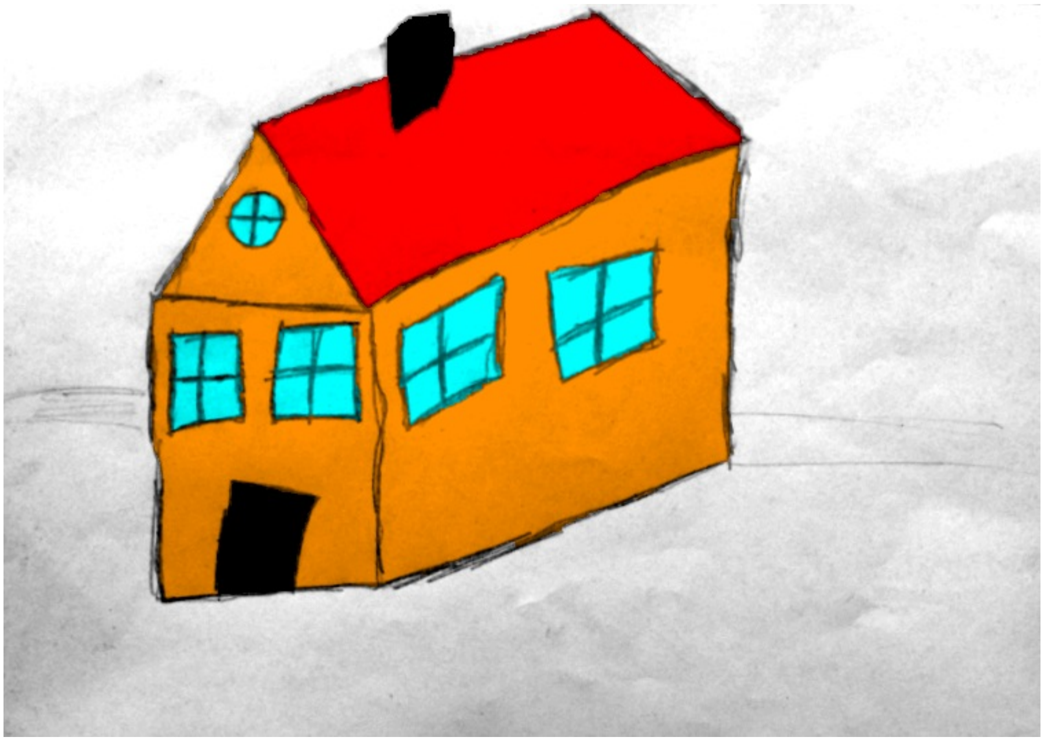


Figure 8.15. Image drawn by respondent 4

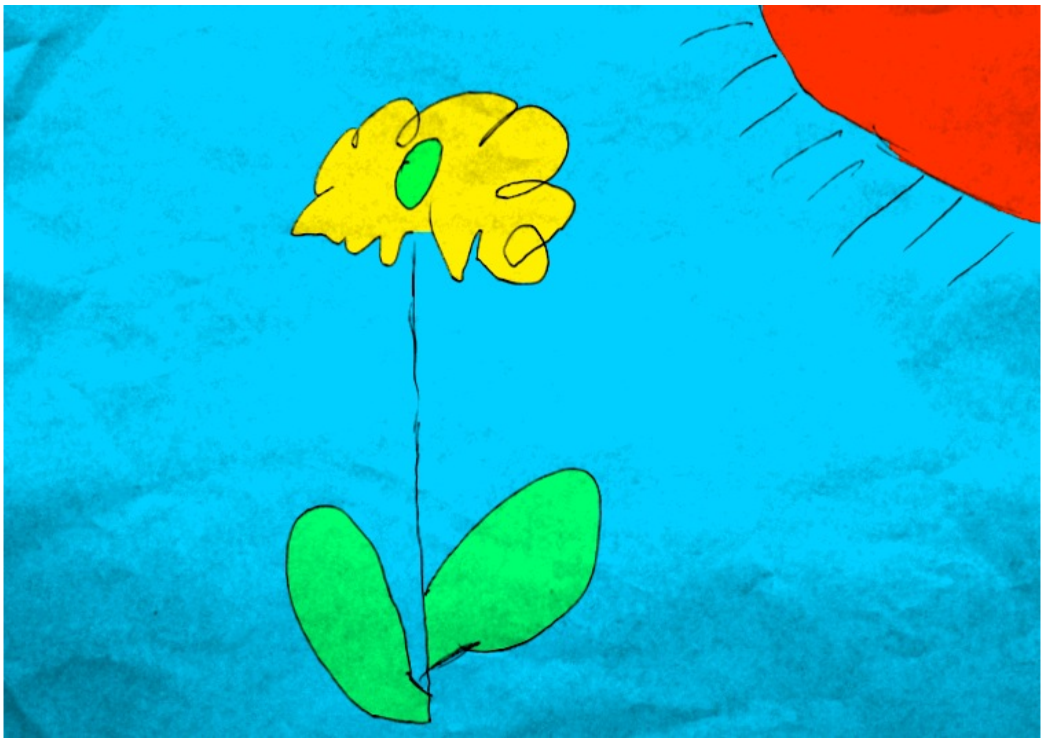


Figure 8.16. Image drawn by respondent 5

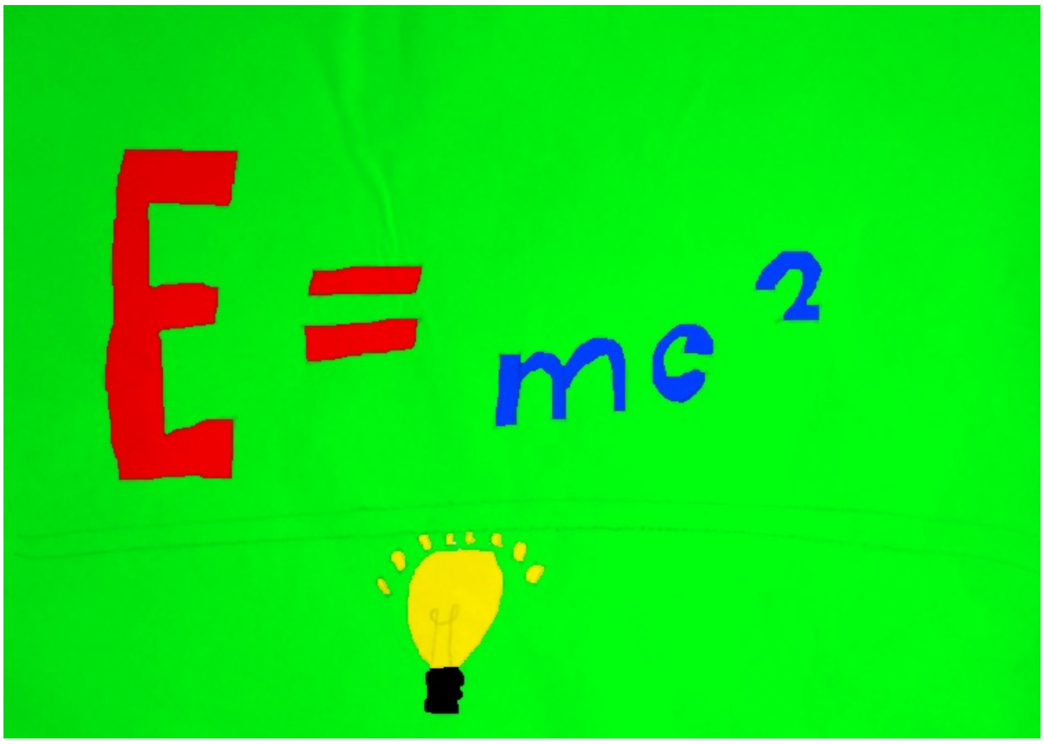


Figure 8.17. Image drawn by respondent 6

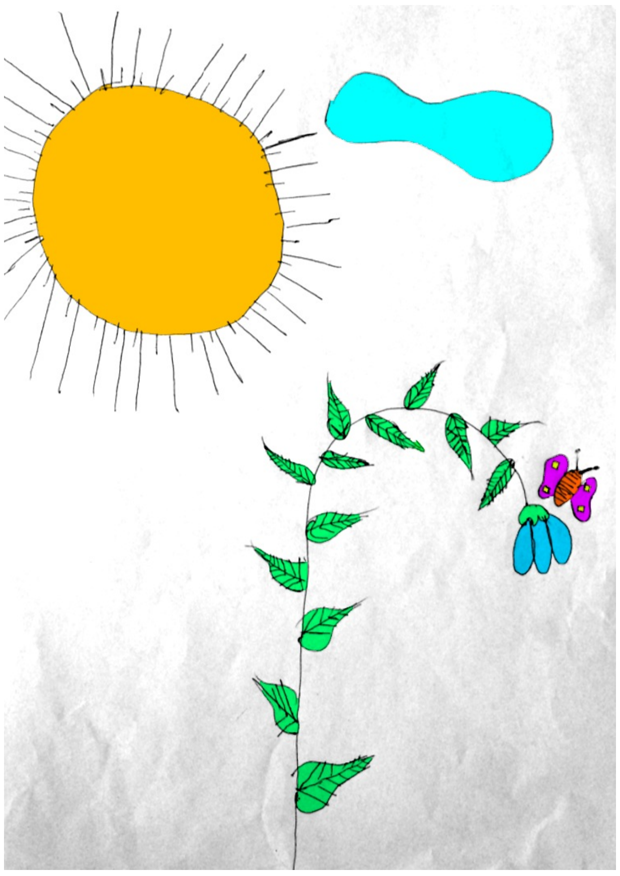


Figure 8.18. Image drawn with pen

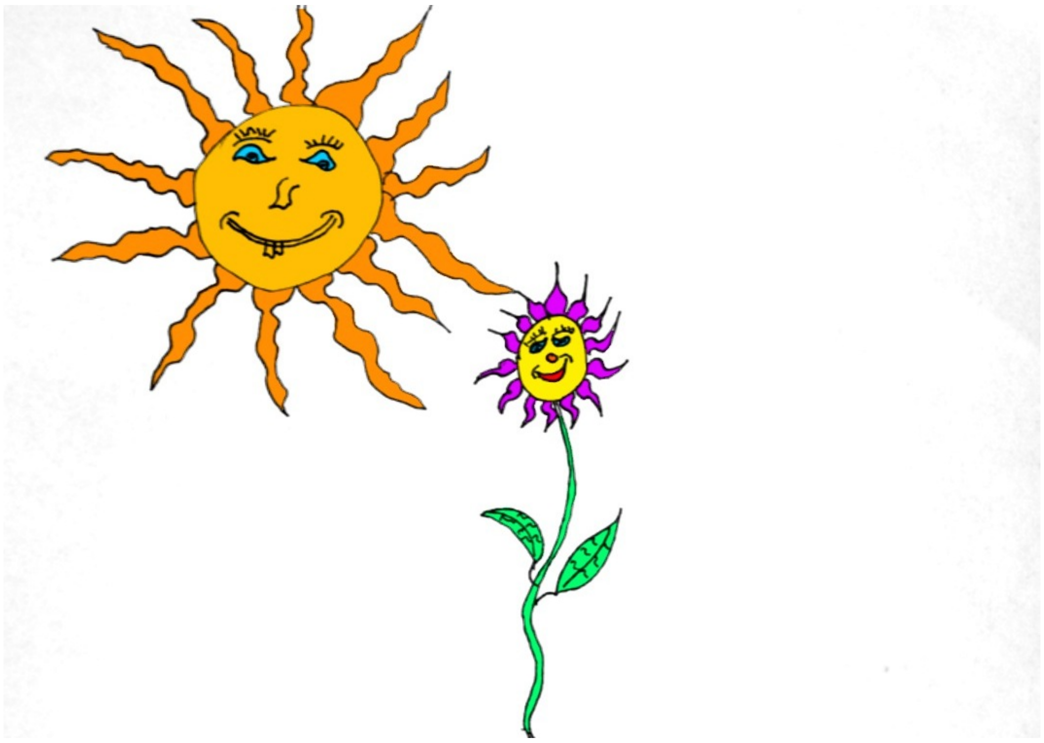


Figure 8.19. Image drawn with thin fix

8.2.7 Conclusion

The application was tested with several children from two different age groups. The test was run in the afternoon or the evening when the direct artificial lighting had to be used. This made taking photos of the pictures more complicated. Unfortunately, all the participants were boys. To obtain other relevant findings, some girls should test the application as well.

The test showed that the application is usable, but still contains some serious problems mainly in the tutorial part. While for adults there is no problem to use the tutorial, children are too quick and therefore they do not remember some important features. After the children colorized their first image they were able to use the application without serious problems. This imply that the biggest design challenge is to attract the user most for the first time he uses the application so he learnes how to use it. The biggest problem here is that the user can get bored first time he uses the application because something is not just being colored the way the user wanted.

8.3 GridCut Benchmark on iOS

For better understanding on the topic how effective the GridCut library on iOS is, performance benchmarks were done. The benchmarks can be downloaded from[1]) and were prepared by Jamriška and Sýkora. Since the benchmarks as well as the GridCut library are written in pure C++, it can be run on any platform supporting C++. From these benchmarks, the tasks that are capable of running on the iOS devices with at least 500 MB of RAM were chosen and bundled into an iOS application. The application benchmarks the GridCut library at first and then it runs the benchmark

[1]) http://www.gridcut.com/dl/Benchmark.zip

of the competitive Boykov-Kolmogorov (BK) library. GridCut is in version 1.1 and BK is in version 3.01. The compiler setting was set with -O3 and -DNDEBUG flags[1]). The results of the benchmark are shown in the graph under Figure 8.20. The original data from which this graph was generated are availabe as an attachment. On the X axis, there are individual benchmark tests. On the Y axis, there is measured time consumed by the task in milliseconds.

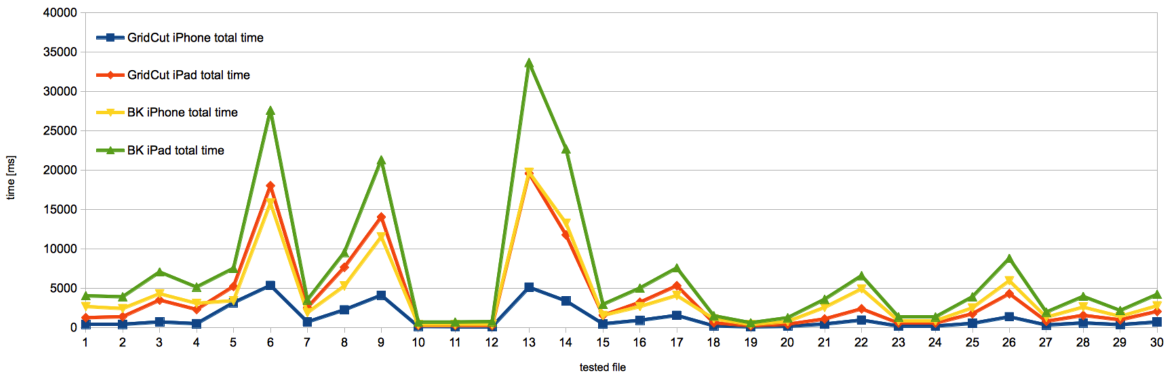


Figure 8.20. Performace comparison of GridCut and BK library on iPad 3 and iPhone 5s.

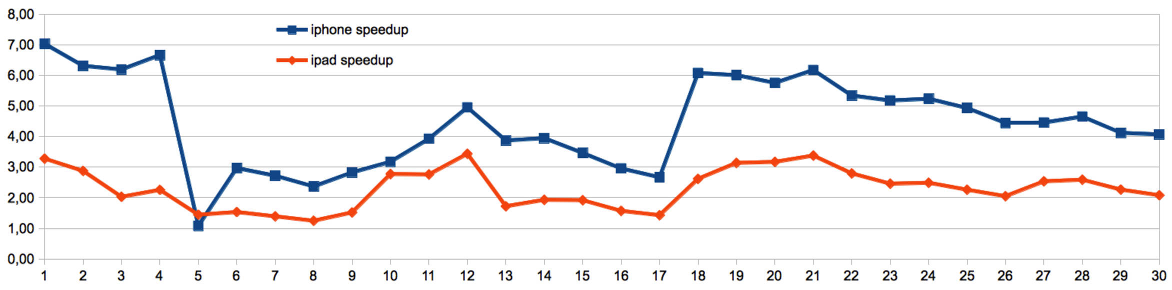


Figure 8.21. Speedup of GridCut with respect to BK on iPhone 5s and iPad 3

From the graph, it can be easily seen that GridCut runs several times faster than BK in all instances. On Figure 8.21, it can be seen that on iPhone 5s GridCut is 7 times faster in best case and 1 time faster in worst case then BK. GridCut on iPad 3 is 3.5 times faster in best case and 1.25 times faster than BK. The first four benchmark instances are the LazyBrush tasks which on iPhone 5s have the biggest speedup with a respect to the BK library.

[1]) Complete compiler settings are attached to the raw measured data in the attachments directory on CD.

Chapter 9
Conclusion

The main goal of this thesis was to implement an iOS application for colorization of hand-drawn images. Such images can be photographed or loaded from the device image gallery, preprocessed, cropped and then colorized. The target group of users for this application are mainly children, but also users that like drawing and the colorization of pictures. The colorization is implemented with the LazyBrush algorithm [14] which is one of the state of the art technique for an automatic image colorization. The critical part of the LazyBrush algorithm — the image segmentation is solved by the GridCut library [18]. The application was intended mainly for children, but the user interface is not restricted to the children users only.

9.1 Summary

The background for the application has been analyzed to be sure that such application brings a novel approach to the image colorization on a mobile platform. The colorization techniques were searched as well to show that LazyBrush is besides other algorithms highly suitable to solve our task. The LazyBrush algorithm has been explained to show all of its features and shortcomings and to show the actual process how the image is colorized inside the application.

The functional requirements on the application were formalized to demonstrate what key features are expected from the application. A qualitative user research with children in a form of an interview to provide the requirements on the user interface design was made. The key knowledge about the children behavior during image the drawing, the colorization and the usage of the current colorization application was used to design the usable user interface. This was a demanding task mainly because the children are not aware of our new approach to the image colorization with scribbles and therefore must be learned about the usage of the tool directly in the application.

The application workflow and architecture according to the requirements was designed. During implementation of the application, various problems came across and were solved. The photograph being captured directly from the application has to be cropped and filtered automatically so the colorization works as fast as possible and the colorized picture gives the pleasing result. The actual implementation of scribbles has to be also done in a nontrivial way to assure the application is smooth even on older devices like iPad 3. Nevertheless, the final application is available for iPhone only as the UI design of the iPad version has not been fully finished yet.

Final iPhone application was taken under usability testing with children. The test revealed further problems that were partially fixed in the latest version of the application. The results of the usability testing were also presented as well as colorized images from the original LazyBrush paper [14]. The iPhone version of the application is therefore prepared to be submitted to the Apple AppStore.

9.2 Future Work

In the future, the application is intended for the further development. Several issues can be improved in the future work. The filtration method for pictures taken as a colorization result gives insufficient results in case the image outlines are not dark enough. Another more robust technique to address this issue should be used.

The tutorial can be expanded to cover problems found during usability testing. Also, information about the process of how children learn could be found and used to design the tutorial better so the children can remember the learned functionalities easier.

There is also a lot of other users for whom our application can be useful — artists, architects, engineers and others. For those target groups, the user interface should be adapted so they can use all the potential the application has. Also, other advanced image processing features can be added into the application to provide a broader set of options for the most demanding users.

References

[1] Lieberman H. How to color in a coloring book. *ACM SIGGRAPH Computer Graphics*, 12(3):111–116, 1978.

[2] Smith A. R. Tint fill. *ACM SIGGRAPH Computer Graphics*, 13(2):276–283, 1979.

[3] Fishkin K. P. and Barsky B. A. A family of new algorithms for soft filling. *ACM SIGGRAPH Computer Graphics*, 18(3):235–244, 1984.

[4] Horiuchi T. Estimation of color for gray-level image by probabilistic relaxation. In *Proceedings of IEEE International Conference on Pattern Recognition*, pages 867–870, 2002.

[5] Levin A., Lischinski D., and Weiss Y. Colorization using optimization. *ACM Transactions on Graphics*, 23(3):689–694, 2004.

[6] Luan Q., Wen F., Cohen-Or D., Liang L., Xu Y.-Q., and Shum H.-Y. Natural image colorization. In *Proceedings Eurographics Symposium on Rendering*, pages 309–320, 2007.

[7] Yatziv L. and Sapiro G. Fast image and video colorization using chrominance blending. *IEEE Transactions on Image Processing*, 15(5), 2006.

[8] Sýkora D., Buriánek J., and Žára J. Colorization of black and white cartoons. *Image and Vision Computing*, 23(9):767–852, 2005.

[9] Qu Y., Wong T.-T., and Heng P.-A. Manga colorization. *ACM Transactions on Graphics*, 25(3):1214–1220, 2006.

[10] Grady L. Random walks for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28, 2006.

[11] Greig D. M., Porteous B. T., and Seheult A. H. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society. Series B (Methodological)*, 51(2):271–279, 1989.

[12] Boykov Y. and Jolly M.-P. Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images. In *Proceedings of Internation Conference on Computer Vision*, pages 105–112, 2001.

[13] Boykov Y., Veksler O., and Zabih R. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001.

[14] Sýkora D., Dingliana J., and Collins S. LazyBrush: Flexible painting tool for hand-drawn cartoons. *Computer Graphics Forum*, 28(2):599–608, 2009.

[15] Potts R. B. Some generalized order-disorder transformation. *Mathematical Proceedings of the Cambridge Philosophical Society*, 48(1):106–109, 1952.

[16] Boykov Y., Veksler O., and Zabih R. Markov random fields with efficient approximations. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 648–655, 1998.

[17] Dahlhaus E., Johnson D. S., Papadimitriou C. H., Seymour P. D., and Yannakakis M. The complexity of multiway cuts. In *Proceedings of ACM Symposium on Theory of Computing*, pages 241–251, 1992.

[18] Jamriška O., Sýkora D., and Hornung A. Cache-efficient graph cuts on structured grids. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 3673–3680, 2012.

[19] Cockburn A. *Writing Effective Use Cases.* Adison-Wesley, 2001.

[20] Stone D., Jarrett C., Woodroffe M., and Minocha S. *User Interface Design and Evaluation.* Morgan Kaufmann, 2005.

[21] Roberts L. G. Machine perception of three-dimensional solids. In *Optical and Electro-Optical Information Processing*, pages 159–197. MIT Press, 1965.

[22] Pingle K. K. Visual perception by computer. In *Automatic Interpretation and Classification of Images*, pages 277–284. Academic Press, 1969.

[23] Prewitt J. M. S. Object enhancement and extraction. In *Picture Processing and Psychophysics*, pages 75–149. Academic Press, 1970.

[24] Marr D. and Hildreth E. Theory of edge detection. In *Proceedings of the Royal Society of London*, volume 207, pages 187–217, 1980.

[25] Canny J. A computational approach to edge detection. *IEEE Transactions Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

[26] Lindeberg T. Edge detection and ridge detection with automatic scale selection. *International Journal of Computer Vision*, 30(2):117–156, 1998.

[27] Duda R. O. and Hart P. E. Use of Hough tranformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972.

[28] Ford L. R. and Fulkerson D. R. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[29] Boykov Y. and Kolmogorov V. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, 2004.

Appendix A
List of Abbreviations

BK	Boykov-Kolmogorov.
CPU	Central Processing Unit. Carries out the instructions of the program.
GPU	Graphics Processing Unit. The graphics chip that takes care of hardware accelerated onscreen or offscreen rendering.
LoG	Laplacian of Gaussian. Filter for edge detection.
RAD	Rapid Application Development. Approach to software development with the main goal to create software in short time.
UI	User interface. By this term we mean the controls that are manipulated by the user in order to control the application.

Appendix B
CD Contents

```
├── application ........................ application directory
│   ├── bin ............................ application binary
│   └── source ......................... application source code
├── images ............ selected result images in full resolution
└── text ............................... the thesis directory
    ├── pdf ......... directory with the thesis compiled to pdf
    └── tex ......... directory with sources written in plain TEX
```