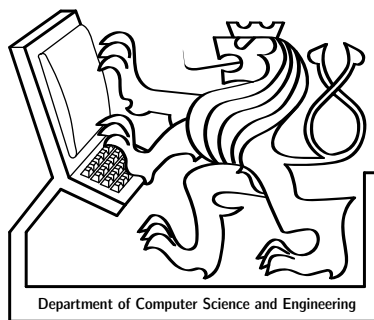


Czech Technical University in Prague  
Faculty of Electrical Engineering

# DIPLOMA THESIS

Daniel Sýkora

**Inking old black and white cartoons**



Department of Computer Science and Engineering

Supervisor: doc. Ing. Jiří Žára, CSc.

*To Zdeno...*

## Abstract

This diploma thesis discuss the problem of applying color to the old black and white cartoons produced by original step by step technology, where each animation phase was exposed on the one frame of film negative. Thanks to possibility, which allows us to convert the original analogue material to the sequence of digital images using resolution suitable for TV broadcasting, we are able to solve our problem by methods of digital image processing. We introduce several algorithms based on unsupervised image segmentation and synthesis techniques, which exploit classical properties of cartoons produced by paper or foil technology, where foreground parts are represented by homogeneous surfaces with constant grey-scale intensity enclosed by bold contours. These algorithms together with couple of prediction techniques provide us to speed up whole inking process. An important part of this thesis is also description of application, where purposed algorithms are implemented. This implementation take into account also human driven interaction which guarantee the final image quality. Described application was practically used as a part of project focussed on restoration of the old black and white cartoon: “*O loupežníku Rumcajsovi*” (directed by well known czech artist *Radek Pilař*). During this project we measured the overall application efficiency which will be also discussed in this thesis.

## Acknowledgements

I would like to say thanks to people who abetted and supported my endeavour during more than one year of hard but wonderful work. I owe much to my opponent and friend in one person: *Jan Buriánek* (always faster than light) who stays beyond the main idea and who gives me the chance to take part in this pretty interesting project. Thanks flies as a matter of course to my supervisor *Jiří Zára*, his everlasting moral support and kindness helps me a lot to accomplish this thesis.

Thanks must also be labelled to my colleagues in UPP, especially to post-production operators *Jiří Šabata* and *Helena Keslová*, to the main artist *Jiří Štamfest* who was responsible for artistic quality and also to the production manager *David Váňa*, because from creative cooperation with those people grows the final likeness of used inking technology presented in this thesis. Additionally I have to thank to chefs of *Universal Production Partners* (UPP) and *Digital Media Production* (DMP): *Vít Komrzí* and *Luboš Celar* respectively, who allow me to publish the main ideas used in this internal project and also to take me the permission to publish example images which are under rights of current owners.

I could not fail to mention users of my application and confer them the big appreciation for their patience and creativity because their hard work provides in fact success of my own work: *Ondřej Sýkora* (always the first beta-tester), *Martin Přeček* (who ever knows more tricks than I do), *Iveta Gobelová*, *Petra Bláhová*, *Tomáš Brabec*, *Zdeněk Machuta*, *Kryštof Sýkora*, *Lukáš Kobl*, *Martin Wacker*, *Štěpán Drbohlav* and others.

The last but not least acknowledgement is dedicated to my big friend *Štěpán Hrbek* who developed an alternative unsupervised background extraction technique and global luminance fluctuation suppression algorithm which was not unfortunately used in final inking technology. He also helps me with stylistic revision and overall redaction of this thesis together with *Adam Sporka* and *Pavel Zvolský*.

Work on this thesis would not be possible without devoted support of my great parents and also of my own family (*Pavla*, *Mikuláš* and *Matýsek*, the author of chapter logos). It is hard to thank for love. . .



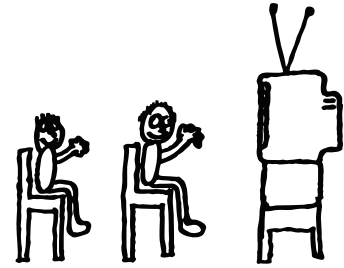
# Table of Contents

---

<b>1. Introduction</b>	<b>1</b>
1.1 Motivation	2
<b>2. Technology</b>	<b>3</b>
2.1 Color conversion	3
2.2 Inking technology	4
2.2.1 Original background	5
2.2.2 Background restoration	6
2.3 Foreground and background	6
2.4 Degradation of aged movies	7
2.4.1 Sequences of the same frames	7
2.4.2 Position instability	8
2.4.3 Luminance fluctuation	8
2.4.4 Dust spots	9
2.4.5 Band scratches	10
2.4.6 Vignetting	10
2.4.7 Contours with low contrast	11
2.5 Input and output	12
<b>3. Segmentation</b>	<b>14</b>
3.1 Previous work	15
3.1.1 Thresholding	15
3.1.2 Homogeneity	15
3.1.3 Watersheds	16
3.1.4 Edge detection	17
3.2 Contour detector	21
3.2.1 Filter design	21
3.2.2 Contour filling	23
3.2.3 Region marking	25
3.2.4 Region statistics	27
3.2.5 Foreground and background	27
3.3 Region growing	27
3.3.1 Skeletonization	28
3.3.2 Gradient seek	28
<b>4. Inking</b>	<b>30</b>
4.1 Color flooding	30
4.1.1 Modulation using RGBI model	30
4.1.2 Luminance correction	31
4.1.3 Unsupervised dust spots removal	33
4.2 Background and foreground composition	33
4.2.1 Digital matting	34
4.2.2 Contour contrast correction	35
4.2.3 Smooth color transition	35
<b>5. Application</b>	<b>36</b>
5.1 Technical specifications	36
5.1.1 System requirements	36
5.1.2 User interface	36
5.2 Layers	37
5.2.1 Layer caching techniques	37
5.2.2 Source layers (ORG, BKG and SRC)	38

5.2.3	Laplacian of Gaussian layer (LOG) .....	38
5.2.4	Contour detector layers (FIL, DEL and EDG) .....	38
5.2.5	Region marking layers (IDX and FRM) .....	38
5.2.6	Output layers (OUT and MSK) .....	39
5.3	Prediction .....	39
5.3.1	Prediction frames .....	40
5.3.2	Position prediction .....	40
5.3.3	Intensity prediction .....	41
<b>6.</b>	<b>Experiment</b> .....	<b>43</b>
6.1	Methodology .....	43
6.1.1	Measured statistics .....	43
6.1.2	Frequently used functions .....	43
6.2	Measurement .....	44
6.3	Results .....	51
<b>7.</b>	<b>Conclusion</b> .....	<b>52</b>
7.1	Future work .....	52
	<b>References</b> .....	<b>53</b>
	<b>Appendix</b> .....	<b>57</b>

# 1



---

## Introduction

*"A slow sort of country!" said the Queen. "Now, here, you see, it takes all the running you can do, to keep in the same place. If you want to get somewhere else, you must run at twice as fast as that!"*

Lewis Carroll

Digital boom on the break of last decades of 20th century proved that consume lifestyle where quantity precede quality and commercial profit knows no law is like everlasting run of *Red Queen* whom met *Alice* beyond Looking Glass.

Everybody wants to be up-to-date. Everything new have to be consumed and recycled as soon as possible. Production speed increases, technology is getting more and more complex but we are staying still on the same place because the surrounding world runs with us. Sometimes we are going back. Especially on the field that is one of the most important for our future. On the filed of animated fairy-tales for children.

What we serve to our children in the age where they are highly sensitive on creation of the value system, that will return to us after they become adults. Recent worldwide cartoon production for children predetermines, that our successors will be primitive consumers with either tyrannical evil or whimper sentimental thrash life ideology thirsted for trivial polished stories about easy available comfortable lifestyle without true artistic sense and self imagination.

If we take off from the running train and look backwards to our history we understand how huge is the change that we go through. Especially in history of Czech and Slovak cartoon making production we could found really valuable and artistically advanced works, which stay still in front of worldwide competition and which are invaluable source of inspiration for children's imaginative world.

Anyhow controversy was socialist history of Czechoslovakia we have to confess, that it provided nearly ideal working conditions for talented cartoon making artists like *Jiří Trnka*, *Zdeněk Smetana*, *Karel Zeman*, *Zdeněk Miler*, *Jiří Šalamoun*, *Radek Pilař*, et cetera. They proved rule that really valuable work could arise only from artistic motivation and never from commercial profit that stands to usual motivation of market oriented production.

Unfortunately these works are aged movies. They are stored in wet depositories on negative film material under imminence of putrefaction. If we want to rescue them for our children we have to abort progressive destruction. We have to clean them from mildew or other impurities and convert them to the digital form. And more if we consider that aged movies are stored on black and white material due to former TV broadcast facilities, we have also chance to enrich them by new color information. Children are much more sensitive on color motley than adults. If we apply sensitively proper bright and dark colors in to the yet designed artificial black and white world we are able to increase specific artistic impression which is well perceived by children's mind. In short we have big chance to add a new artistic value.

This is uneasy and expensive task. We need someone who is able to provide big investment to such project without expectation of fast overheads recovery and also someone who is capable to make sensitive transfer of colors to black and white material without losing original cartoon style, the best if it could be directly the original author of such cartoon.

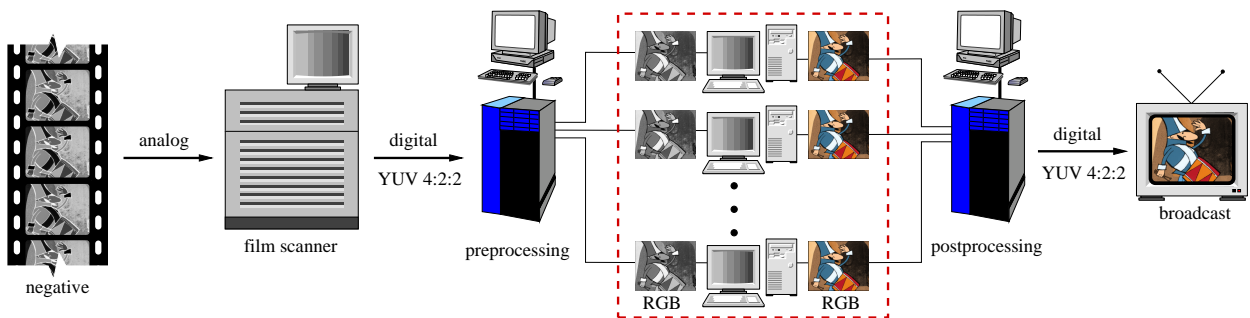
## 1.1 Motivation

This diploma thesis is a part of bigger project that came out from co-operation of three subjects: *Czech TV* (CTV), *Universal Production Partners* (UPP) and *Digital Media Production* (DMP). The main task was to rescue great aged black and white cartoon “*O loupežníku Rumcajsovi*” directed by unfortunately yet deceased but still undead guru of czech cartoon school *Radek Pilař*.

If we set apart fact that we need to convert aged negative film material to digital form and only focus on inking process itself, we at first assume that only well experienced artist supported by expensive hi-end post-production software running on the dedicated film editing hardware could satisfy needed artistic quality. But considering the fact that such systems are not well equipped by tools for simplified transfer of color information in to the grey-scale images, artist would be endow on lots of the featureless and repetitive work which will disturb him from real artwork. Moreover we have to expect also huge outgoings, while film editing system have usually really expensive machine time. It is clear that this technology is going to be unacceptable, because we could not expect fast overhead recovery. We need to speed up whole process and make it cheaper.

The main motivation of this thesis was to automatize routine work using fast semi-automatic computer vision algorithms implemented inside PC application for cost effective computer workstations. This proprietary application allows us to perform parallelism of routine work by cheaper computers (see *Figure 1.1* in the red dashed rectangle). Main film editing hardware will be responsible only for necessary preprocessing and postprocessing tasks and will distribute image sequences through fast network to these PC workstations.

We assume that proposed computer vision algorithms could not be obviously so powerful to prepare all frames in clear form. There are usually lots of small visible errors which have to be corrected by human driven intervention with no heavy artistic knowledge. Therefore we also need to lay down suitable, fast to operate and easy to learn user interface which allows not so experienced operator to make correction work effectively.



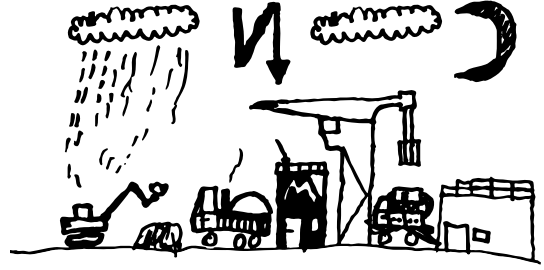
*Figure 1.1:* Work distribution using parallel working cost effective workstations.

While we take into account the human intervention, it is also reasonable to preserve interactive performance of our application to avoid operators from idle time during correction work. Built-in algorithms must not compute one frame longer than one second. It would be uneasy to reach this short processing time limit using recent PC architectures because of expected TV broadcasting resolution of input images. Purposed semi-automatic inking technology based on image segmentation in higher resolutions is time consuming. We will see later that fast implementation of several algorithms should not be possible without couple of advanced optimization techniques also described in this thesis.

Although purposed application was developed and employed only for cartoon “*O loupežníku Rumcajsovi*” and also all example images in this thesis came from this piece of *Pilař*’s work, purposed principles are general hence they could be applied on cartoons with similar artistic properties. In other words, this thesis solve general problem: We have sequence of grey-scale images with features like bold dark contours and intensity homogeneous regions, and we want to apply proper color information as effectively as is possible.

# 2

## Technology



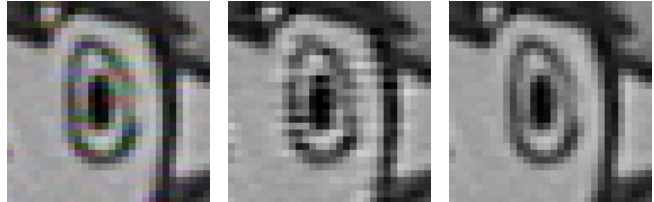
First we summarize what kind of data we have on the input and which results are expected on the output of our application. We will explain in particular several suitable inking technologies and required post-processing and pre-processing phases. Also detailed description of main movie degradation artifacts which increase difficulty of inking process will be included. This chapter would be alternatively titled as: What we have to do and discuss before we will be able to produce high-quality colored cartoons from degraded grey-scale material.

### 2.1 Color conversion

Usual cartoon series consists of several episodes. Each episode is approximately 8 minutes long. Considering a constant frame rate of 25 frames per second, we expect approximate 12000 frames per one episode. Each frame is usually scanned and stored as  $YCrCb$  image using 4:2:2 sampling scheme with PAL resolution (720x576) (see [Pank98] for details). Although it would be much better to work with original 2K film resolution (2200x1640) to avoid undersampling of details, the motivation was to move inking process from the hi-end post-production hardware to the standard PC platform. On the PC platform, the PAL resolution is acceptable for both amount of data storage and for enough processing speed.  $YCrCb$  image is finally exported for PC purpose into the RGB bitmap stored in uncompressed 24-bit TIFF, which is more than 1.2MB of the disk space.

The first stage of the pre-processing phase is the conversion from 24-bit true-color format to the 8-bit grey-scale image, which will be stored in a lossless compressed grey-scale image (in PiNG format) afterwards. This automatic process significantly decreases the size of input data (20% of the original size) without loss of any important information.

A proper conversion is not trivial since the color components of pixels, representing under-sampled high-frequency details (e.g. thin contours), are far from similar value of corresponding luminance due to low bitrate of color components in 4:2:2  $YCrCb$  color. This is the first sampling problem caused by PAL resolution and ill posed color conversion. For detailed insight into this problem see the magnified rectangles taken from selected movie frame on *Figure 2.1*.



*Figure 2.1: RGB to grey-scale conversion: source RGB rectangle (left), red, green and blue average (middle), correct weighted average (right).*

We could not simply compute average of red, green and blue component. If we did so, we would have destroyed the original contour's anti-aliasing. Instead, we should use conversion equation that preserves human eye perception capability of red, green and blue component of the final color:

$$I = 0.3R + 0.59G + 0.11B + 0.001 \quad (2.1)$$

Conversion (2.1) produces image that is very similar to the direct conversion from  $YCrCb$  color without intermediate RGB conversion. The same equation is used for general conversion of RGB color image to corresponding grey-scale output. The reason why we emphasize this here was to show that we should take care even in quasi trivial problem to keep maximum information about original analog image.

## 2.2 Inking technology

After color conversion we have grey-scale frames ready for following inking process. Therefore it is time to choose suitable inking technology. Transferring color to grey-scale image is heavy underconstrained problem. There are several approaches which could be used for this task but none of them could be full automatic. Human intervention is always necessary.

First we have to premise that probably lots of work has been done on unsupervised inking of aged black and white movies. If we notify that world of classical handmade cartoons has many commercial aspects due to expected profit based on high broadcast costs we could not hope in opened research projects focused on this problem. So if we search for previous work on semi-automatic inking we will met with a big disappointment. So we should unfortunately describe only two previous methods.

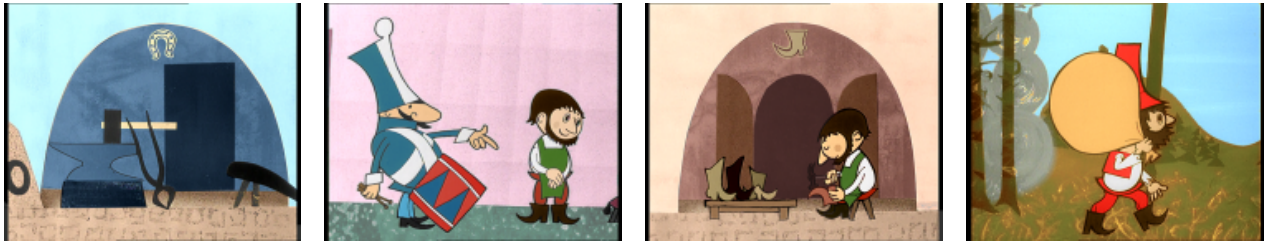


Figure 2.2: Example of the output provided by luminance keys driven colorizing technology.

In well known *Gonzalez's* digital image processing book [Gonzalez87] we could find basic inking technique that is also widely used in hi-end movie post-production software as generic method for transferring color into grey-scale images. Selected luminance values are converted using user defined look-up table to wanted hue, saturation and brightness. Smooth selections of input luminance values are known as *luminance keys*. They could be used simultaneous on several regions with different luminance median and almost disjoint deviation interval. Since the different luminance median introduces really hard constraint for most of cases (e.g. background has nearly similar intensity as character skin, etc.) we could not use this method without other additional tools (e.g. segmentation masks or animated shapes) which unfortunately provide significant slowdown of inking process as was proved during testing phase on cartoon “*O loupežníku Rumcajsovi*” (see Figure 2.2).

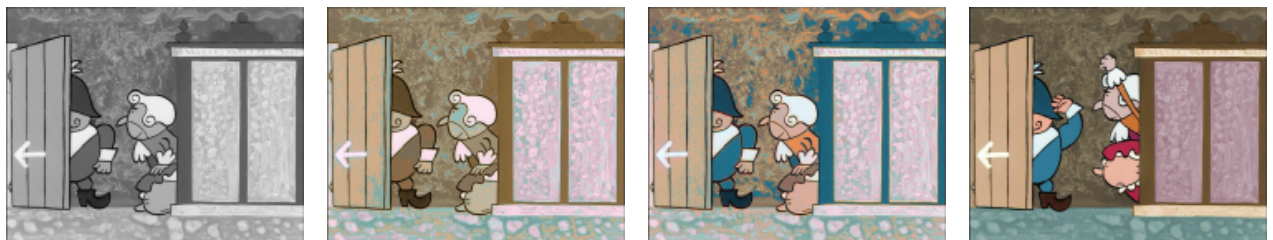


Figure 2.3: Example of scene where Welsh's algorithm failed: (from left to right) target grey-scale image, unsupervised color transfer, color transfer with user defined swatches and source color image as example.

Another approach due to *Welsh et al.* [Welsh02] take advantage of *textural information*. Color transfer between already inked source and grey-scale target is based on local luminance distribution matching in  $l\alpha\beta$  color space [Ruderman98] and [Reinhard01]. This technique is inspired by *Hertzmann's* image analogies framework [Hertzmann01]. Jitter sampling is used to select subset of representative pixels in color image. It is also possible to choose these samples manually as rectangular swatches in both images to reach better matching results. This technique is surprisingly successful in natural scenes (e.g. tree on meadow with sky on horizon, deep forest with brown trunks and green leaves, etc.). But cartoons like “*O loupežníku Rumcajsovi*” have not enough textural information. Lots of frames only consist of almost plain regions vary above all in global intensity and thus this simple process will fail (see Figure 2.3).

As we saw that existing methods are not suitable for our case. We should start from a scratch and develop new usable inking framework. If we exploit the fact that whole cartoon was mainly done by *foil technique* we could introduce main idea of our inking technology. We divide input frame to *background* and

*foreground* layer and apply ink on them separately. After this procedure we create final coloured composite from both layers.

There are two possible inking approaches based on this main idea. They have very different properties and thus we have to compare their fiscal and temporal aspects to decide which technology is more suitable for our purpose.



Figure 2.4: Comparison of two different background inking technologies: natural foreground composite with original background (left), synthetic foreground composite with restored background (right).

Both methods used the same approach for applying color on the foreground parts and vary only in background inking. Foreground inking process will be described in detail later. In short it consists of contour detection, area segmentation, color indexation with prediction and final composition with the restored or original background. Exclusion of two words *restored* and *original* from last sentence introduces main difference of presented methods.

### 2.2.1 Original background

The original background approach preserves the unchanged background in every animation frame. It uses only the color information from the restored background and then applies it on a grey-scale frame. The foreground elements and the original background bitmap are merged in an ingenious image. As described, the method seems to be simple, however it proves very efficient (see Figure 2.4). Nevertheless there are several circumstances which introduce drawbacks of this technique. Reducing the inking process only to adding hue and saturation on existing brightness could restrict the artist's freedom of color selection. If we allow to change the brightness we introduce some intensity aliasing problem on area boundaries. Moreover we have to perform exact motion tracking at sub-pixel level in order to hide visible differences between original and restored background (see Figure 2.5), and also to ensure frame-by-frame removal of the degradation artifacts of aged movie.

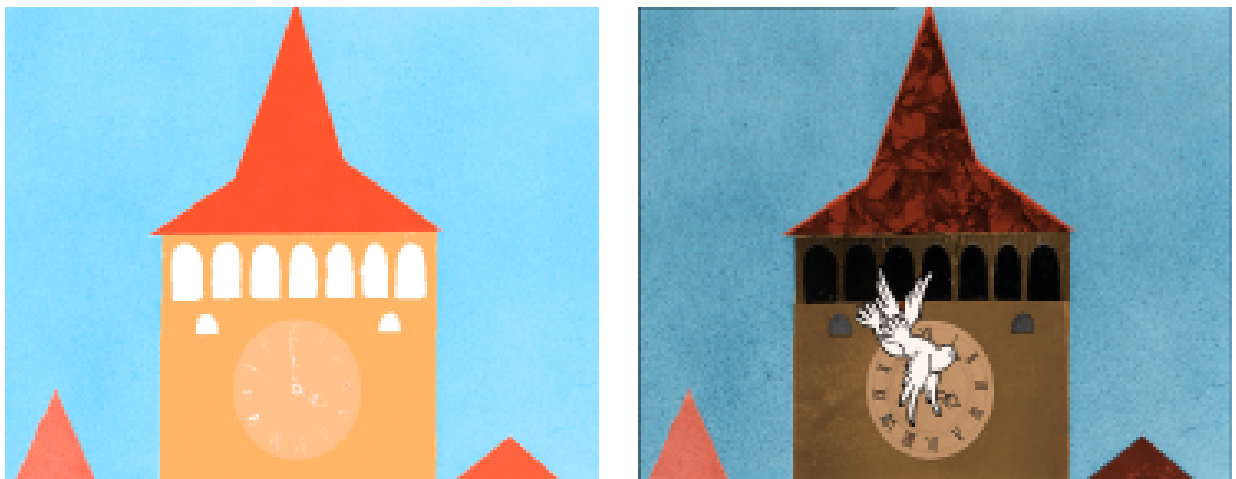


Figure 2.5: Draw backs of original background inking technology: hue and saturation plane extracted from restored background (left), grey-scale image inked by hue and saturation (right).



### 2.2.2 Background restoration

Technology based on background restoration replaces the original background by its new polished and inked version. The original foreground will form synthetic composition with the new background. This technique is also known as natural digital matting and in most cases it could not be solved exactly, especially in grey-scale case (see [Chuang02]). Therefore we sometimes observe the synthetic look of contour's outer edges (for comparison see *Figure 2.4*). But this main disadvantage is lessened by the following advantages: artist, who is responsible for background restoration, is less constrained for he/she has full creative freedom, the new background could contain completely different texture details, we need only motion tracker with pixel precision (available on PC platform) and because of the restored background is almost new drawing then the side effect of this inking technique is that all aged motion picture degradations which resides in background area on original source material are fairly removed.

If we want to preserve exactly the same cartoon appearance including all motion picture degradations, we use the *original background* technique. On the other hand, if we prefer less expensive and not so time-demanding solutions without a strict requirement on the original cartoon style, we choose *background restoration* method. According the basic motivation of this thesis, we conclude to select this approach as a proper inking technology.



*Figure 2.6:* Restored and inked background.

## 2.3 Foreground and background

For our inking process, it is really important to discuss which parts of the input image will be understood as background and foreground respectively. We exploit the significant features of typical *Radek Pilař's* cartoon style. Foreground parts are defined as shapes with luminance homogeneous area enclosed by bold contour that changes its shape, size and position (see exception that proves the rule on *Figure 2.7*). Rest of the static areas without bold contours are defined as background. This assumption also means that some foreground parts could be occluded by background area layer.



*Figure 2.7:* Foreground area (tree) without contour.



*Figure 2.8:* Example of automatic background movement tracking from the set of frames in the sequence where camera shifts horizontally. Vertical black lines indicate relative position of the viewport rectangle.

This division is very important because of the different inking techniques applied on the background and foreground parts of the image. The background is manually reconstructed from selected frames by experienced artist using a standard image manipulation software on PC platform. Usually this reconstruction is performed with exploitation of one or more frames from the whole sequence that covers as much background area as possible (see *Figure 2.8*). This reconstructed plane will be manually polished and colored by experienced artist (see *Figure 2.6*) and used as the input background layer for foreground semi-automatic inking process.



The background is usually static (relative to camera view on the scene) but sometimes the camera changes its position and scale. In this case we should track the whole sequence to extract all visible parts of the background. We also need an information about the camera motion, since we have to know which rectangular part from the whole background plane will appear in the corresponding animation frame. This process could be done almost automatically on PC using existing film post-production software with sub-pixel accuracy (see *Figure 2.8*). However sometimes, a human intervention is necessary because of possible texture-free background, non-linear image distortion due to film degradation, or in such cases, when the static foreground part of the image is very large and automatic tracking will completely fail or produce unacceptable errors.

## 2.4 Degradation of aged movies

Our source material is aged handmade movie, therefore there are some unwanted degradation artifacts that we have to be able to identify and then remove. We are not going to make detailed description of all the possible degradations in aged image sequences and their removal techniques. Better resource for further study would be for example the great *Kokaram's* book [Kokaram98] or dissertation [Kokaram93].

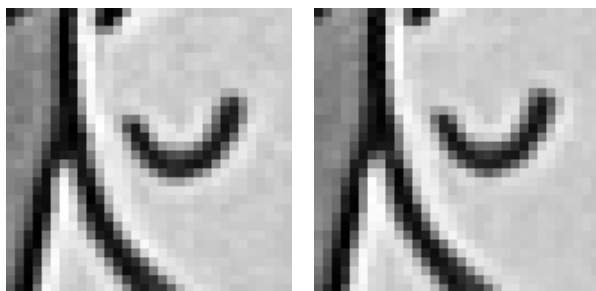
First, we have to identify the originator of the the artifacts and then we decide, how to remove the artifacts without loss of the original artistic impression. Particularly, it is very difficult to decide whether the artifact is or is not unwanted. This is not an easy task and it depends on the human artistic sense and preferences.

The example of such problem is the position and scale stability of the static views. Small frame-by-frame shifts of background and unwanted shifts of foreground relative to static background have to be stabilized. The first case is caused by an inexact camera focus and position stabilizer combined with the film position instability during scanning process, so we may consider it as a removable artifact. However, the second case (generally much more visible) is human inaccuracy in positioning moving foreground parts on the static background. Here we must decide whether or not we want to remove this feature.

One approach is to preserve the low technical quality of movie and only to colorize the raw grey-scalers without any polishing or reconstruction (suitable for *original background* technology). This opinion is more artistically driven than another approach, which prefers high-quality output comparable with latest *Walt Disney's* computer aided cartoons. An optimal point of this trade-off is to be found.

A usual reasonable decision is to preserve as much of the original look as possible and remove only physically caused artifacts raised by the natural or mechanic film deterioration, including the errors imported into source material during scanning phase. At this point, we list some of these artifacts with suggestion how to deal with them (*background restoration* technique is considered):

### 2.4.1 Sequences of the same frames



*Figure 2.9:* Differences between doubled frames.

Especially different noise texture in each frame is visually perceived as warm flavour of the real-world computer unassisted animation. If we want to preserve this feature, we have to apply ink on the each frame separately. In our case, this is relevant only for the foreground part of output frame. Unfortunately the selected *background restoration* technique do not preserve this feature on the background layer, so we should simulate it by automatic noise superposition during finalization phase on hi-end post-production platform. We exploit foreground masks to avoid adding this synthetic noise to regions where original foreground is located.

### 2.4.2 Position instability

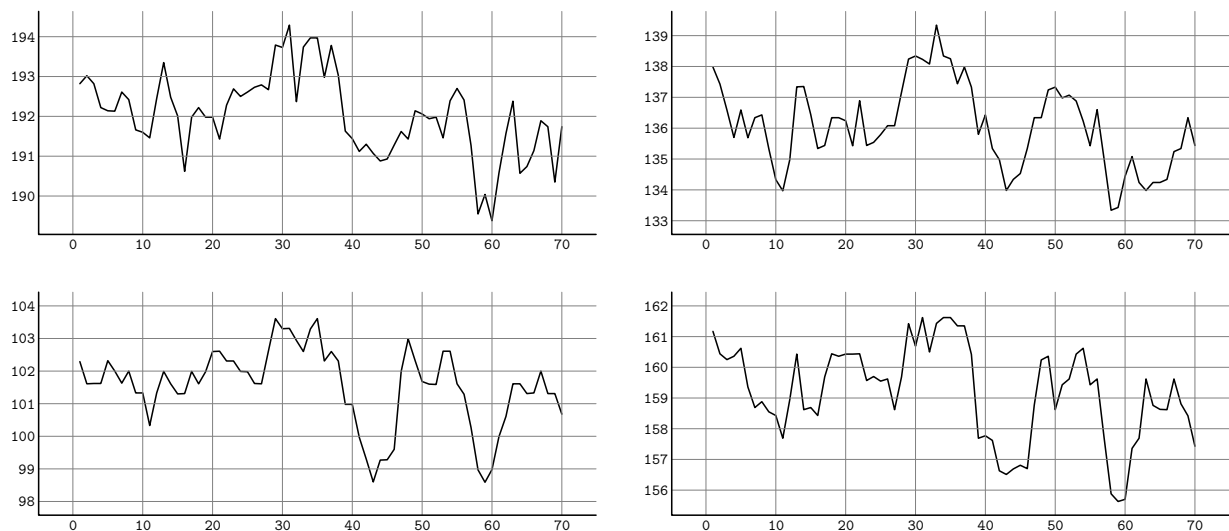
The position instability was already discussed in the introduction of this section. It is reasonable to stabilize only the global shifts of each frame and the local foreground. (And to leave the background movement alone.) If we consider that visible shifts are present even on the sub-pixel resolution, then the stabilization task moves to the nontrivial problem. It could be partially solved by one of two known motion estimation techniques: the first, based on block matching [Liu93] with higher computational costs, and the second, much faster but not as exact, gradient based approach [Martinez86] or [Driessen91]. Both methods are already implemented in hi-end post-production tools but they are not precise enough for our needs. We exploit this motion tracking inaccuracy in phase when we are generating frames from restored background. Since the motion tracking process is not exact, the unwanted sub-pixel shifts will be destroyed by a smooth approximation of motion curve. Disadvantage of this perfect stabilization is the increasing foreground instability.

### 2.4.3 Luminance fluctuation

Luminance fluctuation or *flashing* is visually very important artifact. Human eye is very sensitive to small luminance fluctuations, especially if they are only local. Let's compare four temporal graphs on *Figure 2.11* where the median fluctuation of the selected homogeneous areas is apparent. Standard deviation is not bigger than 1% of intensity domain, however the human eye still can observe the luminance fluctuation of even lower deviations. We could also find out the correlation between graphs which indicates the global intensity shifts. To understand why the global luminance is not constant, we have to know that the production of cartoons is a time consuming process and the lamps used for the illumination of the animator's board do not have constant temporal luminance over their short lifetime. Also, we have to consider the possible charge drops or inexact objective blind setup and other similar influences. See *Figure 2.12* for example of really significant luminance fluctuation.



*Figure 2.10:* The scene where luminance fluctuation statistics were measured.



*Figure 2.11:* Temporal graphs which demonstrate median fluctuation of the selected homogeneous areas: cobbler's skin (upper left), table with boots (upper right), stool under cobbler (bottom left), cobbler's apron (bottom right). See also *Figure 2.10* to locate these areas.

This artifact is very important because it makes difficult to apply trivial semi-automatic inking process based on *intensity keys* (as we discuss in *Section 2.2*). Unsupervised removal of global luminance fluctuation is not trivial task and was partially solved in [Hrbek02] while the accuracy of *Hrbek's* method depends upon correct motion estimation.

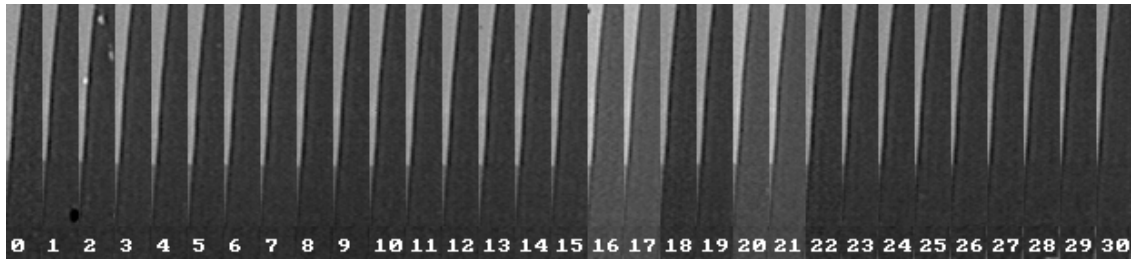


Figure 2.12: Strong impulsive luminance fluctuation: temporal history of selected rectangle on static background entitled by frame numbers.

But removal of this artifact (for background case) is trivial when we use *background restoration* inking technology (presented in Section 2.2.2). Constant luminance of the foreground areas is assured by couple of stabilization techniques that we describe in detail in Section 4.1.2. They preserve temporally constant user defined luminance median (see Figure 2.13 on the left side) for (medium and large sized) homogeneous areas without losing original area features (i.e. natural texture noise, edge sharpness, etc.).

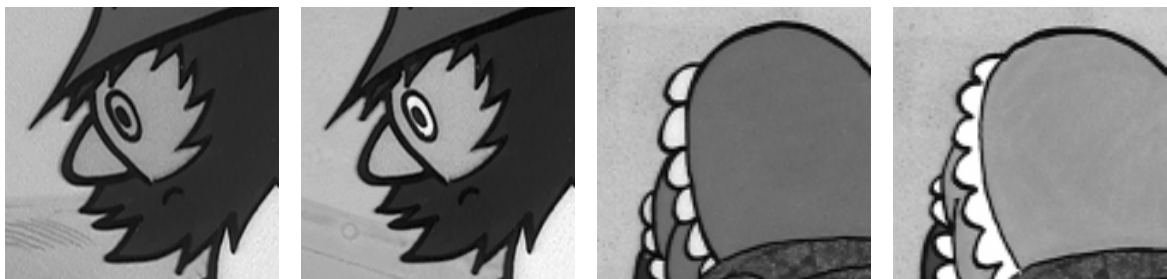


Figure 2.13: User defined luminance median (left) and luminance fluctuation effect (right).

Unfortunately there are also couple of sequences where we could find special effect based on local or global luminance increasing or decreasing, for example cuts with global fade-in/outs and local effects like shame skin or stomach nausea (see Figure 2.13 on the right side). Because of minority of these effects they are solved manually during final post-production phase on hi-end platform.

#### 2.4.4 Dust spots

Dust spots, sometimes called *blotches* or *dirt and sparkle* are well known impulsive distortions of aged movies. They have strong edges and their average luminance tends to clean black or white. They could be found at the specific position on one frame only. Black dust spots and short hairs reside on the original negative surface and after exposure process they are reprojected to the spots with white luminance. In our case the negative copy of the cartoon was passed through film scanner and after that digitally converted to the positive without exposure process. So white spots have higher frequency than black spots because black spots reside above all on the positive material. To see various types of impulsive distortions see Figure 2.14.

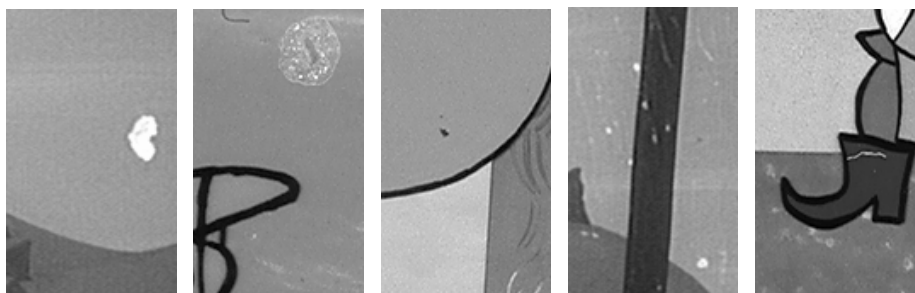


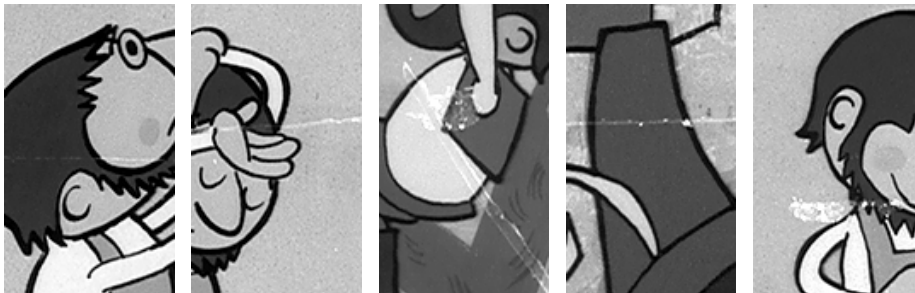
Figure 2.14: Various types of impulsive distortions. In the middle we could see rare example of the black spot.

Lots of work has been done on impulsive distortion suppression. One-shot detection and removal methods based on global multilevel median filtering [Nieminen87] or standalone detection using model of local

interaction of neighbour pixels [Kokaram93] or grey-scale morphological filters [Joyeux01] followed by reconstruction of degraded area with B-spline interpolation [Lee83], Markov random fields [Li96], autoregressive model [Kalra97], Fourier transformation [Joyeux01] or with advanced hole fitting algorithm using fast texture synthesis by patch-based sampling [Liang01] were presented. But inking process based on background reconstruction provides straightforward removal of these artifacts in background case (see *Section 2.2.2*). Foreground case is solved by simple and effective black or white distortions removal algorithm that exploits area homogeneity feature (see *Section 4.1.3*).

### 2.4.5 Band scratches

Band scratches are very similar to impulsive distortions and have same photometric characteristics but they are much more aggressive and produce large image degradation. Frames with the band scratches are rare because they are mostly located at sticking frames where cuts of original film was glued together by transparent sticky tape. They could be located and removed manually in pre-processing phase (see *Figure 2.15*).



*Figure 2.15:* Several examples of band scratches.

Another common type of deterioration are vertical line scratches raised by repetitive positive film rubbing through mechanical parts of film projector. If we have original negative material of the cartoon movie which have never met film projector then the scanned material is also void of this type of scratches. Anyway this is also well studied problem. Example of the efficient solution should consist of detection by line tracking algorithm using Kalman filter on vertically sub-sampled images of frames and Fourier series removal preserving high-frequency information as was presented in [Joyeux01].



*Figure 2.16:* Examples of manually painted effects: bee's cluster (left), motion blur (middle) and smoke (right).

Another artistic effects similar to what we see in luminance fluctuation section are manually painted motion blurs or smoking effects (see *Figure 2.16*). They symbolize very fast movement or foggy atmosphere of the scene. Because of similarity with band scratches they are classified as movie degradation artifacts and should be reconstructed by experienced artist in final post-production phase.

### 2.4.6 Vignetting.

Vignetting. Camera objective is a system of disperse and joint lenses which transform perpendicular rays from animator's board to confluent rays and produce smaller copy of the projected scene on the negative film frame. But this conversion is not perfect due to known lens defects so we observe the vignetting artifact on the boundary of enlightened area (see *Figure 2.17*). Boundaries are relative brighter or darker than spare

inner area of the frame. This problem is finally in practice very similar to luminance fluctuation because of same inconveniences connected with *intensity keys* pseudocoloring process.

It is easy to remove vignetting when we could perform camera calibration. We can simply measure vignetting effect by shooting plain solid plane with known intensity. Digital picture of this plane could be used as multiplicative correction mask for vignetting artifact (see [Hlaváč94]). Unfortunately this is not our case. But we could remove it in the same way as luminance fluctuation artifact by inking technology based on background restoration and by user defined foreground area luminance median stabilization algorithm.

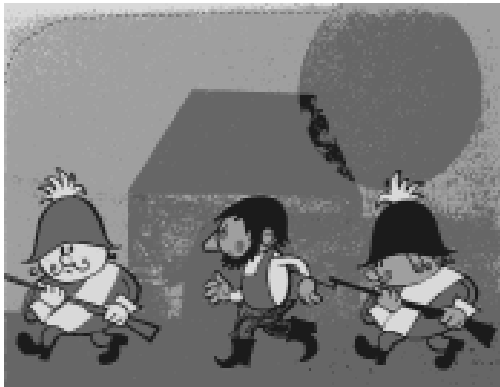


Figure 2.17: Vignetting visualized by color quantization.

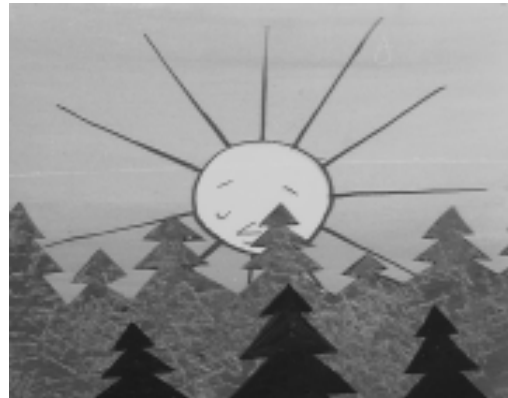


Figure 2.18: Bright sun needs original contour brightness.

## 2.4.7 Contours with low contrast

Contours with low contrast are mostly produced in three independent cases. First case is artist's purpose (see Figure 2.18). Bright contour visually propagates higher luminance of the enclosed area. Second case is due to vignetting artifact. Foreground contours near the boundary of the frame are brighter than contours which are more inside the frame. Last case comes on when all contours have histogram peak in intensity band where non-contour parts of the image usually reside (see histogram on Figure 2.19). This happened due to scanning process and sometimes due to film overlighting during cartoon shooting. Experimental setup of conversion curve which projects high-resolution multispectral intensity to 8-bit range during the film scanning process is usually done one-time and during the whole part remains static. This non-adaptive framework do not respect temporal global luminance fluctuations thus several cuts may be overlighted. We can see that low-contrasted contours problem is subset of previous mentioned luminance artifacts.

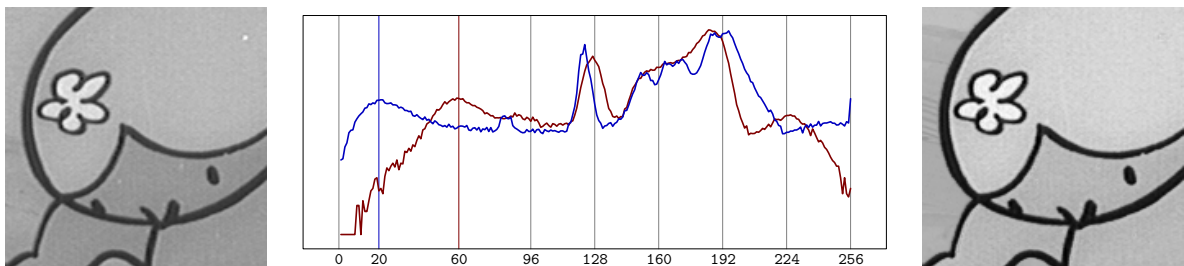


Figure 2.19: Low-contrasted contours (left, red histogram), histograms with marked contour's peaks (middle), Well-contrasted contours reconstructed by local gamma correction (right, blue histogram).

First case was solved by algorithm based on smooth directional interpolation between background and foreground colors exploiting ideas based on distant field technique (see Section 4.2.3). Other types of contours with low contrast are slid off to the black by local histogram equalization smoothly applied only at contour location with user defined gamma correction (see Section 4.2.2 for details). Local equalization is destructive process that imports new gaussian smoothing to the original image (see in Figure 2.19 on the right side). If we want to preserve original cartoon look without adding any pixel polishing it will be reasonable to use this algorithm only for really badly scanned or overlighted sequences.

## 2.5 Input and output

In this section we summarize input and output of application that provide semi-automatic foreground inking process (we call it COL). This will also contain rough list of pre-processing and post-processing operations which have to be done by external applications on hi-end and low-end platforms.

First we have to find real movie cuts that should be separated into stand-alone sequences of frames. They will pass through inking process separately. This pre-processing phase is human driven. Hi-end film post-production system operator who is responsible for exporting these cuts for low-end platform gathers also lots of additional information about each sequence that will be used during inking process:

- **Frames for background restoration.** This will help experienced artist in background restoration phase. He perform background reconstruction only from these selected frames. This information is not necessary if we perform semi-automatic extraction of whole background plane by camera movement tracking.

- **Type of background movement:** static, zooming, panning, inter-fading, fade-in/out. This will decide if we generate only one static frame of background or perform camera movement tracking additionally connected with alpha-blending of two background planes.

- **First, last and start frame containing foreground parts.** These will decide on which frame interval of current movie cut we apply foreground inking and where we should start with this process. Start frame could be same as fist frame but in scenes where due to camera movement all foreground parts are not visible on first frame is seasonable to start in frame where all foreground parts are visible. This save lots of correction work during foreground inking phase as we will see in *Section 5.3*.

- **Frames for foreground color sample.** Needed for preparation of demonstration frame with inked foreground parts which will be used as sample layer for proper color picking in COL application. This frame will be done by experienced artist during background restoration phase.

- **Frames where special effects are located.** As we saw in *Section 2.4* there are several visual effects (i.e. handmade motion blur, local intensity fluctuation of selected area, etc.) which could not be inked by COL application so it is necessary to handle them as post-production task.

- **Optional background masks.** There are several scenes, where foreground parts are occluded by background parts (see *Figure 2.18* where foreground sun is occluded by background forest). Between foreground area and front background layer does not exist real bold contour as is usual in case of foreground area on truly background. Due to this circumstance contour detection algorithm will fail (see *Section 3.2.2*) and COL operator should make complicated virtual contours which is not easy task if we consider that he is endowed only on COL's user interface not suitable for real painting. We should solve this problem by preparation of background masks during background restoration phase using image manipulation software (see *Figure 2.20*).



*Figure 2.20:* Helpful background mask.

Now if we have all information about input we explore the data flow diagram of whole inking process (see *Figure 2.21*). We will start on hi-end post-production hardware (left blue big-tower station) where we prepare true-color image sequences and export them to uncompressed 24-bit TIFF images. Conversion to grey-scale PiNG format (see *Section 2.1*) and background restoration phase on PC platform using image manipulation software (gray mini-tower computer) follows. Also sample image with inked foreground as long as optional foreground masks will be prepared. When the colorized background plane is ready we perform motion estimation using camera tracking software on PC platform (blue desktop computer) to produce sequence of colored background frames. Sequence of colored backgrounds, original grey-scale frames with foreground parts, example image and optional background mask are input of COL application (in bold red circle). Sequence of full color frames with background and foreground composite and grey-scale foreground alpha-masks both in uncompressed 24-bit TIFF format are expected as output of the COL application. Both sequences will be imported back to hi-end post-production hardware for post-processing tasks.

Although data flow diagram on *Figure 2.21* was used in practice we developed much more efficient innovation (see *Figure 2.22*) that replaces external professional camera motion tracking application with

our proprietary semi-automatic motion tracker also for PC platform based on ideas presented in [Hrbek02] that has got lots of additional features. During tracking process it automatically extracts background plane without foreground parts (areas that are never visible are marked with black pixels) and it preserves information about rectangular area of current frame on the big background plane. After this extraction it is much easier to apply ink on this plane using image manipulation software on PC platform and more we do not have to create image sequence of background frames (this will save lots of disk space if we consider that only one frame in compressed 24-bit PiNG format consume in average about 500kB), we only pass the big plane with restored background as input of the COL application while we exploit numerical information about rectangles for each corresponding frame in grey-scale sequence to extract correct rescaled background. This information also helps COL application to predict motion of foreground parts and saves lots of work connected with manual shifting of prediction masks as we will see in *Section 5.3*.

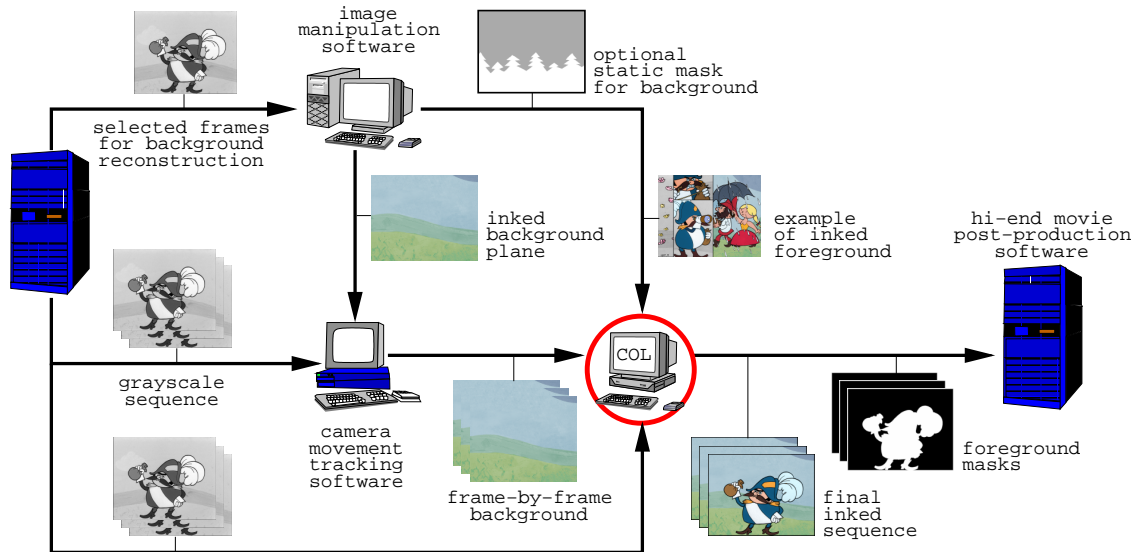


Figure 2.21: Generic data flow diagram of inking process.

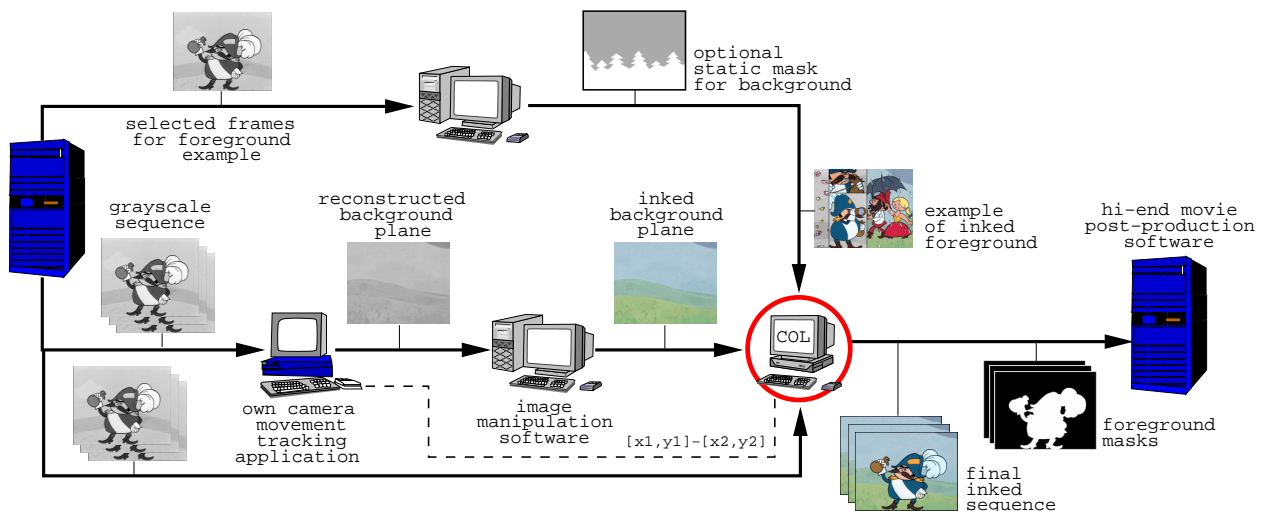
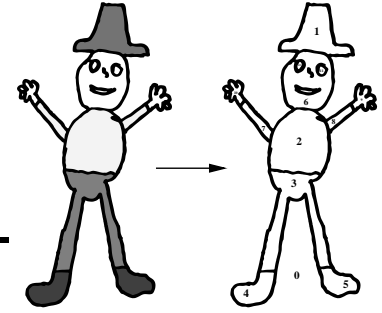


Figure 2.22: Optimized data flow diagram of inking process.

# 3

## Segmentation



In this chapter we will present a short overview of the recent image segmentation techniques and several practical results based on experiments with different methods which are suitable for our case, including *multi-level thresholding*, *watersheds* and *edge detection* to show main disadvantages that lead us to the development of a novel, robust, almost unsupervised segmentation technique that is able to produce the requested fine detail segmentation with very low computational costs.

It is clear that the segmentation is one of the main tasks in a cartoon inking process. Our problem is very similar to the *reverse engineering* because we know that in the input grey-scale frames there are scenes created artificially using a composition of planar objects. We need to extract them to acquire original standalone layers. That allows us to simplify the following inking process.

Moreover if we know that our input image is a picture of the artificial composition of several foreground and background layers and even if foreground parts are bounded with bold contours then our task is going to be much more easier compared with general image segmentation techniques focused on the natural photos or CT and MR images.

Because the artist's and his viewer's imagination deals with simple homogenous surfaces marked out with well defined boundaries, the best information which we could earn from the input image is the vector based description of the scene, which consists of curves and filled areas. This type of description will simplify the following manipulation. It is clear that the vectorised information is much more valuable for computerised image understanding than the original single resolution intensity matrix.

For our purpose it is not necessary to compute a complete image vectorisation. It could be finally understood as a disadvantage because of e.g. curve approximation which does not exactly follow original shapes created by artist. We only need to localize and identify each important region in the input grey-scale image using its original fixed resolution. More technically, we convert it into the indexed bitmap where each region has a unique index.



Figure 3.1: Different precision of segmentation: (from left to right) grey-scale image, nearly ideal segmentation, contour based segmentation, real regions with different color index.

To make a better understanding see *Figure 3.1*. It demonstrates which results are expected as an output of our segmentation process. Some regions are classified as foreground and other as background but



only foreground parts will continue to the next stages of the colorizing process so background parts should be merged together with one region index. Last but not least important type of area is a contour which represents the visible boundary between foreground and background parts.

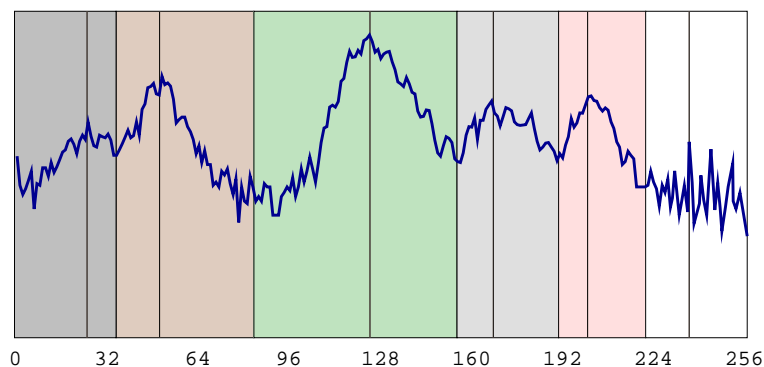
On *Figure 3.1* we could also see an omitted inner foreground region without bold contour (cobble's cheek). If it represents only local intensity changes that do not influence color selection process and only modulate final color brightness then it could be omitted. But that is not the case. Cobble's cheek should be more rosy than his skin. As we stated in *Section 2.4* this effect is out of our responsibility because it will be solved during the post-production phase. On the other hand similar properties have also inner contours. They could be also treated as unimportant (cobble's eye, ear, fingers, etc.) but if we want to e.g. locally enhance the contour contrast or exactly compute intensity statistics of proper region we have to know where they are located.

### 3.1 Previous work

Now when we know what degree of the source image segmentation precision we want, we should try to find out a proper segmentation method which is supposed to be robust, mostly unsupervised and fast. We will analyze and prove several classical and widely used segmentation techniques and decide which of them are suitable for our needs.

#### 3.1.1 Thresholding

First, easy to implement and fast method is the simple image *thresholding* on an intensity domain. This technique has a distant connection with the analysis of the input image histogram. Frequency peaks in the intensity histogram usually represent average intensities of the significant regions in the original image.



*Figure 3.2:* Segmentation via multi-level thresholding.

Segmentation via *multi-level thresholding* is done by peak finding algorithm which finds significant peaks in the image histogram and divides the original intensity domain to several sub-intervals. Bounds of each sub-interval are localized near the lowest frequency value of the valley between two detected peaks. Pixels from the original grey-scale image with intensities inside these intervals could be converted to a proper sub-interval index. With this straightforward process we could easily produce final image segmentation represented as indexed bitmap with the same resolution as original image.

#### 3.1.2 Homogeneity

Thresholding approach could be generalized with histogram on *homogeneity* domain as was shown in [Cheng00], where *homogeneity* is understood as normalized product of two local statistics: standard deviation (using window  $5 \times 5$ ) and discontinuity of intensities (*Sobel* operator using window  $3 \times 3$ ). The main advantage of this generalized histogram is that it takes in account not only local but also global information. *Cheng* count in to the homogeneity histogram only pixels with uniform neighbourhood (see *Figure 3.3*) to eliminate non-uniform pixels from global statistic and to avoid false detection on small region peaks.

Although this innovation made the thresholding segmentation technique more robust we can never develop such a powerful peak finding algorithm which would be able to avoid the oversegmentation raised by noise overlapping as we can see on *Figure 3.2*. *Cheng* used the *region merging* technique based on color information, which unfortunately is not suitable for our grey-scale case. But we could use *region*

*merging* by variational framework based on energy minimization using multi-scale pyramid [Koepfler94]. Unfortunately, that approach has  $\mathcal{O}(N \log N)$  complexity. A review of other multi-scale methods can be found in [Schneider00]. Recent research on the multi-scale techniques lead us to the interesting wavelet based semi-automatic segmentation [Haenselmann00].



Figure 3.3: Review of the different homogeneity thresholds: (from left to right) 0.01, 0.05, 0.10, 0.20, 0.50, where black pixels are not homogeneous.

Results of *region merging* with certain predefined homogeneity criteria are very similar to *color quantization* and *region growing* technique presented in [Deng01]. Another good successor of homogeneity approach is adaptive *clustering* by extraction of color and texture features [Chen02], which is suitable for natural images. We should not leave out important segmentation structures which exploit the area homogeneity feature. They are known as *active contours* or *snakes* (see [Yezzi97] and [Hug99]). We define them as (usually closed) spline curves that approximate the boundary of the homogeneous area by an energy minimization process. They are widely used as intelligent scissors in digital image editing software and because of that they unfortunately are not suitable for our task since they represent only a good tool for a human-driven segmentation.

Most of methods presented above are not appropriate for our case because they use color as a really helpful source of information or because they are human-driven. Other unsupervised grey-scale oriented methods based on homogeneity feature promise surprisingly good results but unfortunately their implementation would not be easy and finally the expected processing speed could be far away from the accepted limit of interactive performance. But we have not mentioned the well known *watersheds* segmentation method yet, which is based on a grey-scale morphology framework and which should be suitable for our case.

### 3.1.3 Watersheds

If we consider grey-scale image as a discrete representation of a 3D map of the mountain chain where each intensity corresponds to a certain height level than we could define *watersheds* as the contour lines separating the catchment basins. Each basin belongs to different local height minima. If a drop of water falls at any pixel of the current basin then it will flow down to its local minima. The watersheds algorithm simulates the real nature phenomenon of the rain water flooding.

The watersheds algorithm has three phases. First the gradient magnitude image is prepared. This phase consists of an advanced low-pass gaussian filtering to remove impulsive noise preserving image sharpness (see [Haris98] and Figure 2.16) and gradient magnitude calculation based on classical gradient operators (the same as in homogeneity approach: *Sobel* or *Perwitt*). Second phase is a pixel intensity sorting with histogram hash table (to reach  $\mathcal{O}(N)$  complexity) followed by incremental *immersion* of sorted pixels while the catchment basins are assigned to unique index which is propagated upwards using *neighbourhood queues* (see [Vincent91]).

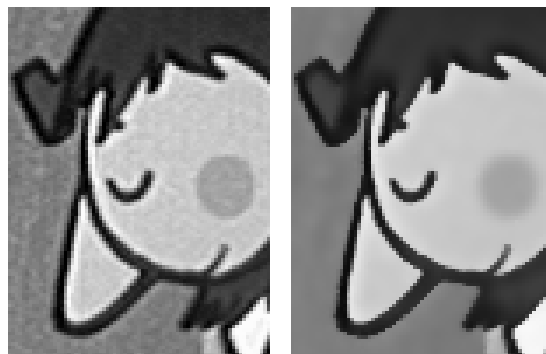
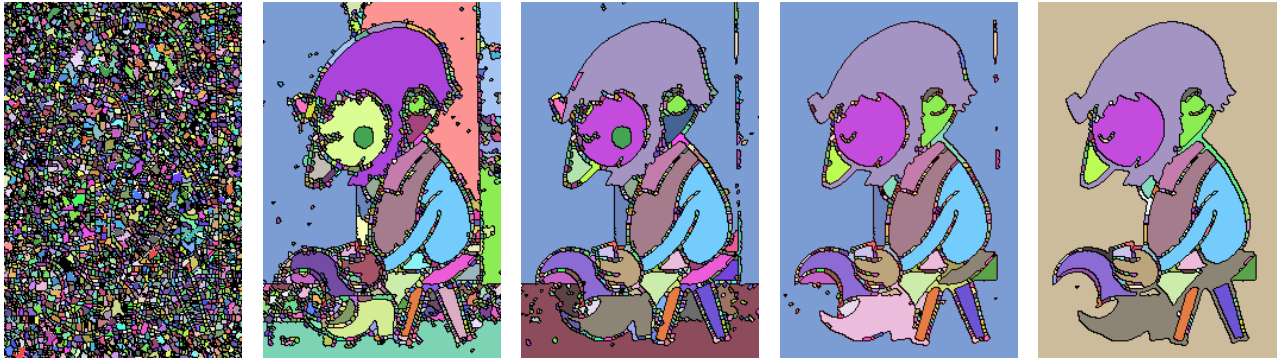


Figure 2.16: Selective gaussian blur.

After this second phase the initial segmentation of the input image is ready. Unfortunately the watersheds algorithm is sensitive to noise and high-frequency textural information so we usually receive lots of catchment basins which yield the typical oversegmentation problem (see *Figure 3.4* on the left). To solve this problem we perform third phase of the watersheds algorithm. There are several approaches: user-driven, when we manually select a small number of basins (*markers*) containing an important regional minimum (see [Meyer90]) or unsupervised technique based on fast nearest neighbor merging of region adjacency graph (see [Haris98]) which is more suitable for our case. Also spatio-temporal coherency should be exploited to perform labelling of the catchment basins via *Markov random field* model using maximization of the conditional a posterior probability (*MAP*) of a given label field (see [Patras01]).

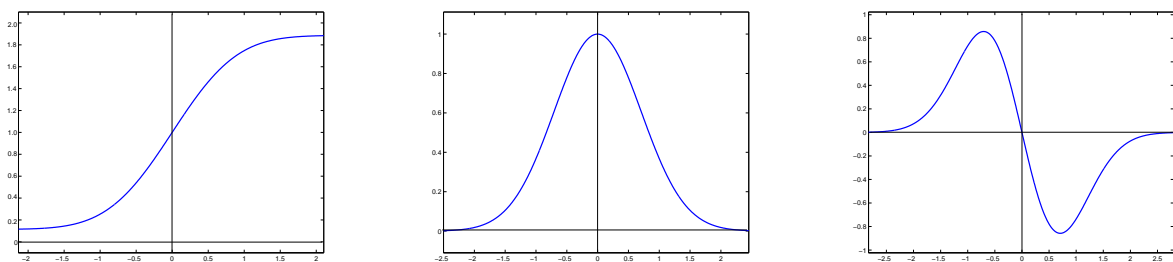


*Figure 3.4:* Progress of the watersheds oversegmentation reduction based on nearest neighbor merging.

Even that the final segmentation result (see *Figure 3.4* on the right) seems to be correct we unfortunately could not accept it because of loss of important details during catchment basins reduction process. Especially the drop-outs of the segments which belong to the bold contours are unacceptable. Considering the fact that the whole segmentation process takes more than one second per frame (on a 750MHz CPU), we have to conclude that watersheds algorithm is also unsuitable for our purpose.

### 3.1.4 Edge detection

Another approach to image segmentation widely used in computer vision is based on *edge detection*. The main idea came out from an observation that uniform parts of a segmented image are usually separated asunder by strong or weak edges. The edges could be understood as significant intensity differences of the neighbor pixels in an image matrix. Spatial location of edges in the image represented by continuous 2D function is exactly defined as a maximum of the first-order derivatives or better as second-order derivatives zero crossings of the image function (see one-dimensional analogy on *Figure 3.5*). If we discover the exact edge locations we can simply divide the image into several regions considering the fact that edges are their topological boundaries.

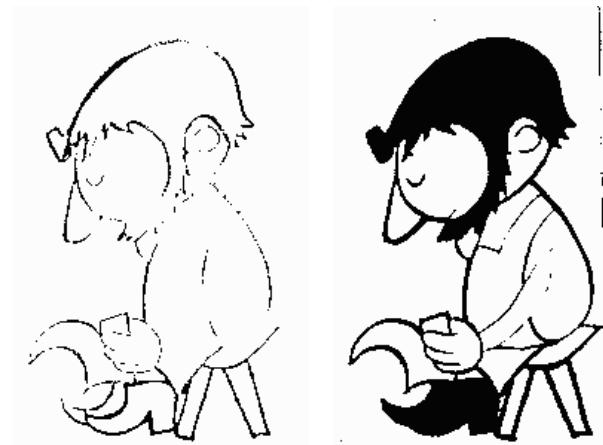


*Figure 3.5:* Edge on continuous one-dimensional function: (from left to right) image intensity, first-order derivatives, second-order derivatives. Middle vertical line in each graph represents the edge location.

The problem of the edge detection is well studied and lots of more or less robust approaches has been developed (see [Ziou97] and [Heath96] for detailed problem description and overview of existing techniques). This state of the research leads us to conclusion that the edge detection is not a trivial problem. If we want to develop the best detector for our conditions we have to perform many experiments. But our

case is really different from usual digitized images. Cartoon edges are visible as much as is possible. An artist needs to construct a clear conception of shapes in the scene in the viewer's mind. Our edges are bold contours represented by two strong intensity gradients appertain to their boundaries. What we are looking for is not an edge but a contour. So our task is to develop robust and fast contour detector.

The first idea how to find bold contours is again *thresholding* now only single-level. Contours look like to be only black parts in the image and so they certainly have own peak in the intensity histogram. This is true but not exactly. Unfortunately the contour's boundary edges are in fact strong intensity steps represented by high-frequency components in the 2D image signal. If we reduce the resolution we usually perform supersampling to avoid visible intensity aliasing of the edges on the resulting image. This anti-aliasing technique converts the high-frequency components to the smooth change of the neighbour pixel's intensities. This smoothing cause that really black pixels are located only in the center of the contour. The worst case is the situation where the original contour is thin as the one pixel and is sampled between two raster pixels. The contour peak is usually hidden in noise of neighbor intensities or if it is detectable then we must trade off between contour continuity and authenticity to select proper thresholding interval (see *Figure 3.6*).

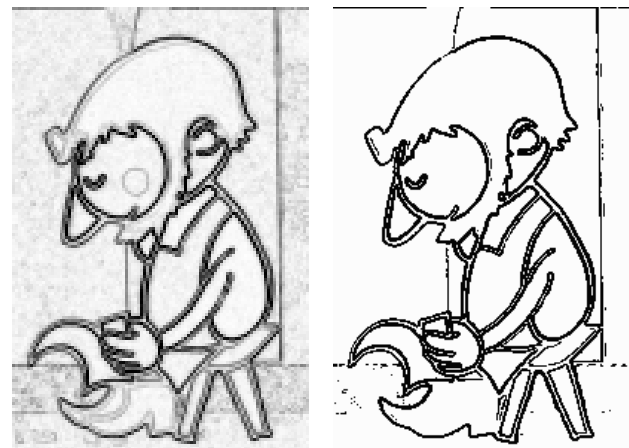


*Figure 3.6*: Contour thresholding: weak connectivity and false detection.

The problem lies in intensity level. Although the contours look like to be dark they significantly vary in intensities. We have to move our attention from an absolute intensity level to its differences, which is the well known first-order derivative approximation approach early developed by *Sobel* and *Perwitt* (see [Perwitt70] and [Mitchell88]). We combine two simple convolutions with 3x3 windows (horizontal and vertical) to reach first-order derivative approximation:

$$\Delta_{xy} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \circ \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}.$$

This two-pass convolution produces the image where pixel's intensity corresponds to the magnitude of the local edge gradient in the original image. To extract locations of edges according to definition we have to found local maxima of the intensity level. This task can be approximated by magnitude thresholding (see *Figure 3.7*).



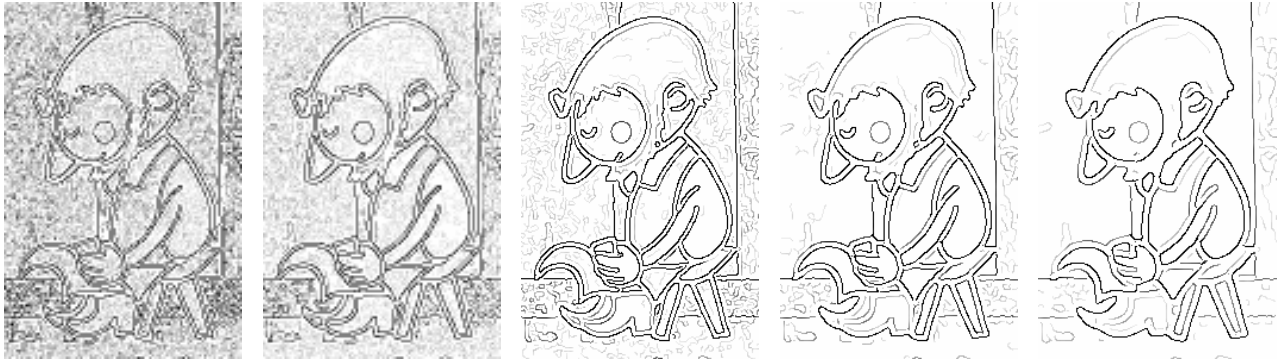
*Figure 3.7*: Sobel edge detection: first-order derivative (left) and magnitude thresholding (right).

But we are unable to locate edges correctly using magnitude thresholding. Additionally, some weak edges that are also important as contour borders are dropped out. We need a more robust technique to find them. The most popular advanced edge detector based on the first-order derivative framework developed by *Canny* possibly solves our problem.

*Canny* derives optimal convolution filter which is able to extract ideal step edge from one-dimensional signal degraded by Gaussian noise [Canny86]. The quality of that filter was evaluated in three criteria: good detection, good localization and unique response to a single edge. The final detector was derived by several symbolic and numeric tools and it is proven that it could be approximated with first-order derivative of the Gaussian (see *Figure 3.5*: 1D Gaussian ( $\mathbf{G}$ ) in the middle and its first-order derivative ( $\mathbf{G}'$ ) on the right):

$$\mathbf{G} = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}, \quad \mathbf{G}' = -\frac{x}{\sqrt{2\pi}\sigma^3} e^{-\frac{x^2}{2\sigma^2}}.$$

*Canny* detector extension for the discrete two-dimensional signal is not a trivial task because the 2D edge has additional properties compared with its 1D analogy, e.g. orientation. We have to convolve the image with the directional first-order derivative of the 2D Gaussian while edge orientation is not known. Second problem is the unique response criterion that is usually solved by thresholding with hysteresis (computationally expensive). Lots of work has been done on the 2D extension of the basic 1D *Canny's* idea, for further study see e.g. [Shen86] or [Ding01]. You can see an example of the different  $\sigma$  scale responses of the simple 2D *Canny* detector on *Figure 3.8*.



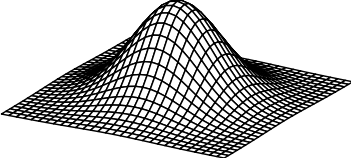
*Figure 3.8*: The response of the *Canny* edge detector in different scales ( $\sigma = 0.5, 1.0, 1.5, 2.0, 2.5$ ). Grey levels denote to the corresponding edge strength or weakness.

Unfortunately advanced edge detection techniques like *Canny* detector are complex and computationally expensive. They produce additional information which is not essentially valuable for our purpose. We need to know exact edge location only. Edge strength or direction is irrelevant information for us. We need faster approach which will be able to satisfy only the second *Canny's* criterion – good localization.

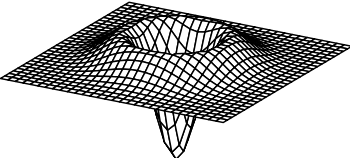
Second-order derivative approximation of the image function will help us. Before we could apply this technique we have to remove impulsive noise because while we increasing the derivative order we enhance high-frequency parts of the 2D image signal. They should destroy the main information that we need to extract.

Before we start inquiry into the noise suppression techniques (e.g. selective gaussian blur in *Section 3.1.3*, see also [Mrázek01] or [Nitzberg93]) we have to state that *Marr's* and *Hildreth's* theory of edge detection [Marr80] take in account a model of the human eye's perceptual system. If we consider that cartoons are made to be perceived by human eye then it should be really interesting to learn how it works. *Marr* and *Hildreth* proved that human shape understanding is based on the process that is very similar to the image ( $\mathbf{I}$ ) convolution with the two-dimensional Laplacian of Gaussian filter ( $\mathbf{L} \circ \mathbf{G}$ ).

This filter should be constructed if we know that:

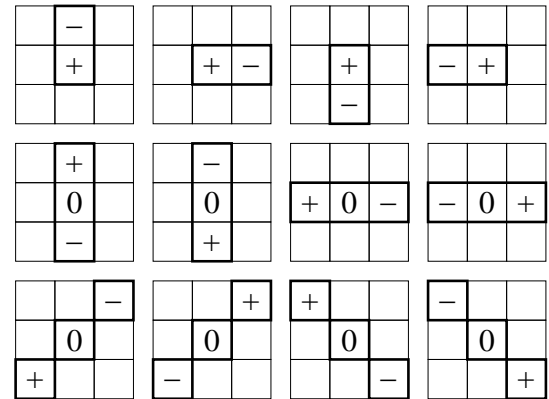
$$\mathbf{G} = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad \Rightarrow \quad \mathbf{L} = \nabla^2 = \nabla \circ \nabla = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$


where  $\mathbf{G}$  is Gaussian and  $\mathbf{L}$  is classical Laplacian operator both in two dimensions. We could take the advantage of the convolution linearity  $\nabla^2(\mathbf{G} \circ \mathbf{I}) = (\nabla^2\mathbf{G}) \circ \mathbf{I}$ , so  $\nabla^2\mathbf{G}$  should be precalculated by symbolic derivation. Final  $\mathbf{L} \circ \mathbf{G}$  convolution filter should be expressed using following formula:

$$\mathbf{L} \circ \mathbf{G} = \nabla^2\mathbf{G} = \frac{1}{\pi\sigma^4} \left( \frac{x^2 + y^2}{2\sigma^2} - 1 \right) e^{-\frac{x^2+y^2}{2\sigma^2}} \quad \Rightarrow$$


This filter performs two operations by one-pass convolution: Gaussian removes noise and Laplacian estimates second-order derivative of the noise-free image function. It is now much easier to find the exact edge locations using the zero crossing test on such convolved image. But correct zero crossing test do not consist of trivial thresholding of pixels with value near zero. Using this technique we destroy zero crossings conductivity.

If we use floating point arithmetic the rotating  $2 \times 1$  mask have to be used. It covers the basic four local directions of the zero crossing curves (see *Figure 3.9* the first row). We localize zero crossing on the current pixel if and only if one or more of the four rules from the first row on *Figure 3.9* are satisfied. This happens only if a proper pixel has positive value and one or more of its four neighbours have negative value. Moreover 4-continous rule should be extended by 8-continous masks, especially if we use integer arithmetic (see *Figure 3.9* the second and third row). To be even more precise, *Clark* developed authenticating method for detecting “phantom” edges generated by zero crossing test [Clark89], where “phantom” edge is defined as local minimum extreme in first order derivative of image function.



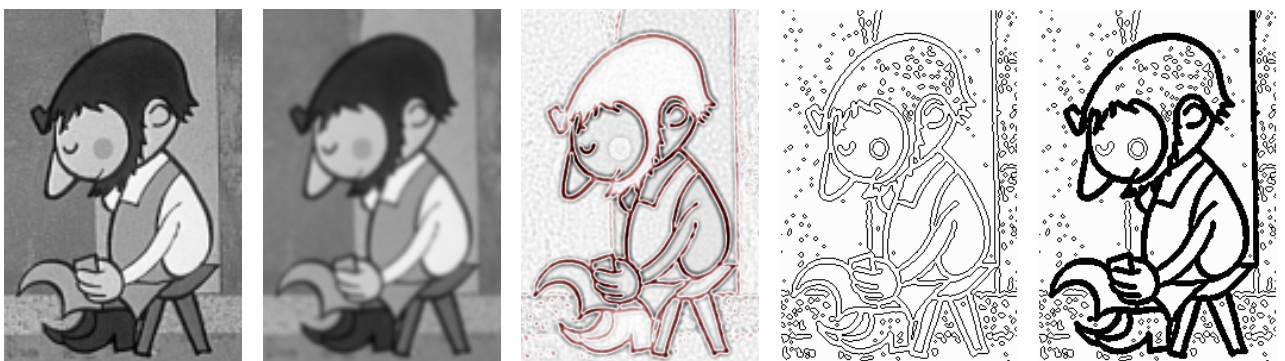
*Figure 3.9:* Zero crossing test masks.

There is one important parameter of the  $L \circ G$  filter, its standard deviation  $\sigma$ . It enables us to select proper filter scale to fit into our range of interest. If we vary  $\sigma$  we are going through image scale-space (see [Witkin86]). Edges that are important for us reside only in a small interval of this space. We have to find it. See *Figure 3.10* for several samples from  $L \circ G$  scale-space visualised by zero crossings. There we could observe that the scale of our edges seems to be between  $\sigma > 1.0$  and  $\sigma < 2.0$ . This scale-space interval will be also discussed in *Section 3.2*.



*Figure 3.10:* Scale-space of the  $L \circ G$  filter: (from left to right)  $\sigma = 1.0, 1.5, 2.0, 3.0, 4.0$ .

Another important feature of the  $L \circ G$  filter is that its zero crossings form closed curves, sometimes so called “spaghetti effect”. What does it mean for our case? Each contour consists of two boundary edges. The contours of one isolated foreground object form a continuous closed area which could be represented by one region in final segmentation. The boundary edges are topological boundaries of this area. If we simply fill this area with constant color using e.g. classical *flood-fill* algorithm we receive searched contour (see *Figure 3.11*)!



*Figure 3.11:* Contour detector based on Laplacian of Gaussian filtering in progress: (from left to right) original image, Gaussian smoothing, Laplacian approximation of the second-order derivatives, zero crossing, flood-fill.



Although the  $\mathbf{L} \circ \mathbf{G}$  filter seems to be the best segmentation tool for cartoons with bold contours of all techniques mentioned above, we must first discuss its computational complexity. Brute force convolution with  $\mathbf{L} \circ \mathbf{G}$  filter using e.g.  $\sigma = 1.60$  (same as on *Figure 3.11*) on recommended basis  $19 \times 19$  amounts to approximately 150 millions floating point multiplications per one PAL frame (over 4 seconds on 750MHz CPU). This CPU load is far over our interactive performance limit. Luckily, as we will see in next section, due to several decomposition techniques we can significantly speed up  $\mathbf{L} \circ \mathbf{G}$  convolution process with the same filtering results.

## 3.2 Contour detector

In this section we will describe in detail the fast contour detection algorithm that will be used as basic pre-processing tool for final image segmentation. We will explain how to efficiently compute full frame  $\mathbf{L} \circ \mathbf{G}$  convolution with correct zero crossing and how to find main closed contours in the source image using the fast selective span-recursive *flood-fill* algorithm and the morphological erosion operator.

### 3.2.1 Filter design

As we saw in the previous section, the brute force full frame  $\mathbf{L} \circ \mathbf{G}$  filtering is a computationally expensive task. Even if we consider that the recommended window size for discrete approximation of the  $\mathbf{L} \circ \mathbf{G}$  filter is  $\lceil 8\sqrt{2}\sigma \rceil$ . First idea how to speed up convolution with  $\mathbf{L} \circ \mathbf{G}$  operator exploits the fact that  $\mathbf{L} \circ \mathbf{G}$  can be approximated with difference of two Gaussians ( $\mathbf{D} \circ \mathbf{G}$ ) with several standard deviations [Nishihara81]. Each Gaussian window should be separated into the two one-dimensional row and column convolution masks. This will reduce total number of multiplications more than fourfold but resulting image is not exactly  $\mathbf{L} \circ \mathbf{G}$ . Later King proved that  $\mathbf{L} \circ \mathbf{G}$  could be exactly decomposed into the sum of two separable filters [King82]:

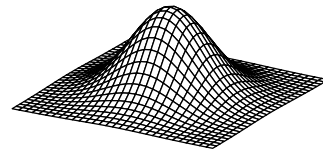
$$\nabla^2 \mathbf{G}(x, y) = \mathbf{h}_1(x) \cdot \mathbf{h}_2(y) + \mathbf{h}_2(x) \cdot \mathbf{h}_1(y)$$

where

$$\mathbf{h}_1(\rho) = \frac{1}{\sqrt{2\pi}\sigma^2} \left(1 - \frac{\rho^2}{\sigma^2}\right) e^{-\frac{\rho^2}{2\sigma^2}} \quad \text{and} \quad \mathbf{h}_2(\rho) = \frac{1}{\sqrt{2\pi}\sigma^2} e^{-\frac{\rho^2}{2\sigma^2}}.$$

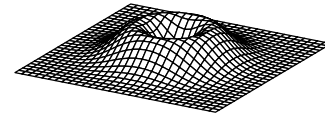
See *Figure 3.12* for  $\mathbf{h}_1(\rho)$  function plot. Filter  $\mathbf{h}_2(\rho)$  is classic Gaussian (see *Figure 3.5* in the middle). Additionally, *Chen et al.* proved that  $\mathbf{L} \circ \mathbf{G}$  filtration should be done in two passes using classical Gaussian with the same standard deviation and smaller  $\mathbf{L} \circ \mathbf{G}$  with a lower standard deviation. To show that we have to use *Fourier* transformation [Chen87]:

$$\mathcal{G}(u, v) = \exp \left[ -\frac{\sigma^2}{2} (u^2 + v^2) \right] \Rightarrow$$



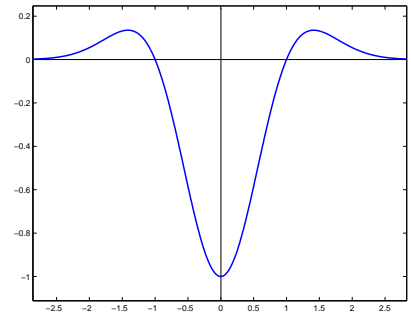
and

$$\mathcal{L} \circ \mathcal{G}(u, v) = (u^2 + v^2) \exp \left[ -\frac{\sigma^2}{2} (u^2 + v^2) \right] \Rightarrow$$



where  $\mathcal{G}(u, v)$  and  $\mathcal{L} \circ \mathcal{G}(u, v)$  are continuous spatial *Fourier* transformations of  $\mathbf{G}(x, y)$  and  $\mathbf{L} \circ \mathbf{G}(x, y)$  respectively. We know that two-pass filtering in spatial domain should be done by one-pass filtration in *Fourier* domain by the window that was computed as a multiplication of two convolution windows from each spatial pass. This process could be reformulated as follows:

$$\mathcal{L} \circ \mathcal{G}(u, v) = \exp \left[ \frac{\sigma^2}{2} \left(1 - \frac{1}{k_\sigma^2}\right) (u^2 + v^2) \right] \times (u^2 + v^2) \exp \left[ \frac{\sigma^2}{2} \left(-\frac{1}{k_\sigma^2}\right) (u^2 + v^2) \right],$$



*Figure 3.12*: The  $\mathbf{h}_1(\rho)$  filter.

where  $k_\sigma$  is a *reconstruction constant*. This constant controls the trade off between the standard deviations of the decomposed filters in the spatial domain:  $\sigma_G = \sigma\sqrt{1 - 1/k_\sigma^2}$  and  $\sigma_L = \sigma/k_\sigma$ .

Before we will be able to select proper  $k_\sigma$  we have to know that Sotak suggest in [Sotak89] step-by-step operator design procedure to estimate the best standard deviation for Gaussian and smaller  $\mathbf{L} \circ \mathbf{G}$  from the given brute force  $\sigma$ . They take in account also allowable aliasing energy  $p_a$  in the spectrum of the truncated digital approximation of the  $\mathbf{L} \circ \mathbf{G}$  filter. If we truncate function in spatial domain then we receive periodical repetition in frequency spectrum and vice versa. This aliasing energy can be expressed for the Gaussian and  $\mathbf{L} \circ \mathbf{G}$  spectrum as follows:

$$\frac{100 - p_a}{100} = \frac{\sigma_G^2}{\pi} \int_{-\alpha_G}^{\alpha_G} \int_{-\alpha_G}^{\alpha_G} e^{-\sigma_G^2(u^2+v^2)} dudv \quad \text{and} \quad \frac{100 - p_a}{100} = \frac{\sigma_L^6}{2\pi} \int_{-\alpha_L}^{\alpha_L} \int_{-\alpha_L}^{\alpha_L} (u^2 + v^2)^2 e^{-\sigma_L^2(u^2+v^2)} dudv$$

where  $\alpha_G$  and  $\alpha_L$  are *aliasing frequencies*. Sotak computed them by numerical integration for the given percentage of aliasing energy  $p_a$  and standard deviations  $\sigma_G$  and  $\sigma_L$ . It is possible to prepare precomputed  $\sigma$ -independent functions  $A_G(p_a) = \alpha_G\sigma_G$  and  $A_L(p_a) = \alpha_L\sigma_L$ . According to [Sotak89] the best  $k_\sigma$  should be tuned by  $k_\sigma = \sigma\pi/(A_L k_d)$ , where  $k_d = \sigma\pi/\sqrt{A_L^2 + A_G^2}$  is *decimation constant*.

It is now interesting to show which  $p_a$  is suitable for our case. If we increase the  $p_a$  then we propagate smoothing ( $\sigma_G$  will grow up and  $\sigma_L$  will decrease, see Figure 3.13). We selected experimentally  $p_a = 10\%$  as a compromise between aliasing and smoothing to reach best filtering quality.



Figure 3.13: Influence of the allowable aliasing energy: (from left to right)  $p_a = 1\%, 5\%, 10\%, 15\%, 25\%, 50\%$ .

Now we could refine the scale-space interval (see Figure 3.10) by assumption that integer part of our decimation constant should be  $[k_d] = 1$ . Otherwise if  $[k_d] < 1$  then our smaller  $\mathbf{L} \circ \mathbf{G}$  decomposition will be undefined and we should use the brute force  $\mathbf{L} \circ \mathbf{G}$  filter with the lowest reasonable support  $7 \times 7$  or better enter the sub-pixel resolution by an image upsampling (see Figure 3.14 on the left side). With sub-pixel resolution we are able to disintegrate one-pixel distant zero crossings that represent boundaries of a very tight region. On the other side if  $[k_d] > 1$  then  $\mathbf{L} \circ \mathbf{G}$  will be sensitive on edges in decimated image with half resolution. This is useful if we detect edges on zoomed image where contours are nearly twice as thicker as in standard zoom and so the  $\mathbf{L} \circ \mathbf{G}$  detector using  $[k_d] > 1$  filter estimates unwanted zero crossings inside bold contours (see Figure 3.14 on the right side).

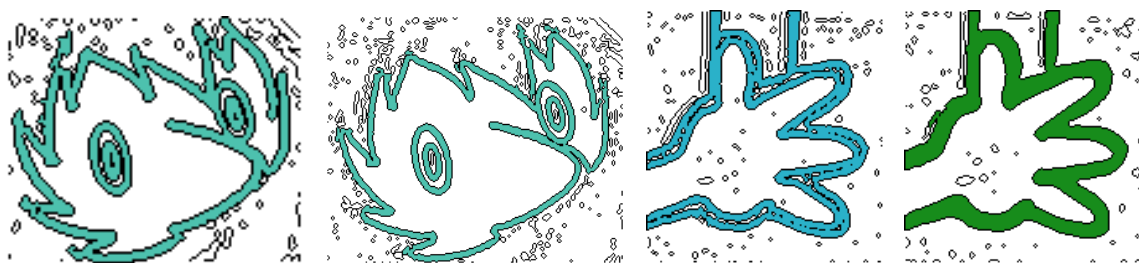


Figure 3.14:  $\mathbf{L} \circ \mathbf{G}$  filtering with different  $k_d$ :  $[k_d = 1, \sigma = 1.23]$  vs.  $[k_d < 1, \sigma = 1.23]$  sub-pixel accuracy (left),  $[k_d = 1, \sigma = 1.23]$  vs.  $[k_d > 1, \sigma = 2.00]$  decimated resolution (right).

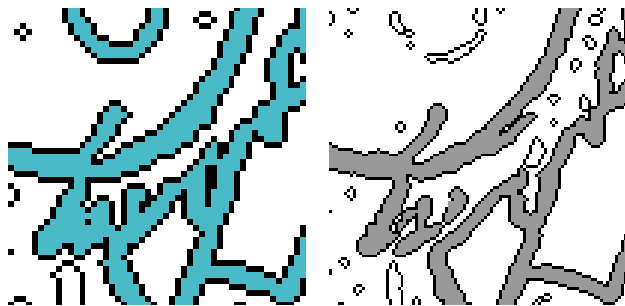


If we follow the *Sotak's* instructions we found that this constraints yield us to eight different  $\mathbf{L} \circ \mathbf{G}$  decompositions as we can see on *Table 3.1*. For a given  $\sigma$  interval (first column) we should decompose brute force  $\mathbf{L} \circ \mathbf{G}$  convolution with support  $\lfloor 8\sqrt{2}\sigma \rfloor$  (second column) into the two-pass  $\mathbf{G}$  and  $\mathbf{L} \circ \mathbf{G}$  filtering using smaller separated (*horizontal + vertical*) one-dimensional supports on full resolution or the decimated image ( $\lfloor k_d \rfloor = 2$ ). To preserve the same output resolution, *Sotak* performs up-sampling using bilinear interpolation. He also recommends to apply the *DC-padding* instead of the *zero-padding* to avoid an erosion of the zero crossings near the image boundaries. Last column in *Table 3.1* presents experimentally measured relative performance gains of decomposed and separated convolutions compared with the brute force filtering including time spent on bilinear interpolation.

$\sigma$	$\mathbf{L} \circ \mathbf{G}$	$\lfloor k_d \rfloor$	$(\mathbf{L} \circ \mathbf{G}) \circ \mathbf{G}$	speed
(0.80, 0.95)	9x9	1	(7+7) $\circ$ (3+3)	4x
(0.95, 1.20)	11x11	1	(7+7) $\circ$ (5+5)	5x
(1.20, 1.50)	13x13	1	(7+7) $\circ$ (7+7)	7x
(1.50, 1.60)	17x17	1	(7+7) $\circ$ (9+9)	10x
(1.60, 1.70)	19x19	2	(7+7) $\circ$ (5+5)	9x
(1.70, 1.90)	23x23	2	(7+7) $\circ$ (7+7)	8x
(1.90, 2.15)	25x25	2	(7+7) $\circ$ (9+9)	13x
(2.15, 2.40)	27x27	2	(7+7) $\circ$ (11+11)	19x

*Table 3.1*: Useful decompositions of brute force  $\mathbf{L} \circ \mathbf{G}$ .

While *Sotak's* analysis does not cover filtering on sub-pixel resolution the *Table 3.1* presents only decompositions for  $\lfloor k_d \rfloor = 1$  and  $\lfloor k_d \rfloor = 2$ . If we want to work with a sub-pixel accuracy we should analyze degradations caused by a bilinear extrapolation. See [Steger98] for precise development of sub-pixel accuracy framework for edge detection. For an example of possible distortion caused by the sub-pixel extrapolation see *Figure 3.15* where complicated hair shape caused separation of two zero crossings with the same spatial location. If we focussed only on topological properties of zero crossings we find out that the best behaviour of  $\mathbf{L} \circ \mathbf{G}$  filtering on a sub-pixel resolution was reached when  $\sigma \in (1.20, 1.60)$  using same filter decomposition as for  $\lfloor k_d \rfloor = 1$ .



*Figure 3.15*: Zero crossings in original resolution (left) and broken topology in sub-pixel extrapolation (right).

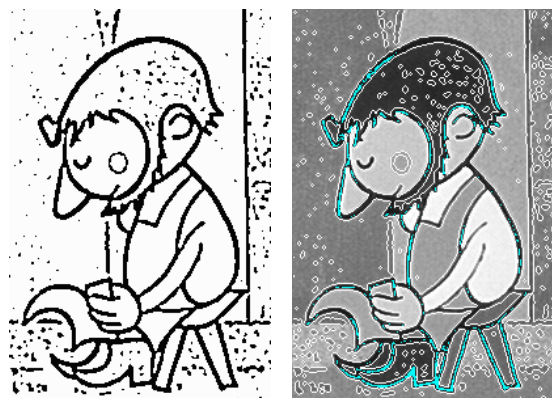
Although the sub-pixel accuracy should be helpful (see *Figure 3.16*) we cannot accept its computational and memory complexity in front of interactive performance limit and memory requirements on recent low-end PC workstations.

We have to emphasize that the purposed contour detection technique should be easily extended to the sub-pixel accuracy without any additional adjustments but with expectation of the fourfold slowdown and additional storage space with the same proportions.

### 3.2.2 Contour filling

Using already designed optimal  $\mathbf{L} \circ \mathbf{G}$  filter and correct zero crossing test (see *Figure 3.9*) we should create intensity invariant bitmap where black pixels indicate less or more significant edges. Now we are going to perform an automatic extraction of bold contours and wipe out the less important edges. As was mentioned in *Section 3.1.4* we should use fast 4-continuous span-recursive flood-fill algorithm to fill contour area bounded by zero crossings.

To identify pixels where we could start with flood-filling we will introduce a novel adaptive thresholding mechanism. First let us make the observation that this start point can be located only at a pixel which receives negative value after convolution with  $\mathbf{L} \circ \mathbf{G}$  filter. See *Figure 3.17* on the left side, where white and black pixels have positive, respectively negative value. We

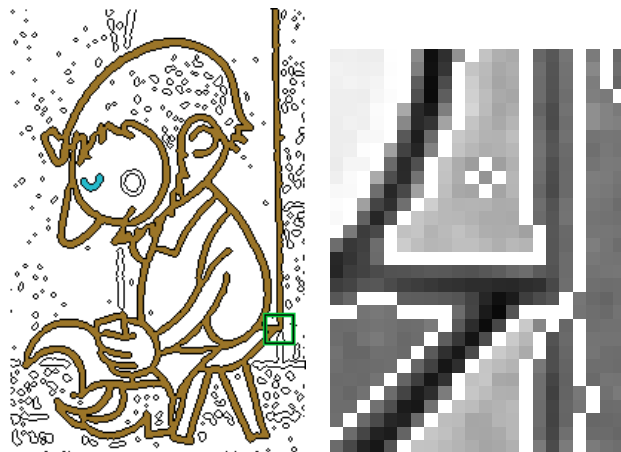


*Figure 3.17*: Pixels  $p_i$ : with  $\mathbf{L} \circ \mathbf{G}(I(p_i)) < 0$  (left) and  $I(p_i) = I_{min}$  (right).

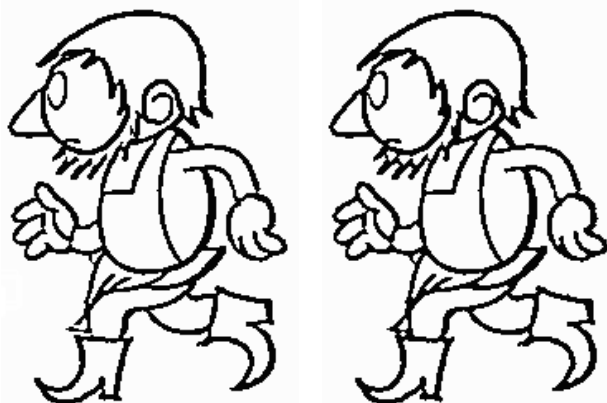
compute a global intensity minimum of these  $\mathbf{L} \circ \mathbf{G}$ -negative pixels  $p_i$  using values from corresponding original grey-scale image  $I$ :  $I_{min} = \min_{i:\mathbf{L} \circ \mathbf{G}(I(p_i)) < 0} \{I(p_i)\}$ . Pixels with this minimal intensity are the good start points for our iterative algorithm. We start flood-fill algorithm on these pixels and compute intensity median  $\tilde{I}$  of pixels from the area being filled. Now unfilled pixels with properties  $I(p_i) < k\tilde{I}$  and  $\mathbf{L} \circ \mathbf{G}(I(p_i)) < 0$  are the next start points for another flood-filling step, where  $k$  is convergence constant which was experimentally adjusted to  $\frac{1}{2}$ . We repeat this operation while median of new filled area is lower or same as current value.

This novel contour finding algorithm converges very fast. On the average it stops after the second pass while new median  $\tilde{I}_{k+1}$  is not bigger than  $\tilde{I}_k$ . This phenomenon illustrates *Figure 3.18* on the left, where brown contours were filled in initial  $I_{min}$  step and only the cobbler's eye was filled in second  $\tilde{I}$  step when the algorithm also terminated. By using this automatically predicted filling threshold  $\tilde{I}$ , we can usually find all the important bold contours in an image without any user interaction. Median  $\tilde{I}$  stays usually constant during whole sequence but sometimes it differ due to luminance fluctuation (see *Section 2.4.3*).

Sometimes it is reasonable to let the user feel free in changing the predicted filling threshold. The user can also run flood-fill algorithm manually on the selected position in image but this usually is not necessary since problems connected with false detection are rare. The main problem of the presented contour filling algorithm are T-junctions of  $\mathbf{L} \circ \mathbf{G}$  zero crossings (see *Figure 3.18* on the right). They connect foreground and background contours topologically to a single solid contour. If we fill foreground contour we also fill the background one. We will see later that if this wrongly detected contour does not bound closed area than it is almost harmless for *original background* technology (see *Section 2.2.1*) because for this inking technology are important only inner areas, not their boundary contours. But using *reconstructed background* technology (*Section 2.2.2*) the contour's boundaries are locations where composition with new background is performed. We sometimes observe a strange dark smoothing especially if the luminance of pixels from a reconstructed background is higher than pixel intensities in original grey-scale background (see *Section 4.2.1*).



*Figure 3.18*: Two steps of contour filling algorithm (left) and T-junction problem (right).



*Figure 3.19*: Contour extraction via image subtraction (left) and morphological erosion (right).

do not know which junctions are important. Instead of that we can apply an easy contour authentication test. We compute the minimal intensity value  $I_{min}$  from a small neighborhood of the contour pixel (e.g. win-

The first possible solution is a user intervention. The user can use coarse brushes to remove these unwanted contours manually. This user driven wiping out is also needed when bold dark contours reside in background area. Unfortunately, we are not able to develop successful removal technique that would solve this problem. But if we consider that this painting work should be recorded to independent retouching layer and applied on the following sequence frames then it is not such a time consuming task as it looks.

T-junctions can be removed sometimes by increasing the  $\mathbf{L} \circ \mathbf{G}$ 's  $\sigma$ . We are also able to automatically find out their locations using a fast corner or junction detector (see [Smith97] and [Shen00]) and create some sort of filling stoppers. Unfortunately, without further analysis we

dow 5x5) and compare it with the median value  $\tilde{I}_n$  from the last iteration of our algorithm. If  $\tilde{I}_n < I_{min}$ , we can conclude that the proper pixel does not come from a foreground contour. We also allow user to set this  $\tilde{I}_n$  threshold and size of neighborhood manually to reach better detection performance.

Finally, residual zero crossings have to be removed. This looks as a trivial task. We should perform simple image subtraction only. However, by using this straightforward operation we unfortunately destroy important topological characteristics. Compare carefully two different pictures on *Figure 3.19* where you can find out on the left one lots of broken contours, especially in the cobbler's beard. This is due to the fact that some zero crossings appeared on the two neighbour pixels and if we remove them we clearly break the contour connectivity.

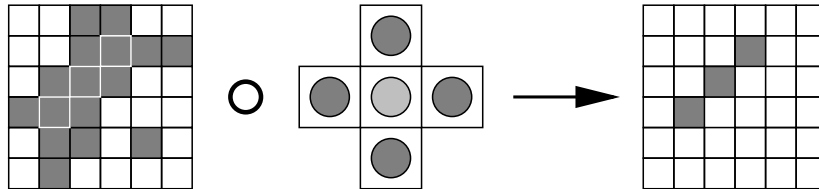


Figure 3.20: Image erosion using 4-continuity structure element.

To solve this problem we use morphological *erosion* operator. This operator was derived using the theory of *mathematical morphology*. For further study on this domain see [Haralick93] or [Serra93]. We will describe just the main idea of erosion operator. Erosion of a binary image is very similar to the convolution process on the grey-scale domain. We use also moving window known as *structure element* (see *Figure 3.20* in the middle). The origin of this window is placed on each pixel in binary image and the following relation is performed: if all pixels covered by the structure element are classified as edges (this means contour or zero crossing in our case) then we place edge pixel on origin position into the output image otherwise we place background pixel (see *Figure 3.20* for example). This process will reduce thickness of contours and wipe off the residual one pixel wide zero crossings. Additionally, it preserves 4-continuity of resulting image due to the used structure element's shape. The right picture on *Figure 3.19* illustrates practical results.

### 3.2.3 Region marking

Now we are ready for the final image segmentation. We have the binary image where black pixels and an image border represent contours and white pixels enclosed regions. We want to assign every closed region its unique index. We can achieve this by the selective flood-filling algorithm which starts new fill on every empty background pixel in our binary image using a unique filling index. This is a straightforward method, but we could do this even faster if we use the scan-line based 4-continuity two-pass region marking algorithm (see [Rosenfeld82]).

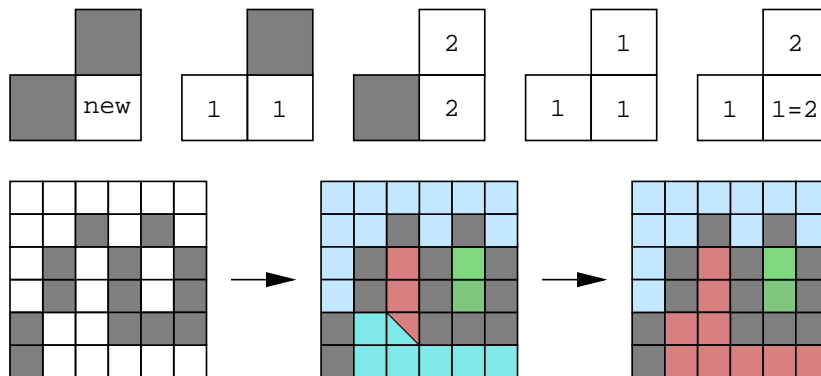


Figure 3.21: Two-pass region marking algorithm: marking rules (top) and algorithm in progress (bottom).

The first pass performs the initial marking driven by pixel rules from *Figure 3.21* on the top. If the top and left neighbor pixels belong to a contour then we create new marker index. If the top is contour and left is not or vice versa we simply propagate an existing marker. This is done also in case when both top and left pixels are not contours and have the same marker. But when they have different markers assigned we

conclude that markers from the top and from the left pixels are equivalent. This equivalent pairs have to be reindexed in the second pass of algorithm. The bottom image sequence on *Figure 3.21* illustrates these two phases. To reach better performance compared with the selective flood-filling, appropriate dynamic data structure representing equivalent markers has to be implemented. We create pointer matrix *M* with the same dimensions as the processed image *I* and the **marker** array for new markers. Each array item has its own (initially empty) list of equivalent markers and an initial marker index value:

```
marker[i].id=-(i+1);
marker[i].equ_list=NULL;
```

We fill matrix *M* with pointers to **marker** items for each background pixel ( $I[x][y]==\text{BACK}$ ) as follows:

```
IL=I[x-1][y]; IT=I[x][y-1];
ML=M[x-1][y]; MT=M[x][y-1];

if (IL==EDGE && IT==EDGE)
    M[x][y]=&marker[num++];

if (IL==BACK && IT==EDGE) M[x][y]=ML;
if (IL==EDGE && IT==BACK) M[x][y]=MT;

if (IL==BACK && IT==BACK) { M[x][y]=MT;

    if (ML!=MT) {
        ML->equ_list+=MT;
        MT->equ_list+=ML;
    }
}
```

The equivalence case is important. We create reciprocal equivalent links. The final reindexation is performed for each item from marker array by a recursive call of function that sets proper item index for all markers in the equivalence list:

```
for (i=0;i<num;i++)
    reindex(&marker[i],i+1);

void reindex(MARKER *m, int id) {
    if (m->id<0) { m->id=id;
        for (e=m->equ_list;e;e++)
            reindex(e,id);
    }
}
```

The positive value of  $m->id$  signalsizes that the *m* marker has been reindexed already and that we should terminate recursion.

You can see the contour based segmentation index map on *Figure 3.22*, the bottom right picture. Whole *Figure 3.22* visualizes basic steps of our novel contour based segmentation algorithm, which is suitable for cartoon images processing. Algorithm has two very important features: performance (on the average it does not exceed the full frame segmentation processing speed limit: 0.5 second on the 750MHz CPU) and accuracy (try to compare the resulting image with *Figure 3.1*).



Figure 3.22: Overview of segmentation process.

### 3.2.4 Region statistics

Now we should compute several region statistics useful for further operations. To do that we exploit original grey-scale image and already computed region map where each element points to the proper data structure with all needed statistic parameters for the corresponding region: intensity mean, standard deviation, intensity histogram, intensity median, maximum and minimum intensity, region size (in pixels), bounding rectangle (maximum and minimum coordinates) and centre of the gravity (image coordinates).

The region intensity median can be effectively computed with complexity  $\mathcal{O}(N)$  via histogram analysis assuming that the region size  $N$  is known. We simply start with zero intensity and during each step we increase this index and decrease the initial size  $N/2$  using frequency values corresponding to the actual intensity index until we reach value below zero. Histogram index from last step corresponds to the region intensity median:

```
size=region->size/2; region->median=0;
while (size>0) size-=region->histogram[region->median++];
```

### 3.2.5 Foreground and background

To divide foreground regions from background ones, we simply use the region size thresholding. For most scenes it is suitable to set the limit of the total image size to 15%. Regions with size below this threshold are classified as foreground, the others as background. The user can change this value but it was experimentally proved that this generic threshold is well posed and does not need fine tuning.



Figure 3.23: Character's topology differences.

Using size threshold we could exclude only large background areas. Unfortunately there are usually lots of small regions that look like foreground parts but they are actually background. Sometimes it is really hard even for a human to distinguish foreground or background regions because they have no specific properties if we use only comparison based on pixel intensities. To analyze that we have to perform deeper image analysis process which analyses e.g. character topology features. If we suppose that cartoon characters are only two-dimensional artificial abstractions of real objects we could not exploit lots of well studied

$2\frac{1}{2}$ D reconstruction based methods (see [Nitzberg93]). This problem becomes even more complicated because of the fact that artists sometimes do not complete basic topology features and use completely different shapes. Examine carefully Figure 3.23 where there are three frames with the same character from the same image sequence. The character's topology vary in each frame. Specifically, see the shoulder and the top of the sack. Due to these circumstances user has to find out all wrongly detected foreground regions and mark them with background area index. If the character's shape does not change a lot in the following frames the *position prediction* technique will perform an automatic correction (see Section 5.3.2).

## 3.3 Region growing

Although the contour based segmentation from Section 3.2 (see Figure 3.22) looks like a final product of our image analysis process, we have to perform one more important step. We already use this segmentation for region statistics calculations and in cases when we need to exactly know where contours, foreground and background parts are located so it is not only intermediate product but we also need to find out real region boundaries. These boundaries should be defined as medial axes of contour shapes also known as binary image *skeleton*. It will help us to estimate how deeply into the contour anti-aliasing we have to apply color flooding using the intensity modulation. On Figure 3.24 we can exactly see what does it mean. The left middle picture is colorized using original contour boundaries and the right middle picture using the contour skeleton.

While zero crossings locations are placed near the middle of the edge downhill and contour boundaries are not ideal step edges because of an anti-aliasing we sometimes omit visually important pixels during color modulation using contour base segmentation. If we correctly compute contour skeleton and then



enlarge shape of actual regions into the contour gaps we receive visually much better results using the same modulation method as in the previous case. But we do not use this extended image segmentation for statistics calculation to avoid incorrect estimation caused by a large number of black pixels inside contours.



Figure 3.24: Color flooding driven by real contour (left) and by contour skeleton (right).

### 3.3.1 Skeletonization

Skeleton is another binary image entity taken from the terminology of the mathematical morphology (see [Serra93]). It could be defined informally as the union of circle centers which are located inside contour and herewith tangent the contour boundaries. They could be theoretically obtained by sequential *thinning* process using so called *hit-or-miss* transformation. This transformation is very similar to the erosion operator but it additionally preserves skeleton topology features. Iterative peeling is usually a very slow process and so several performance enhancing techniques has been developed. *Suzuki* purposed the sequential thinning algorithm based on the distance field transformation [Suzuki86] (see also [Zhou99]). *Kégl* developed a robust piecewise linear skeletonization algorithm based on principal curves (smooth curves which pass through the middle of a  $d$ -dimensional probability distribution) polished using several curvature penalty and vertex degradation rules [Kégl02].



Figure 3.25: Region enlargement into the contour: (from left to right) original grey-scale image with tight region inlet, coarse contour based segmentation, skeleton enlargement and gradient enlargement

We solve this problem using a slightly different approach. We do not need to know an exact skeleton geometry approximated by one-pixel tight curve but only enlargement of already segmented regions into the contour area that hides visible gaps between real region boundary and contour anti-aliasing (see *Figure 3.24*). This should be done for every contour pixel by simply finding the nearest region using growing circle mask (very similar to the distance filed transformation, see *Section 4.2.3*) which produce very similar results to that using segmentation based on skeleton boundaries. But as we can see on *Figure 3.25* the middle right picture, this approach is not as robust as we want to be. It will fail in cases when the image contains tight V-shaped inlets which should be classified as contour even if we select large  $\sigma$  for  $\mathbf{L} \circ \mathbf{G}$  filtering.

### 3.3.2 Gradient seek

To make enlargement more robust we purpose a novel region growing technique that is not based on looking for a minimal distance (which is definitely equivalent to skeletonization) but on the region highest intensity using image intensity gradient. This is inspired by the fact that intensity near the contour boundary usually upraises from dark contour pixels to brighter value similar to the region intensity median. If we follow this gradient slope we will indeed found the nearest region. On the other side it is clear that if we simply find

out only the nearest highest intensity we do not complete purposed task. We have to follow image gradient using 4-continous or 8-continous curve to make sure that we do not cross over deep intensity valley. This constraint is very similar to that we used in watersheds segmentation (see *Section 3.1.3*).

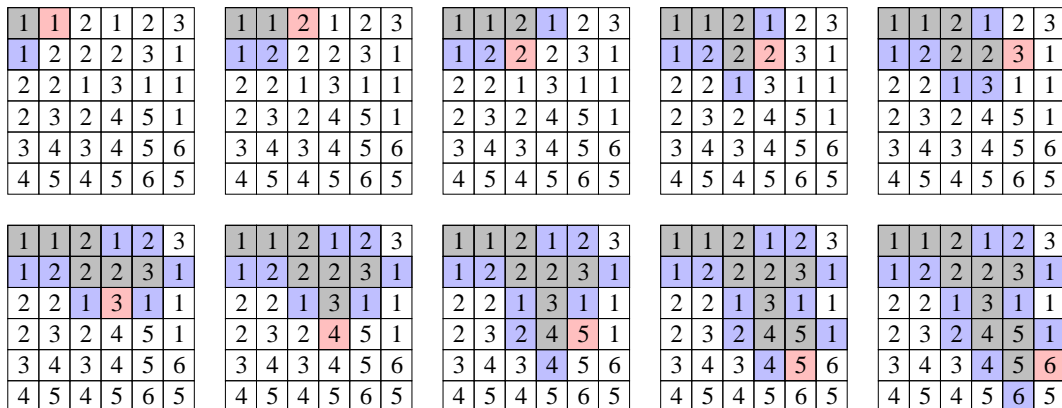


Figure 3.26: Gradient seeking in progress: visited pixels (grey), never visited pixels (white), pixels in priority queue (blue), pixel on the top of the priority queue (red).

For every contour pixel from the marker matrix  $M$  we call a simple function that finds out the marker of the first pixel from the nearest non-contour region that has to be connected with initial pixel via a 4-continous gradient curve. To solve this task we exploit priority queue of the already visited pixels in the original grey-scale image  $I$ . See following function source code:

```
MARKER *gradient_seek(int x, int y)
{
    PQueue *pixel=new PQueue(I);

    while (M[x][y]->id==CONTOUR) {

        pixel->Push(x+1,y);
        pixel->Push(x,y+1);
        pixel->Push(x-1,y);
        pixel->Push(x,y-1);

        pixel->Pop(&x,&y); }

    delete pixel; return &M[x][y];
}
```

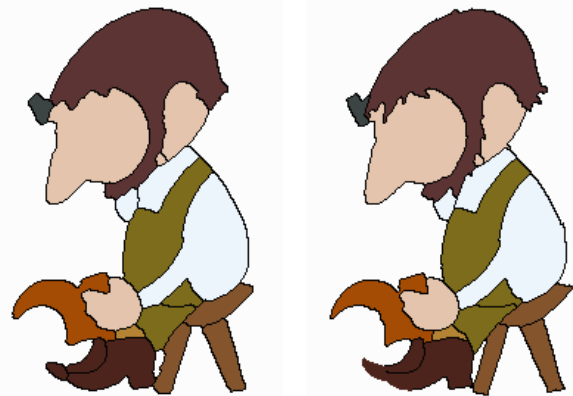


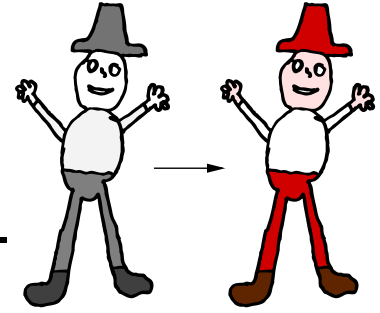
Figure 3.27: Final segmentation: based on contour skeleton (left) and on gradient seek (right).

Method `Push()` of `PQueue` class inserts the new pixel into the priority queue while preserving an ascending order of pixel intensities and at the same time it checks if the included pixel is not already in the queue. The second method `Pop()` takes out a pixel from the top of the priority queue and returns its coordinates. See *Figure 3.26* for gradient seeking algorithm in progress. Especially steps 5 and 6 are interesting. There, seeking process strided to the deep intensity valley which should be here a simulation of the noise artifact or any other type of local distortion.

While presented algorithm is based on a robust *backtracking* search technique which is able to make the round of local intensity peaks or valleys, it may introduce significant slowdowns for a large dataset. But if we consider that contours usually take place in approximately 5% of image pixels and the longest path toward a non-contour region is on average shorter than 10 pixels we conclude that this algorithm demands negligible computational overhead in front of sophisticated skeletonization methods presented above. While it also uses original grey-scale intensity image it is also much more precise in sense of the real contour shape in contrast to simple skeletonization algorithms based on binary image (see *Figure 3.24* the picture on the right side and *Figure 3.25* or *Figure 3.27* to compare results).

# 4

## Inking



While previous chapter was mainly about image analysis, this chapter will be focussed especially on synthesis techniques. We show how to apply user defined color to previously segmented grey-scale regions without losing original intensity texture and how to smoothly compose already inked foreground with new reconstructed background including synthesis of smooth foreground alpha masks. Also unsupervised dust spots and scratches removal or contour contrast enhancement techniques will be presented.

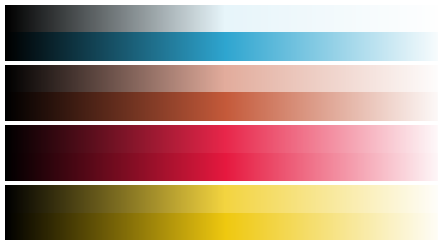
### 4.1 Color flooding

In this section we will discuss techniques that are responsible for the main procedure of whole inking process: transferring color to the grey-scale image. Final visual quality of resulting image depends mainly on two basic properties: selection of suitable color luminance modulation model and proper modulation technique.

One of the important data entity for whole inking process is already inked image with color examples produced by experienced artist using standard image editing software (see *Section 2.5*). User is able from this RGB image acquire necessary colors and place them into the own palette which consists of 32 name assigned colors (indices).

#### 4.1.1 Modulation using RGBI model

Each palette index has its own look-up table with precomputed *brightness shift* values of original RGB color. Brightness modulation is based on RGBI color luminance model which should be obtained by linear interpolation of original RGB color components using different brightness shift  $I$ , where  $I = 0$  is original RGBI color,  $I = -1$  clean black and  $I = 1$  clean white as follows:



$$\begin{aligned} R_I &= \begin{cases} (I + 1) \cdot R & I \in \langle -1, 0 \rangle \\ I \cdot (1 - R) + R & I \in \langle 0, 1 \rangle \end{cases} \\ G_I &= \begin{cases} (I + 1) \cdot G & I \in \langle -1, 0 \rangle \\ I \cdot (1 - G) + G & I \in \langle 0, 1 \rangle \end{cases} \\ B_I &= \begin{cases} (I + 1) \cdot B & I \in \langle -1, 0 \rangle \\ I \cdot (1 - B) + B & I \in \langle 0, 1 \rangle \end{cases} \end{aligned}$$

Figure 4.1: RGBI vs. HSB model.

If we use 8-bit representation for each color component the look-up table is indexed by integer value  $I \in \langle 0, 512 \rangle$ , where index  $I = 255$  corresponds to original RGB color.

The main reason why we introduce RGBI model is that we have to preserve original luminance of wanted RGB color and herewith modulate this luminance by grey-scale intensities inside the selected region to preserve original intensity noise texture, contour anti-aliasing and other typical surface features.

Luminance of user defined color usually does not corresponds to grey intensity median on the source image. Using e.g. HSB,  $l\alpha\beta$  or another color space that exclude luminance and color components (see [Wyszecki82] and [Pitas93] for survey) we will obtain image that has not the same color tinge as is presented on example image. See *Figure 4.1* where luminance of four different RGB colors is modulated from clean black to clean white using RGBI (above) and HSB (below) model. Original RGB color is in the middle of RGBI band. It is important that when luminance of user defined color is nearly white or black, the precise estimation of real color components (i.e. *hue* and *saturation*) is impossible due to loss of information.

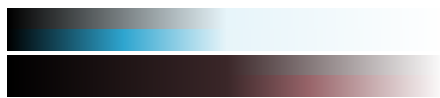


Compare also two grey-scale images on *Figure 4.2* where the left one is luminance bitmap derived using (2.1) from example image (on the left side) and the right one is original scanned image. They vary in intensities especially on boots, apron and stool. You should compare results obtained using luminance modulation via HSB model and RGBI model. It is not hard to see that as big is difference between grey-scale intensity and RGB luminance as different is resulting color when we use modulation based on HSB model.



*Figure 4.2:* HSB vs. RGBI model: (from left to right) image with example colors, only luminance of example colors, original grey-scale image, color luminance modulation using HSB model and RGBI model.

The main advantage of RGBI model is that we could easily set centre of the brightness shift ( $I = 0$ ) to be in correspondence with the region intensity median and final color luminance will be modulated by deviations in original grey-scale image using simple equation  $I = I_g - \tilde{I}$ , where  $I$  is brightness shift,  $I_g$  original grey-scale intensity and  $\tilde{I}$  intensity median of corresponding grey-scale image region. We should also analogically define the same intensity median adjustment using HSB values estimated from user defined RGB color to obtain HSBI model as follows:



*Figure 4.3:* RGBI vs. HSBI model.

$$\begin{aligned}
 H_I &= H \\
 S_I &= S \\
 B_I &= \begin{cases} (I + 1) \cdot B & I \in \langle -1, 0 \rangle \\ I \cdot (1 - B) + B & I \in \langle 0, 1 \rangle \end{cases}
 \end{aligned}$$

But as we can see on *Figure 4.3* using HSBI model we compute brightness shift table which significantly differs in expected color tinge due to inexact estimation of hue and saturation components during color conversion between RGB and HSB models. There is no reason to convert original RGB values to different color space if we could use red, green and blue components directly.

#### 4.1.2 Luminance correction

If the intensity median of proper region is similar to the user defined color luminance the results are acceptable. But when we want to apply really bright color on originally dark region, we introduce visible step change of color luminance (see *Figure 4.4* in the middle). It is possible to perform linear correction of pixel intensities using simple equation  $I = (I_g - \tilde{I}) / (L - \tilde{I})$ , where additionally  $L$  is requested luminance of resulting color, in other words the new intensity median.



*Figure 4.4:* Grey-scale intensity vs. user defined color luminance: (from left to right) original grey-scale image, user defined luminance, color luminance step, linear correction, non-linear correction (white noise suppression).

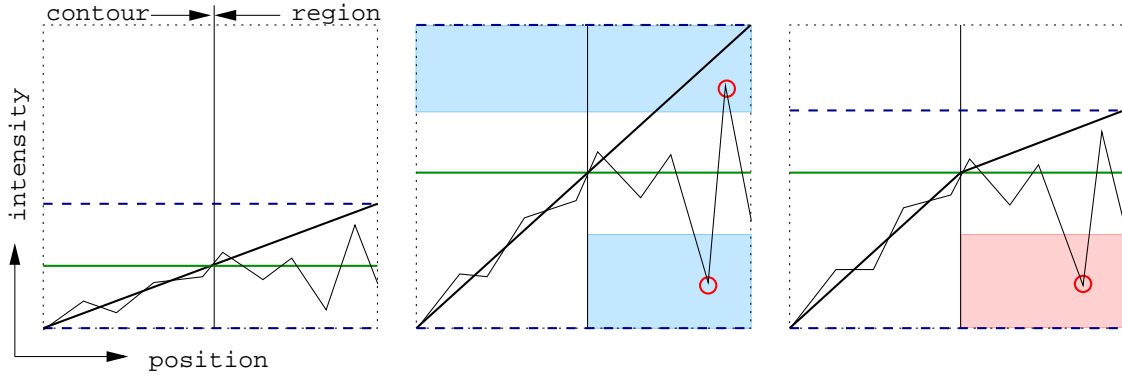


Figure 4.5: Intensity corrections: original intensities (left), linear correction (middle), non-linear correction (right). Green line represents intensity median and blue or red regions forbidden intensity values estimated using intensity dispersion bounds (blue dashed lines).

This correction help us to reach user defined intensity median and make the spatial crossing from contour to region area to be visually consistent, but unfortunately as we can see on *Figure 4.4* (second picture from the right side) it introduces white and black noise overlapping. The main reason is visualised on *Figure 4.5*. Original intensity dispersion is scaled towards wide intensity interval which is more visually significant than original one (see blue dashed lines on *Figure 4.5*). Red circles show us noise values inside forbidden areas that tend to be emphasized after intensity correction. If we want to preserve original intensity dispersion above the new intensity median and remove the white noise, we have to perform non-linear intensity correction (*Figure 4.5* and *Figure 4.4* on the right side).

Global white noise suppression by non-linear intensity correction has still no feasibility to remove black noise which is also disturbing especially within the sequence of similar images. To produce same image as we can see on *Figure 4.6* (right picture) we have to apply smooth combination of local non-linear correction and simple luminance modulation. Only pixels near the contours have to be rescaled to new values. Pixel intensities inside the region will modulate final color luminance using original equation  $I = I_g - \tilde{I}$ . To make smooth transition between both modulation methods we estimate Gaussian expansion of region contours. Each contour pixel raise local incremental image addition using values from fixed size matrix containing discreet approximation of Gaussian hat with user defined standard deviation. Pertinent saturation will be truncated to maximal value. Resulting layer on *Figure 4.6* (left picture) where dark blue pixels represents original contours and grey-scale ones local weights for two types of luminance correction methods. So the final formulae that estimates proper brightness shift will be

$$I = \alpha \cdot (I_g - \tilde{I}) + (1 - \alpha) \frac{I_g - \tilde{I}}{L - \tilde{I}}, \quad (4.1)$$

where  $\alpha$  is weight of pixel taken from Gaussian expansion of contour bitmap.

Crucial value in (4.1) is region intensity median  $\tilde{I}$ . Although the median statistic is much more robust than simple arithmetic average of region intensities in sense of noise suppression, it is usually hard to estimate true median statistic when the size of region is very small. Region undersampling causes that lots of region pixels lay inside contour anti-aliasing and their actual intensities are degraded due to averaging with neighbour dark contour pixels. Median statistics usually produce significantly lower value of  $\tilde{I}$  comparing it with the same median value estimated using pixel intensities from spatially large region. Smaller  $\tilde{I}$  value cause

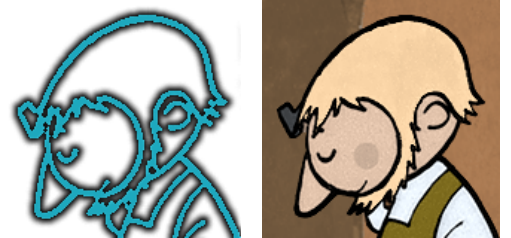


Figure 4.6: Gaussian expansion (left) and final luminance modulation (right).



Figure 4.7: Inaccurate region intensity median estimation due to its small size.

higher brightness shift hence region will be inked by brighter color which should be sometimes visually really disturbing (see *Figure 4.7* on the left side).

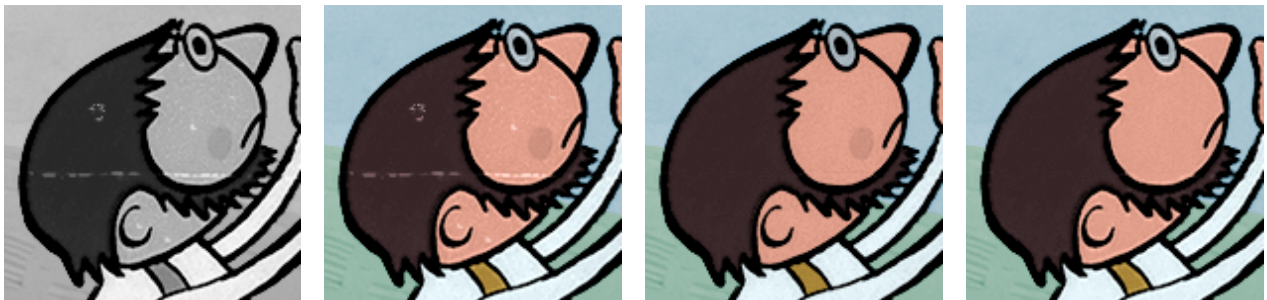
To correct this artifact we should compute weighted average (using region size as weight) of medians from all large enough regions marked with the same color index and use this value instead of real median statistic. To decide if the region is small or large we have to use fuzzy transition to avoid step changes in median estimation modes. We let user to tune this fuzzy transition for best performance but it was experimentally proved that regions with size smaller than 16 pixels are completely small and above 64 pixels are large enough to produce reliable intensity median value. Between these two values we apply simple linear combination of both methods. See *Figure 4.7* on the right side where this technique was used.

The another problem rises when the small region is only the one with specific color index in the whole image or all regions with some color index are small (e.g. occluded cobbler's hammer or open eyes). We should compute median statistic using pixel intensities from all small regions and if number of pixels is still not enough to reach upper bound of fuzzy transition we simply treat the maximum intensity as median. It is also helpful to compute median statistics only from pixels that has intensity above intensity median of contours.

### 4.1.3 Unsupervised dust spots removal

Exploiting the region homogeneity feature actually represented by intensity median and by small dispersion due to black and white noise, we could introduce simple but effective dust spots removal technique which saves lots of retouching work during post-production phase.

The main idea is based on fact that if some region pixel has intensity out of the standard deviation interval we should identify it as white or black impulsive distortion and reconstruct this value by selecting random intensity using median centered gaussian distribution with standard deviation same as in current region. Notwithstanding that this algorithm is really simple, results are surprisingly successful (see *Figure 4.8* on the right side).



*Figure 4.8*: Unsupervised dust spots removal algorithm: (from left to right) grey-scale image, inking without dust spots removal, white dust spots removal, black and white dust spots removal.

We could use this technique only when pixels are not located inside Gaussian expansion of contours. Otherwise several advanced detection and non-linear filtering methods have to be used e.g. local median filtering or high frequency preserving reconstruction using two-dimensional FFT (as we saw in *Section 2.4.4*) to preserve original edge shape.

Sometimes it is reasonable to use only white dust spots removal. While black spots are really rare due to circumstances posted in *Section 2.4.4*. This restriction preserves several darker details in region intensity texture e.g. cheek on *Figure 4.8* and allow us to apply removal technique also inside Gaussian expansion but without big expectation on quality of reconstruction.

## 4.2 Background and foreground composition

If we decide to use *reconstructed background* inking technology we have to solve problem of composition of two background and foreground layers to produce final image and foreground alpha channel to exclude both layers in post-production phase (see *Figure 4.9*). While new background should differ especially in local texture and luminance properties we should not perform simple image difference to exclude foreground layer. We exploit boundaries of already segmented regions and apply algorithm which is very similar to natural image matting technique.

### 4.2.1 Digital matting

Main task of digital matting is to estimate an opacity for each foreground pixel and produce layer which is also known as alpha channel. Digital matting is usually heavy underconstrained problem especially in *natural image* version when background has no special properties. Nevertheless problem of smooth extraction of the foreground parts from already composed RGB image is well studied and several techniques has been developed. The most popular method used especially in TV broadcasting systems also known as blue screen matting is based on well tuned digital or analog chroma keys.



Figure 4.9: Image composition (right) of already inked foreground (left) with new reconstructed background (second from left) using smooth foreground alpha channel (second from right).

Ground truth alpha masks should be obtained by technique of several images with the same foreground and different backgrounds using *SVD* computational framework (see [Blinn96]). Methods that solve natural matting with only one image are based on segmentation which divide image to three regions: definitely background, definitely foreground and unknown. Then unknown are divide to several sub-segments (box or circle shaped) and local samples of color distributions (represented as mixture of Gaussians) from nearest background and foreground pixels are paired together using specific constraints. From this network of pairs the alpha is estimated by intermediate distribution for which observed color has maximum probability [Tomas100] or better using *Bayesian* framework [Chuang02].

Natural grey-scale image matting is more complicated task because of missing invaluable color information. But our case with bold contours is trivial comparing it with complicated images containing e.g. hairs or fur. Originally complex problem is reduced to easy decision which of background pixels are interfering with contour boundary anti-aliasing. Grey-scale intensities of these pixels directly determine requested alpha values. We exploit color luminance from reconstructed background and compare it with intensity of current processed pixel. If grey-scale intensity is lower, it probably represents anti-aliased boundary pixel and we have to modulate background color luminance using magnitude of this difference multiplied by weight taken from the Gaussian contour expansion (see Figure 4.6) to avoid influence of distant pixels. Side effect of this process is on-line generation of alpha channel for post-production purposes (see Figure 4.10 on the right).

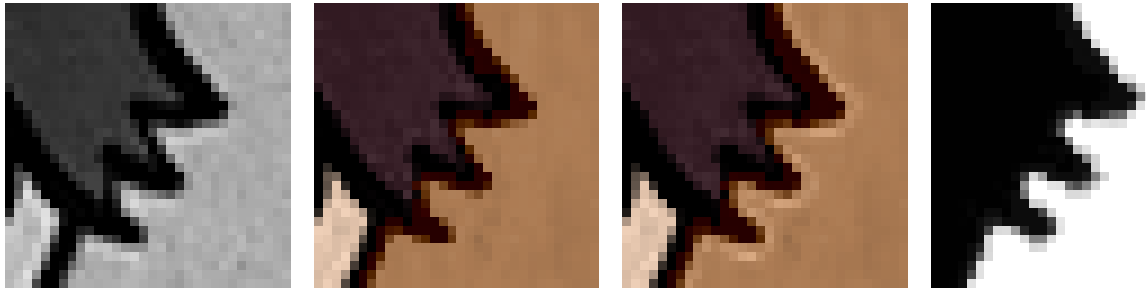


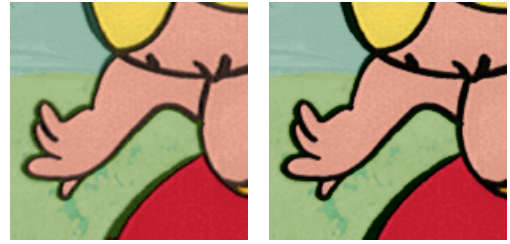
Figure 4.10: Details of foreground and background compositions: (from left to right) original grey-scale image, only dark pixels composite, all neighbor pixels composite, final foreground alpha channel.

See detail image on Figure 4.10 (second picture from the left side) to consult accuracy of this simple method. Although several anti-aliasing pixels are lost, we could conclude, that this technique is suitable for our requirements because it produces visually acceptable results as we could see on Figure 4.9 or other on already composed images in this thesis. It is also possible to extend purposed approach using modulation even if grey-scale intensity is higher than background color luminance to produce more accurate matting. Unfortunately when luminance of color in reconstructed background is much higher than intensity in original grey-scale image, this extension introduces white “halo” effect which visually enhance sharpness of the boundary edge (see Figure 4.10 second picture from the right side), so it does not achieves as good results as in previous method.



### 4.2.2 Contour contrast correction

Until now we assume that contours has significantly darker intensities comparing it with other image regions. If not so, we have to satisfy this assumption by correction of contour contrast, because dark contour intensities visually suppress the step transition between two different region colors (see *Figure 4.12*, left picture). It was already posted in *Section 2.4.7* that low contrasted contours are slid off to clean black intensity using local gamma correction. Local means that we apply standard histogram equalization weighted by Gaussian expansion of contours (see *Figure 4.6*). We will precompute one look-up table for intensities  $i \in \langle 0, 255 \rangle$  and user defined  $\gamma \in (0, 1)$  using simple equation  $\text{lut}(i) = 255 \cdot (i/255)^\gamma$ . Final pixel intensity  $I_f$  is estimated by convex combination:  $I_f = \alpha \cdot \text{lut}(I) + (1 - \alpha) \cdot I$ , where  $I$  is original intensity and  $\alpha$  pixel weight from Gaussian expansion. See *Figure 4.11* where local gamma correction with  $\gamma = 2$  was performed.

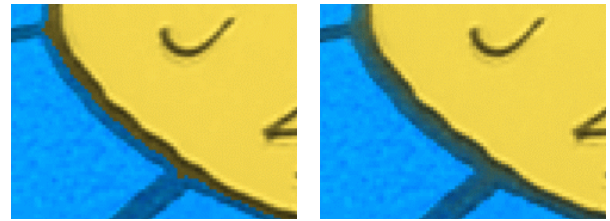


*Figure 4.11*: Contour contrast correction: before (left) and after (right) correction.

### 4.2.3 Smooth color transition

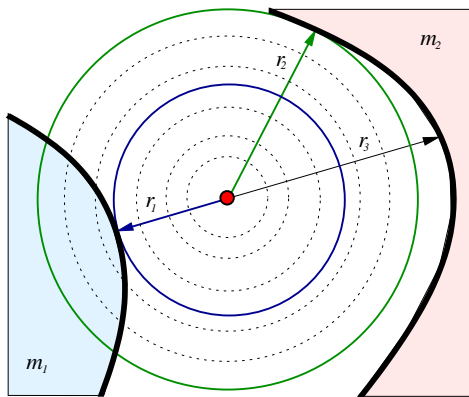
Unfortunately there are some special objects with bright colored regions and contours e.g. sun, where darker contour intensities could corrupt original visual brightness of whole object. If we do not perform contour contrast correction, we introduce visible step changes between background and foreground colors as we already mentioned in previous section. To solve this quandary we develop simple algorithm based on distant filed technique which produces visually good-looking smooth color transition.

Distant field is matrix with same dimensions as corresponding binary image (also in 3D) where each element contains number that represents minimal distant (in certain length metrics) to nearest object in binary image. Unfortunately exact computation of distant filed using *Euclidean* metrics should be obtained only by brute force algorithm which is very expensive even for two-dimensional images. Several speed-up approximation techniques has been developed. Most of them are based on the special distant filed transformations (for survey see [Borgefors86]). However if we consider that no full frame distant filed is needed, we should not take care about processing speed. We compute only small portion of this filed which corresponds to the contour pixels (usually less than 1% of whole image) using brute force algorithm.



*Figure 4.12*: Color transition inside low contrasted contours: step (left), smooth (right).

Our algorithm is based on growing circle method (see *Figure 4.13*). For each contour pixel we start drawing virtual circle using classical integer *Bresenham's* algorithm. Instead of putting pixel we perform test if on the same coordinates resides contour or region pixel. To obtain smallest distance  $r_1$  to nearest region we grow circle until first region pixel with marker  $m_1$  is found. Next we continue with circle growing until we found radius  $r_2$  where certain circle pixel tangents another region with marker  $m_2 \neq m_1$ . Using  $r_1$  and  $r_2$  values we compute pixel color as follows:  $C = (r_2 C_1 + r_1 C_2) / (r_1 + r_2)$ , where  $C$  is red, green or blue component of final RGB color and  $C_1, C_2$  the same components taken from colors in regions marked by  $m_1$  and  $m_2$  respectively. You can see on *Figure 4.13* that this method does not produce correct Euclidean interpolation which have to be done using linear extrapolation of tangent vector  $r_1$  towards opposite boundary of contour

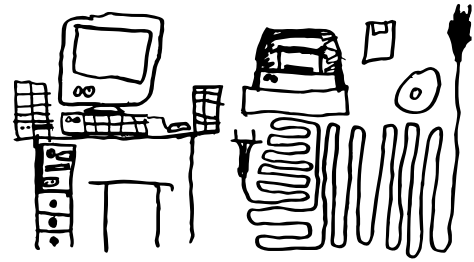


*Figure 4.13*: Distant field estimation by growing circle algorithm.

with length  $r_3$ . This inaccuracy is not perceptible, moreover it help us to suppress interpolation artifacts inside bold contour T-junctions (see *Figure 4.12*) where linear extrapolation of several tangents going to be very long.

# 5

## Application



This chapter will be mainly focussed on implementation details and several user interface features available in semi-automatic inking application called COL. This application realizes practical usage of yet described analysis and synthesis techniques, especially their incorporation in to the inking pipeline. Detailed description of important automatic prediction and speed up tools will be included. They help user to effectively solve tasks full of featureless and repetitive correction work. Moreover this chapter should be also used as helpful addendum for user reference manual because of augmented description of specific operations.

### 5.1 Technical specifications

The COL is written in ANSI C/C++ language as full portable application. Using free ports of GNU C compiler it is possible to produce support for both *Linux (X11)* and *Microsoft Windows 9x/2k/XP (DirectX)* platforms. Optimal processing speed is accomplished by exploiting advanced low-level code optimization features of *GCC* compiler and using own intelligent memory management mechanism supplied by memory and permanent disk caching techniques.

#### 5.1.1 System requirements

To reach interactive performance limit it is recommended to use 750MHz and faster CPU with more than 64MB of free memory and with disk/memory data transfer speed above 15MB/s. Afterwards the application response time after user intervention is not longer than one second in worst case. The minimal supported and same the best output resolution is 1024x768x32bit. Lower bitrates or higher resolutions are not recommended due to visual inconvenience. For ergonomic purposes the high frequency CRT display with diagonal length above 17 inches have to be used.

If we want to work on several episode cuts using the same standalone workstation, huge HDD drive have to be installed. Merely source data (new reconstructed background and original grey-scale image) will take place approximate of 1MB/frame, permanent application disk cache space is 3MB/frame and final output color frame and foreground mask consume ensemble 2MB, all in all this takes 6MB per one frame. If we consider that cuts with background movement are rare hence only one static reconstructed background frame is needed for whole sequence. We should conclude that in average storage space per one frame is 5MB. This amount 60GB of free disk space for whole episode.



Figure 5.1: Working desktop of COL application.

#### 5.1.2 User interface

Application working desktop is divided into the four important parts (see *Figure 5.1*). Left information panel were active color marker and marker of region located under mouse pointer are displayed, information switch lights of activated layers (see *Section 5.2*) and other switch flags take place also inside this panel. Next important part is the top status line where several active parameters are displayed. Right panel contains

user defined palette of 32 active color makers with user defined names. Each marker has its own RGB color taken from color example image. Active layers are displayed in the central part of whole desktop. It is possible to magnify layer details by smooth zoom (up to the 8x magnification) or by small 4x zoom window that follows mouse movement (see *Figure 5.2*).

Interaction between user and COL application is strictly based on operations handled by mouse or keyboard short cuts (*hot-keys*). Moreover mouse strokes and keyboard actions could be recorded into ten independent macro clipboards. There is also special mode available where all operations are stored to action buffer without any slowdown caused by refreshing computation and they could be performed all at once later. This function allow experienced user to reach minimal interaction delay, however it put novice user to the inconvenience during learning phase. Although list of basic hot-keys is not short, it was proved that adroit novice operator will learn them usually after no more than two hours of active application usage.

Project management is driven by configuration `.ini` files using standard readable text format. Each configuration file contains absolute path to proper episode cut directory and file name masks for enumerated source and output files. Frame number bounds are determined automatically during initialization phase. Also important parameters are included: starting frame, actual frame, default values of basic thresholds, layer activation flags and RGB palette of 32 name assigned colors.

Another important application feature is *journaling* mechanism. All error messages and user interactions are recorded including time of activation. During whole session they are continuously written in to the special `.log` file with readable text format. Using this journaling mechanism it is possible to analyze overall efficiency of user work as we will see in the next chapter.



*Figure 5.2*: Desktop zoom (top) and windowed zoom (bottom).

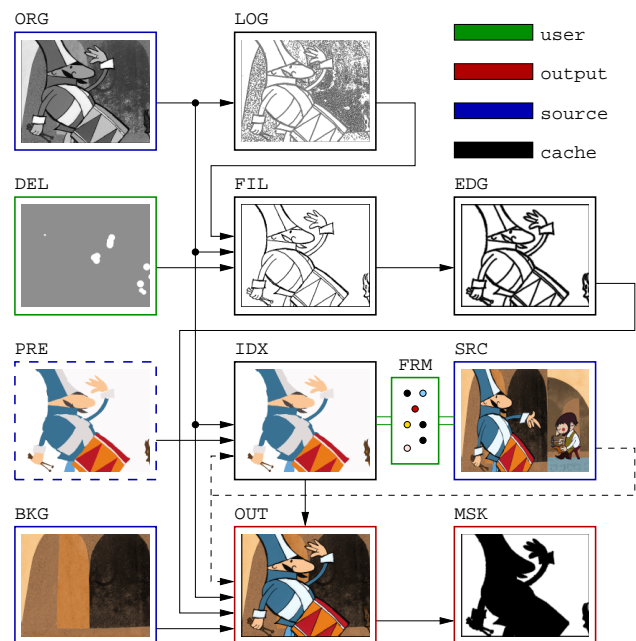
## 5.2 Layers

Data flow in the COL application is divided in to the several independent layers. Each layer is represented by memory bitmap with same dimensions as original grey-scale image (720x576 for PAL images) varies in type of information stored inside pixels and in features of associated user driven modification parameters. In this section we will describe properties of application layers in detail.

### 5.2.1 Layer caching techniques

Application layers form hierarchy of dependencies which should be represented by acyclic oriented graph (see *Figure 5.3*). Each layer has its own sources and consumers. We exploited this structure to speed up application interaction response time by memory caching technique where local modification inside specific layer causes corresponding refreshing calculation only inside its consumers. The memory caching is supplied by disk mirroring mechanism where yet computed layers are stored on HDD in specific directories. Disk cache provides us to refresh content of the memory cache whenever user changed the actual frame number.

For backup purposes the source layers (blue frame on *Figure 5.3*) and layers with information about user intervention (green frame) are the most important. Output (red frame) and cache (black frame) layers could



*Figure 5.3*: Data flow diagram of COL application.

be regenerated from source and user layers using automatic refreshing application mode where all frames are over again rendered. Disk cache is also used as fast source of information for prediction techniques (see *Section 5.3*).

### 5.2.2 Source layers (ORG, BKG and SRC)

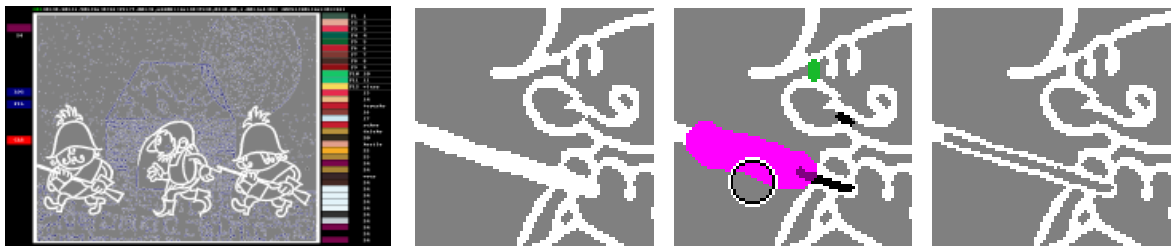
The lowest source layer is original grey-scale image. In COL application known as ORG layer. See *Figure 5.3* where ORG layer has blue frame as same as another read only source layers: BKG and SRC. It is stored (as we mentioned in *Section 2.1*) in 8-bit grey-scale read only PiNG file. If we change actual frame number, ORG layer have to be read again to memory cache from proper image file. New reconstructed background layer BKG is stored in 24-bit true-color PiNG file. If image sequence contains dynamic background we have to load always corresponding BKG layer together with ORG frame. But usually it is possible to load only one BKG layer during initialization phase while most of episode cuts has usually static background. SRC layer contains yet inked image with color examples and is loaded only once during initialization phase. User is able to switch on SRC layer and transfer wanted RGB color to actual palette marker using mouse.

### 5.2.3 Laplacian of Gaussian layer (LOG)

For efficient memory and disk caching is LOG the most important layer in the inking pipeline. It contains zero crossings and  $L \circ G$ -negative pixels (see *Section 3.2.1*). If we consider that standard deviation parameter  $\sigma$  is user driven, then it is not hard to see, that change of this value rises the longest refreshing procedure which could be activated in the COL application. This is because of LOG is the second lowest layer in dependance hierarchy after ORG layer (see *Figure 5.3*). Moreover performance of  $L \circ G$  filtering spend in average half of CPU time needed to produce final color image, if we do not consider any layer caching using only original sources. While it is much more faster to load LOG from disk than to compute it again, the layer is stored in permanent disk cache as indexed PCX file to reach best storage size vs. save/load time ratio.

### 5.2.4 Contour detector layers (FIL, DEL and EDG)

The FIL layer contains binary bitmap of image contours derived from LOG using adaptive unsupervised contour filling algorithm followed by morphological erosion and contour authentication mechanism (see *Section 3.2.2*). User has available special retouching layer DEL where he is able to make following corrections (see *Figure 5.4* for reference): add contour (green), erase contour (black), retrieve original contour or locally apply intensity threshold to create new region inside bold contour (pink). These corrections are done using circular brush with user defined radius. It is also possible to draw retouching lines or raise flood-fill algorithm to fill any  $L \circ G$ -negative area with contour or background color. Considering the fact that retouching work is usually very similar in adjoining frames we could exploit the disk cache to copy yet created DEL layers into the clean retouching layer of unvisited frames.



*Figure 5.4:* Application desktop with two activated layers: LOG and FIL. Detail of unmodified FIL layer (left), available retouching brushes inside DEL layer (middle), FIL layer after correction (right).

After correction the FIL layer is ready for the Gaussian expansion (see *Section 4.1.2* and *Figure 4.6*) which is produced and stored inside the EDG layer. Although the standard deviation parameter  $\sigma$  of the Gaussian expansion is user driven, the EDG layer is not directly visible inside application desktop. FIL, DEL and EDG layers are stored in disk cache using indexed PCX. While DEL layer contains information about user intervention, it has to be stored in user directory for further backup purposes.

### 5.2.5 Region marking layers (IDX and FRM)

Layer IDX is produced from corrected FIL layer by fast region marking technique followed by gradient growing algorithm (see *Section 3.2.3* and *Section 3.3*). Each pixel of IDX layer contains color index from user



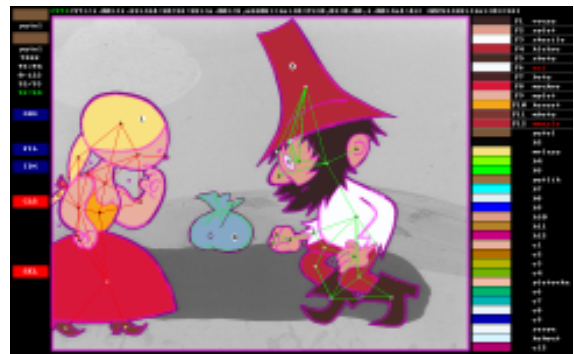
defined palette. From standalone IDX layer is not possible to distinct real regions because they are actually represented by virtual table of active regions which is also accessible via internal region pointer matrix (with same dimensions as IDX layer). Only memory cache is used to store this region table and pointer matrix because it is much more faster to compute these structures again instead of load them from disk cache. We have to store only indexed IDX layer for prediction purposes (see *Section 5.3*).

Proper assignment of color indices to image regions is task that badly wants human intervention even in cases when all prediction techniques failed or no prediction layer is available. The application user is able to select wanted color index using mouse or keyboard and when the mouse pointer is near or inside the region which has to be inked by this selected color, he presses the mouse button. This action places out color index marker which apply selected color to the proper region. There is also special type of marker that moves region from foreground to background. It is necessary when the region size threshold mechanism is not able to distinct foreground and background regions correctly.

All markers are stored inside layer FRM which is in fact represented by linked list of structures that contain image coordinates and type of marker. This list of markers together with another frame specific parameters e.g. user defined thresholds, type of prediction, prediction frame, etc. are stored in compressed file (using LZ77 compression) inside user directory where also DEL layers are located.

Another helpful tool is region connectivity graph (see *Figure 3.1*). This graph allow user to select e.g. single character to be in specific region group using one region group marker. There are seven region groups available and initially all regions are stored inside group zero. This division is important for increasing success of prediction techniques as we will is in *Section 5.3*.

Graph building algorithm is very similar to the technique of region marking presented in *Section 3.2.3*. We perform simple test on each element of region matrix: if left and/or bottom neighbor element points to different region then we examine into neighborhood list of this region and if new region address is not present here we include it. It is also important to include reciprocal link to the neighborhood list which belongs to the left and/or bottom region. Thereby created structure of neighborhood lists form final graph structure which could be easily traversed using simple *Dijkstra's* algorithm.



*Figure 3.1:* Color markers and region connectivity graphs of three different region groups.

### 5.2.6 Output layers (OUT and MSK)

Final output layer OUT and foreground mask layer MSK are synthesised by algorithms described in previous chapter. Visual quality of output image could be modified by following parameters: local gamma correction of contour intensities, local switching of unsupervised dust spots removal algorithm, optional smooth color transition inside contour regions and varying with standard deviation of the Gaussian expansion.

If we consult *Figure 5.3* we could observe that final color flooding exploits almost all available application layers. But the memory caching mechanism allow user to make the most common correction work (placing proper color indices into the IDX layer while OUT layer is also activated) even with realtime refreshing speed, because OUT layer refresh overhead is significantly smaller comparing it with time required to make refresh of the all active layers inside inking pipeline.

## 5.3 Prediction

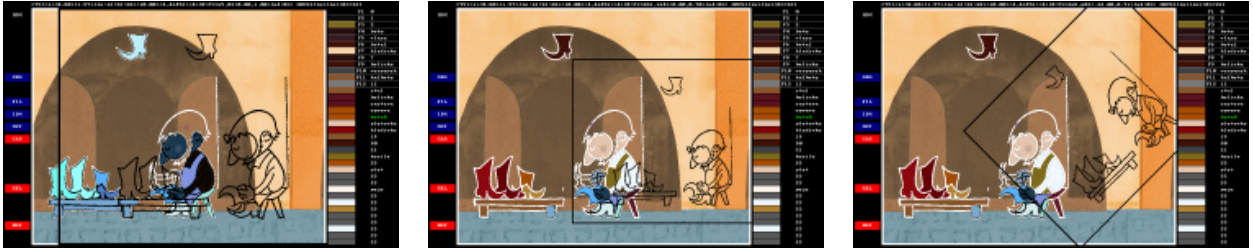
Crucial part in the whole inking process is correct color assignment to reach the same color placement which was designated by experienced artist. This task falls into the field of pattern recognition. Unfortunately recent research in this domain is mainly focussed on scenes from real world with important geometrical features like topology consistence, object solidity, etc. But cartoons objects are usually composed as abstractions of real world entities where the most of the useful geometry properties are not preserved. Cartoon characters are elastic and their topology features vary even between neighbour animation frames due to artists mistakes (see e.g. *Figure 3.23*). Sometimes it is really hard or totally impossible even for human to decide for example if proper region appertain to the background or foreground part of image.

As luck would have it, we are able to exploit two important features of cartoons with bold contours and solid foreground regions: spatial coherency and the frequent phenomenon that different colors usually corresponds to the different grey intensity levels. We call these two simple heuristics as *position* and *intensity* prediction respectively. In this section we describe both methods in detail.

### 5.3.1 Prediction frames

For both prediction methods is important the selection of prediction frames. It is useful to select one special frame to be a starting point for whole inking process. This frame need not to be the first frame of the sequence, better the frame where all foreground parts are visible (if it is possible) ought to be selected.

For the starting frame is no prediction information available, so we have to perform complete user driven marking and retouching work. Afterwards we could continue forwards or backwards in frame order and set previously stored IDX layers to be the prediction PRE layers (see *Figure 5.3*) for actual frame.



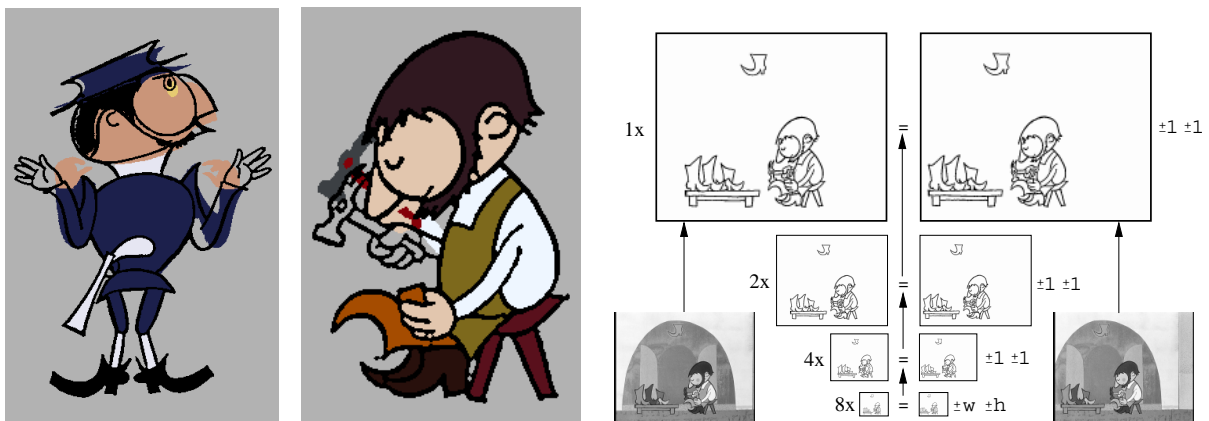
*Figure 5.5*: Examples of affine transformations available for user driven fitting of prediction layer (PRE): translation (left), scale (middle), rotation (right).

The main advantage of this technique is that user is able to set prediction frame to fit the animation loops while COL automatically increase the prediction frame number, the further loop passes will be correctly predicted without any user intervention. And more if we allow user to set different prediction method or even frame number for each region group then we give him the chance to increase the success of prediction.

It is also clear that correct inking depends on right frame order due to data dependence of prediction layers. This circumstances cause that random frame access is not recommended or better completely forbidden otherwise we introduce cache inconsistency in particular if we want to perform far jump to the not yet visited frame.

### 5.3.2 Position prediction

Position prediction assumes that if we simply superpose actual IDX and prediction layer PRE from the current image sequence (loaded from disk cache using formerly saved IDX layer with same number as the user defined prediction frame), then the large regions or regions that do not change its relative position usually overlap each other (see *Figure 5.6* on the left side). Moreover we allow user to adjust layer's geometry using simple affine transformations like translation, scale and rotation (see *Figure 5.5*) and if we compute the largest spatial intersection of actual regions and regions from transformed prediction layer, then we are able to estimate proper region color using color indices from prediction layer.



*Figure 5.6*: Principle of position prediction (left) and foreground translation shift retrieval (right).

While the most common affine transformation is translation, the COL application provides (on demand) simple and fast tracking algorithm that is able to estimate relative position shift between PRE and IDX layer with pixel precision by brute force retrieval of the global minimal difference of two shifted grey-scale bitmaps accelerated by spatial decimation pyramid on contour domain. While images on contour domain (FIL layer) are insensitive to background movement, purposed algorithm is more robust comparing it with classical grey-scale block matching techniques. Spatial decimation is computed by simple averaging using 8-bit precision and allow us to compute brute force shift retrieval using only 8x shrunk image where original number of pixels is reduced 64x and the final horizontal and vertical shift is estimated with accuracy of  $\pm 4$  pixels. This initial inaccuracy is refined up to the pixel precision by  $\pm 1$  shift retrieval using decimated layers with 4x, 2x and 1x reduced image resolution (see *Figure 5.7* on the right side).

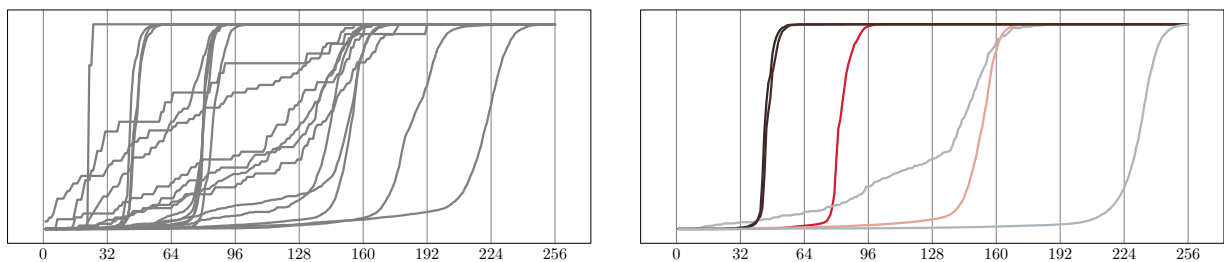
### 5.3.3 Intensity prediction

Sometimes is spatial coherency insufficient to reach successful prediction results. See *Figure 5.8* where change between actual (far right) and prediction frame (far left) is not small enough to be suitable for position prediction. But there it is well visible that the most of different colors correspond to different intensities in original grey-scale images and it is possible to use only information about region intensity to select proper color index.



*Figure 5.9*: Comparison of prediction hit rate: (from left to right) grey-scale prediction frame, correctly inked prediction frame, position, intensity and combined prediction, correctly inked and actual grey-scale frame.

We could simply compare intensity medians of regions from prediction frame with medians from actual frame and estimate proper color indices by minimal difference retrieval but we are able to increase robustness of intensity based prediction via accumulated discrete distribution functions of region intensities. We simply accumulate intensity histograms of regions that were marked with same color index and then perform numerical integration to obtain final discrete distribution function on grey-scale domain. The same process we apply on histograms of regions from actual frame and then perform global minimal difference retrieval to find out the best fitting distribution from prediction frame.



*Figure 5.10*: Principle of intensity prediction: retrieval of the best match between intensity distribution functions of regions from actual frame (left) and from prediction frame (right).

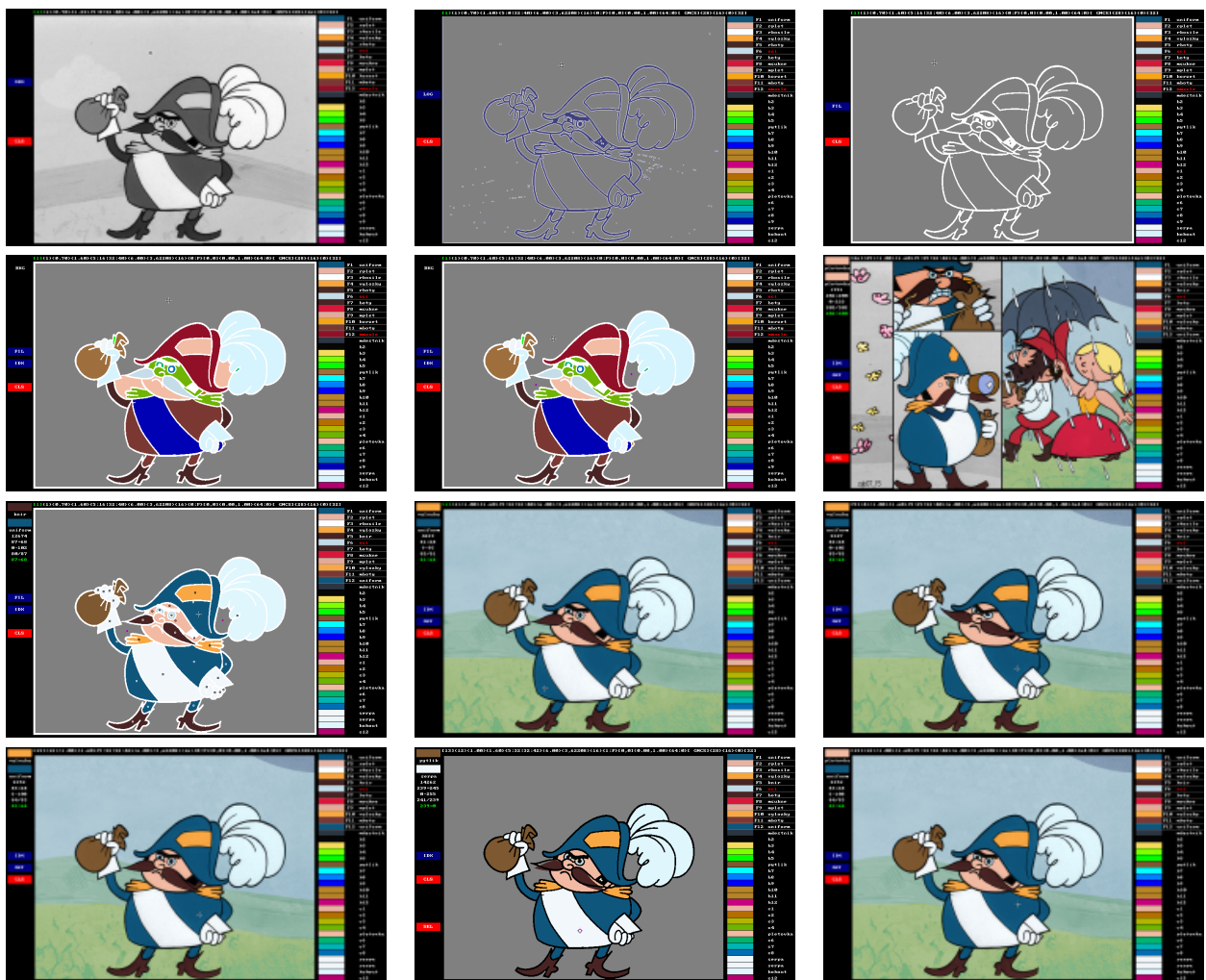
For our example on *Figure 5.9* the distribution functions are shown on *Figure 5.10* (right graph). It is important to see that e.g. two different color indices has similar color distribution (boots and hairs) and distribution of eye's pixel intensities is degraded by noise due to insufficient region size. In fact this degraded distribution have to be actually similar to the distribution of skin regions because eyes has the same intensity median. On the left side of *Figure 5.10* we could see intensity distributions of several regions from actual image. Some of them are also degraded by noise due to small number of pixel samples.

Result of retrieval is visible on *Figure 5.9* in the middle. Intensity prediction in contrast to position prediction is unable to distinct foreground and background parts. All background region with size under

background region size threshold are detected incorrectly. This problem should be partially eliminated by combined prediction (also presented on *Figure 5.10*) where background regions are predicted using position and other using intensity.

If we compare the results of intensity prediction with position prediction we should conclude that they look like much better. The main problem is hidden inside small regions with degraded distribution and in colors of boots and hairs, because these colors has similar hue, saturation and intensity we are not able to visually distinct the differences but due to similar intensity distributions the false detection is impending. This problem is general and decrease success of intensity prediction method in cases when regions with same intensity have assigned different colors.

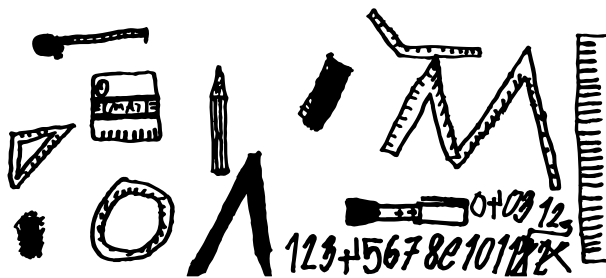
Generally it is up on user to select the best prediction method. He has to give due weight to the intensity of further manual corrections. It was proved that best behavior in common cases reports position prediction because it is able to correctly distinct background regions and also estimate proper color for small regions. But sometimes intensity prediction is invaluable even when we exploit region grouping tools (see *Section 5.2.5*). We eliminate the problem of different colors for the same intensity distributions by performing intensity prediction only inside small region groups which belong to the one object inside the scene where the intensity collision between objects is present.



*Figure 5.9:* Example of inking in progress using COL application (from the left top to the right bottom): ORG layer overview, fitting scale using  $\sigma$  of  $L \circ G$  filter (LOG), unsupervised contour detector (FIL), initial color index assignment (IDX), background regions removal, color acquisition using example layer (SRC), correct color assignment (IDX), final inked background composite (COL), position prediction on the next frame, prediction failed on another frame, animation loop fitting using prediction frame setup (PRE), correct prediction.

# 6

## Experiment



We try to determine application efficiency using performance statistics of user interaction by analysis of journal files taken from scenes vary in overall difficulty which were inked by experienced application users. Further analysis of these statistics allow us to make better conception of real difficulty of inking process.

### 6.1 Methodology

Measured shots are sorted by human assigned difficulty which was evaluated by five levels using following adjectives: *trivial*, *easy*, *intermediate*, *laborious* and *hard*. Also short scene overview will be presented including five illustrative frames selected from whole sequence where the main shot difficulties are depicted.

It is usually uneasy to estimate real scene difficulty ahead. Only experienced user of COL application is able to predict this important characteristic. Rough approximation would be obtained by relative ratio of total number of frames and compressed file size using any of widely used entropy sensitive video encoders based on variable bitrate encoding.

#### 6.1.1 Measured statistics

In this section we will present detail description of measured statistics that allow us to better compare scene difficulty and to estimate overall application efficiency:

Total number of measured frames (*total frames*), actually the original cut can be longer, but operator applied color only on presented number of frames. It is important to recall that almost all frames are doubled, so the real number of frames where any interaction is performed is half.

Total time spent by active work (*total time*). We treated longer delays between two user driven operations (*maximal interaction delay*) as idle time. These breaks are omitted from performance statistics and are count in the time which was spent by relaxation (*total break time*). If we want to compare user's performance or difficulty of different movie cuts, then the important value is time needed for inking single frame (*average seconds per frame*), but the best overview of scene difficulty provide temporal graph where for each frame the time of active work was posted.

Also important is maximal active time spent by inking of single animation frame (*maximal frame delay*). Maximal frame delay take usually place at the starting frame of whole sequence, where no prediction frame is available so complete marking and retouching corrections have to be done.

Total time spent by retouching (*total stroking time*). This time was measured while the operator holds pressed button of mouse inside retouching application mode. For better comparison the percentage value relative to total active time and average retouching time (*average stroking per frame*) are also presented.

Whenever any mouse button was pressed inside index marking application mode the counter of *total marker clicks* was increased. From this value was derived well comparable associated statistics: average number of seconds between two mouse clicks (*average seconds per click*) and average number of mouse clicks per one frame (*average clicks per frame*).

#### 6.1.2 Frequently used functions

For reference we also present operator names (*operator name*) and their working speed relative to fixed norm 60 frames per hour. Also histogram of ten mostly used application functions is presented and percentage of overall usage of these functions relative to total number of available. This value usually summarize user's ability to use our application effectively. Follows short overview of these frequently used functions:

- **add-index-marker**: When active prediction technique fails on several regions or when no prediction is possible, user is able due to this function set up correct color assignment by proper color marker placed

- using mouse pointer inside IDX layer (see *Section 5.2.5*). Similar function is `change-index-marker` it unlike changes old color index of already placed marker to the actual one.
- `delete-nearest-marker`: Using this operation user deletes nearest active color index marker and thereby force application to estimate proper region color automatically.
  - `pick-index`: This sets actual color index using color from region that is actually under mouse pointer. Similar to `select-index` where color index is selected from user defined palette using mouse or keyboard.
  - `add-select-graph-marker`: This operation allows user to select regions that are connected by continuity graph to be in one region group. This is useful when two or more region groups are in the scene and when we want to apply for each one different prediction method, prediction frame or change of prediction layer geometry (see *Section 5.2.5*).
  - `switch-FIL`: This switch hides or shows the FIL layer inside application desktop. Following operations are usually performed afterwards:
    - `mode-erase`: or
    - `mode-delete`: which set up proper retouching mode (see *Section 5.2.4*).
    - `back-stroke`: Inside FIL it layer is possible to perform several retouching operations (see *Section 5.2.4*) this one erase edges, `edge-stroke` add edges and `clean-stroke` returns original edge shape. Very often before mentioned retouching operations the
    - `change-radius`: function is called due to fine tuning of circular stroke shape.
    - `copy-previous-strokes`: Default is copying of strokes from prediction frame automatic, however it is usable to switch off this operation and control copy function manually. This is usually used when it is possible to change prediction layer geometry and afterwards copy transformed retouching strokes from prediction frame to actual one.
    - `chagne-sweep-threshold`: Sweep threshold is important value that affect unsupervised contour authentication mechanism (see *Section 3.2.2*).
    - `change-prediction-frame`: This operation allows user to change prediction frame for actual region group and it is usually called whenever image sequence contains animation loops (see *Section 5.3.1*).
    - `change-prediction-type`: User performs switching between *intensity*, *position* or *combined* prediction inside IDX layer responsible for unsupervised color to region assignment (see *Section 5.3*).
    - `mode-realtime`: Function that switch between realtime and buffered mode is usable when user perform some sort of changes and he or she does not want to perform refreshing of modified layer immediately.
    - `mode-select`: Inside this application mode is user able to change region groups and to set up geometry of prediction layers.
    - `mode-zoom`: This operation is recorded whenever user switch on the windowed zoom function. Similar switching operation is `mode-scale` when desktop zoom is activated (see *Section 5.1.2*).
    - `move-prediction-plane`: Whenever translation of prediction layer is changed using both unsupervised shift estimation or manual tuning, this operation is activated (see *Section 5.3.2*).
    - `next-frame`: Simple operation that increase actual frame number is similar to function `previous-frame` made for frame decreasing.
    - `refresh-layers`: Whenever user is not sure that all layers are consistent due to memory caching or when disk cache is broken or totally missing he or she is able to refresh all layers of actual frame using this function.

## 6.2 Measurement

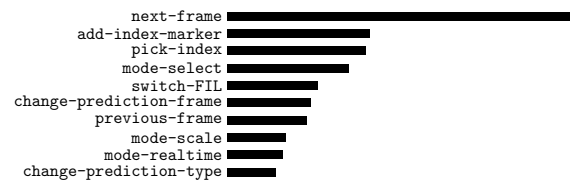
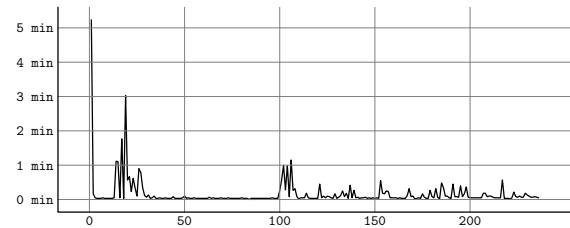
Next seven pages contain overviews, tables and graphs with statistics described in previous section. These statistics were obtained by measurement of 13 different image sequences from cartoon “*O loupežníku Rumcajsovi*”. It is important to know that each application user has different experiences and abilities. Another important feature is that following results come from real inking work performed without any special environmental conditions. Moreover users were in advance not well posted in fact that they are being measured. So there is possibility that such scene can be inked much more faster preserving same output quality, however we want to present realistic results not the best ones.





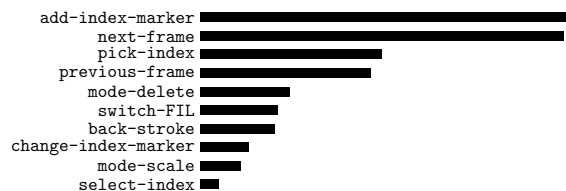
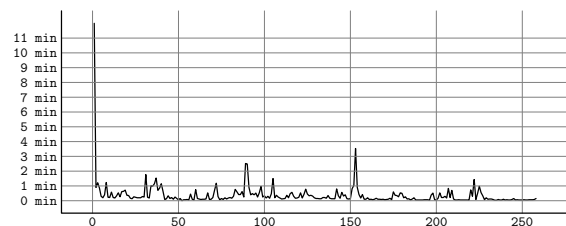
Only one big character, small number of active color indices. Contour free background. Almost static shot, rare fast changes between frames. Lots of frames should be perfectly predicted by position prediction. Practically no retouching work is needed. Example of really trivial scene.

movie shot	7th episode, cut 24
shot difficulty	<i>trivial</i>
total frames	<b>236</b>
total time	2340s (00:39:00)
total break time	154s (00:02:34)
total stroking time	1s (0%)
total marker clicks	316
maximal frame delay	314s
maximal interaction delay	14s
average seconds per frame	<b>9s</b>
average seconds per click	7s
average clicks per frame	1
average stroking per frame	0s
operator name	<i>D. Sýkora</i>
operator speed	6.1x (60 fph)
overall application usage	38%



Almost static scene without contours in background. The most of work have to be done on the first frame. Although character movement is not large there are lots of small regions that have to be corrected. Scene is suitable for position prediction.

movie shot	5th episode, cut 31
shot difficulty	<i>easy</i>
total frames	<b>258</b>
total time	5806s (01:36:46)
total break time	654s (00:10:54)
total stroking time	390s (6%)
total marker clicks	994
maximal frame delay	720s
maximal interaction delay	20s
average seconds per frame	<b>22s</b>
average seconds per click	5s
average clicks per frame	3
average stroking per frame	1s
operator name	<i>P. Bláhová</i>
operator speed	2.7x (60 fph)
overall application usage	28%

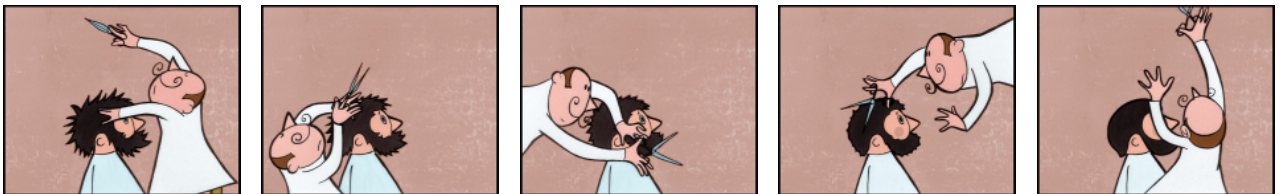
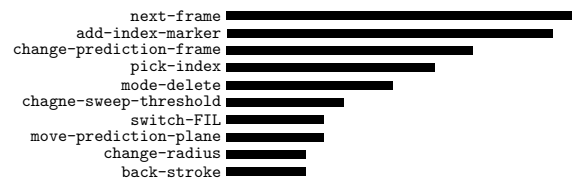
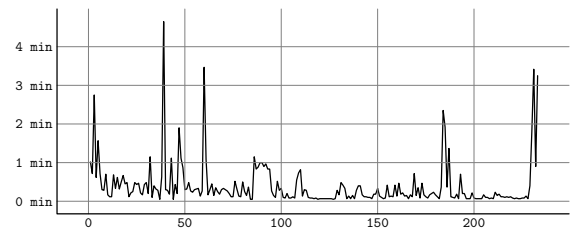






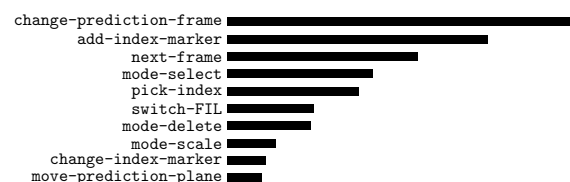
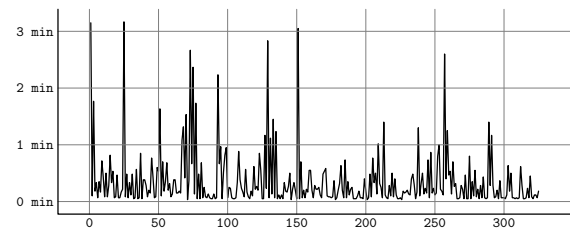
Camera zoom-out, left character gets out of sight, right sits down and camera zoom-in on his legs. Small number of shape changes or character motions. Zooming is handled by properly scaled masks for position prediction. Start frame is 40, where both characters are visible.

movie shot	1st episode, cut 22
shot difficulty	<i>easy</i>
total frames	<b>233</b>
total time	5444s (01:30:44)
total break time	777s (00:12:57)
total stroking time	617s (11%)
total marker clicks	531
maximal frame delay	279s
maximal interaction delay	29s
average seconds per frame	<b>23s</b>
average seconds per click	10s
average clicks per frame	2
average stroking per frame	2s
operator name	<i>T. Brabec</i>
operator speed	2.6x (60 fph)
overall application usage	30%



Solid contour free background, small number of active color indices, one static character. Rapid movement of second character. Each phase has really different spatial location. Narrow regions due to scissors. Shot is suitable for intensity prediction.

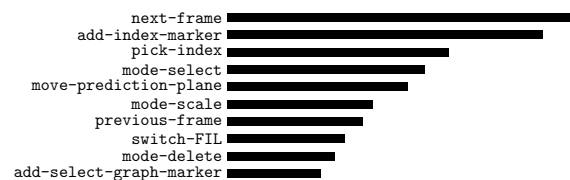
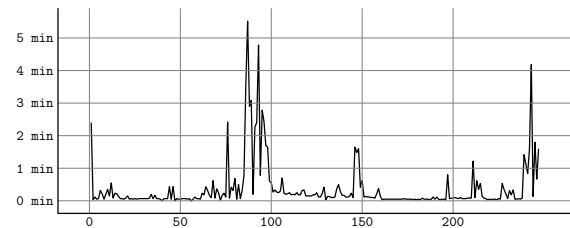
movie shot	2nd episode, cut 14
shot difficulty	<i>easy</i>
total frames	<b>325</b>
total time	6975s (01:56:15)
total break time	610s (00:10:10)
total stroking time	331s (4%)
total marker clicks	880
maximal frame delay	190s
maximal interaction delay	28s
average seconds per frame	<b>21s</b>
average seconds per click	7s
average clicks per frame	2
average stroking per frame	1s
operator name	<i>T. Brabec</i>
operator speed	2.8x (60 fph)
overall application usage	32%





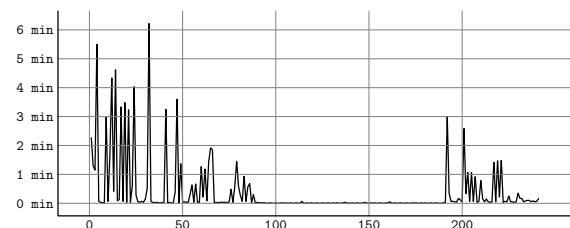
Start frame is 80. Camera zoom-in and out. Three independent region groups. Contour free background. Not so complicated character movement. Contour free ring reflex effect was done during post-production phase.

movie shot	4th episode, cut 11
shot difficulty	<i>easy</i>
total frames	<b>247</b>
total time	5571s ( <b>01:32:51</b> )
total break time	1952s ( <b>00:32:32</b> )
total stroking time	496s (8%)
total marker clicks	914
maximal frame delay	<b>331s</b>
maximal interaction delay	26s
average seconds per frame	<b>22s</b>
average seconds per click	6s
average clicks per frame	3
average stroking per frame	2s
operator name	<i>S. Drbohlav</i>
operator speed	2.7x (60 fph)
overall application usage	42%



Camera panning from left to right and back. There is no frame where both characters are completely visible. General is laughing during first 50 frames. This effect caused relative large shape shift between each frame and lots of human driven corrections.

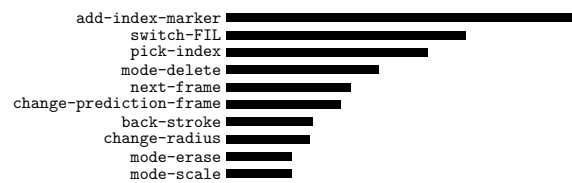
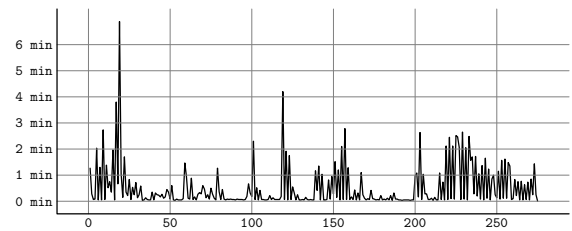
movie shot	7th episode, cut 25
shot difficulty	<i>intermediate</i>
total frames	<b>241</b>
total time	5444s ( <b>01:30:44</b> )
total break time	3869s ( <b>01:04:29</b> )
total stroking time	134s (2%)
total marker clicks	1577
maximal frame delay	373s
maximal interaction delay	20s
average seconds per frame	<b>22s</b>
average seconds per click	3s
average clicks per frame	6
average stroking per frame	0s
operator name	<i>O. Sýkora</i>
operator speed	2.7x (60 fph)
overall application usage	42%





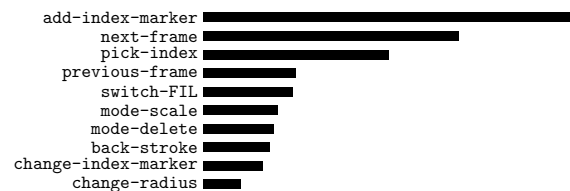
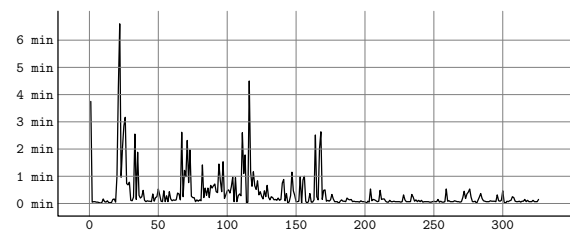
Fast changes of two basic shapes (look on the left and right side) suitable for mirrored position masks. Intensity fluctuation due to small regions with standalone color index (eggs). During last 50 frames little problematic translation movement of both characters.

movie shot	3th episode, cut 9a
shot difficulty	<i>intermediate</i>
total frames	<b>275</b>
total time	8425s ( <b>02:20:25</b> )
total break time	562s ( <b>00:09:22</b> )
total stroking time	1178s (13%)
total marker clicks	1600
maximal frame delay	413s
maximal interaction delay	29s
average seconds per frame	<b>30s</b>
average seconds per click	5s
average clicks per frame	5
average stroking per frame	4s
operator name	<i>T. Brabec</i>
operator speed	2.0x (60 fph)
overall application usage	43%

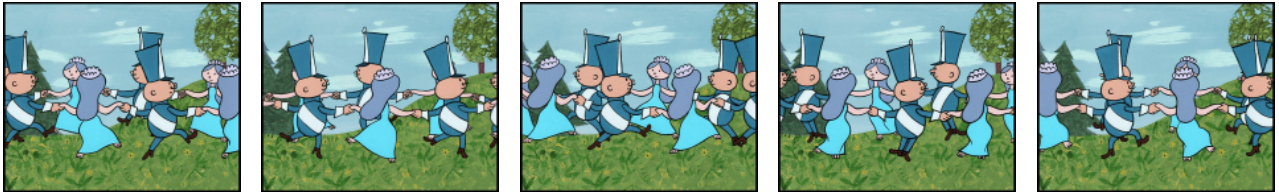


Fast camera panning from left to right. Lots of different color indices and several tight regions. Perfume splash and princess lips were done during post-production phase. Almost static scene without any complicated character movement well posed for position prediction.

movie shot	5th episode, cut 25
shot difficulty	<i>intermediate</i>
total frames	<b>326</b>
total time	7303s ( <b>02:01:43</b> )
total break time	4098s ( <b>01:08:18</b> )
total stroking time	675s (9%)
total marker clicks	1167
maximal frame delay	396s
maximal interaction delay	29s
average seconds per frame	<b>22s</b>
average seconds per click	6s
average clicks per frame	3
average stroking per frame	2s
operator name	<i>P. Bláhová</i>
operator speed	2.7x (60 fph)
overall application usage	30%

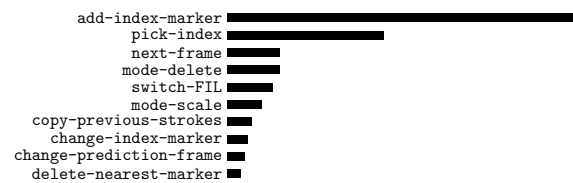
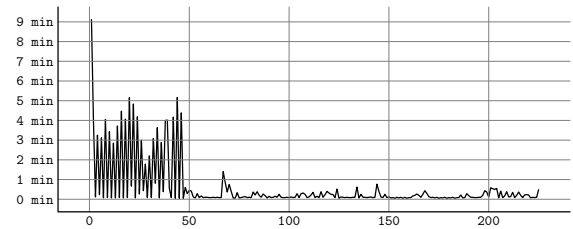






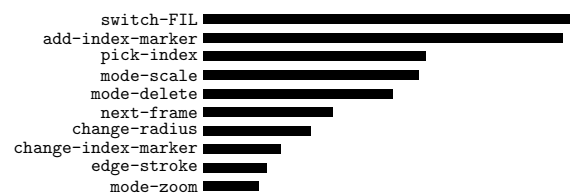
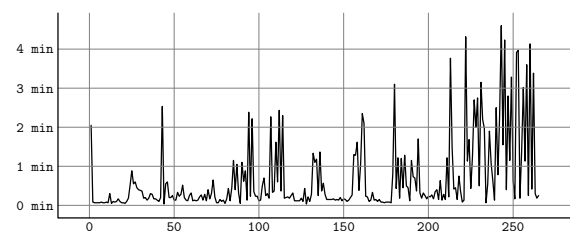
Dancing round of characters. Many connected and partially occluded region groups. Contour free background. Repetitive sequence of 50 frames. Next 175 frames with the same loop. Shot is suitable for user defined sequence of prediction frames.

movie shot	3th episode, cut 23
shot difficulty	<i>laborious</i>
total frames	<b>225</b>
total time	7954s ( <b>02:12:34</b> )
total break time	409s ( <b>00:06:49</b> )
total stroking time	691s (8%)
total marker clicks	2870
maximal frame delay	547s
maximal interaction delay	30s
average seconds per frame	<b>35s</b>
average seconds per click	2s
average clicks per frame	12
average stroking per frame	3s
operator name	<i>T. Brabec</i>
operator speed	1.7x (60 fph)
overall application usage	31%



Two separated region groups. Only local movement in the beginning of the shot. Suitable for both position and intensity prediction. Contour free background. Advanced retouching is not necessary. In last 50 frames little complicated directional movement of both characters.

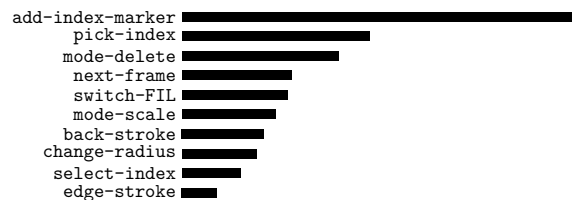
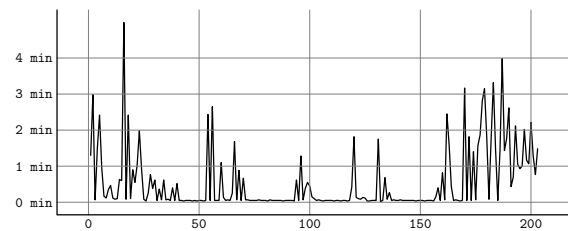
movie shot	6th episode, cut 36
shot difficulty	<i>laborious</i>
total frames	<b>265</b>
total time	10620s ( <b>02:57:00</b> )
total break time	1370s ( <b>00:22:50</b> )
total stroking time	1399s (13%)
total marker clicks	2306
maximal frame delay	276s
maximal interaction delay	27s
average seconds per frame	<b>40s</b>
average seconds per click	4s
average clicks per frame	8
average stroking per frame	5s
operator name	<i>I. Gobelová</i>
operator speed	1.5x (60 fph)
overall application usage	37%





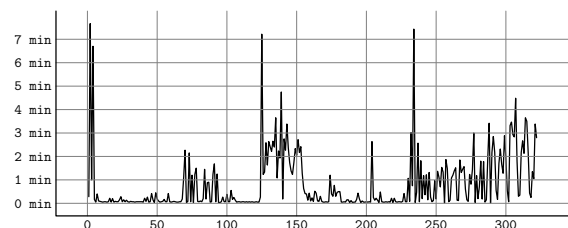
Continous camera panning from left to right and zoom-in on the end of shot. Contour free background. Foreground parts are sometimes occluded by background shapes. Complicated retouching of missing contours. Lots of different color indices and fast unpredictable character movement.

movie shot	3th episode, cut 1b
shot difficulty	<i>laborious</i>
total frames	<b>203</b>
total time	6637s ( <b>01:50:37</b> )
total break time	498s ( <b>00:08:18</b> )
total stroking time	777s (11%)
total marker clicks	1505
maximal frame delay	299s
maximal interaction delay	30s
average seconds per frame	<b>32s</b>
average seconds per click	4s
average clicks per frame	7
average stroking per frame	3s
operator name	<i>T. Brabec</i>
operator speed	1.8x (60 fph)
overall application usage	36%



Small foreground regions, unwanted contours in background, partially occluded, complex retouching. Two spatially connected region groups. Complex movement, especially rotation (position prediction with mask rotation). Camera zoom-in during last 50 frames.

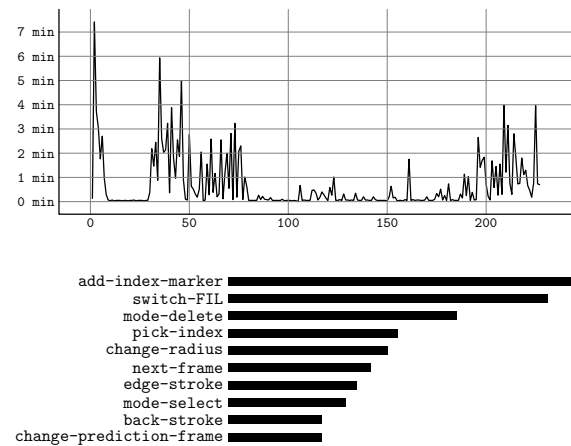
movie shot	8th episode, cut 15
shot difficulty	<i>hard</i>
total frames	<b>322</b>
total time	15551s ( <b>04:19:11</b> )
total break time	4480s ( <b>01:14:40</b> )
total stroking time	4377s (28%)
total marker clicks	1823
maximal frame delay	460s
maximal interaction delay	30s
average seconds per frame	<b>48s</b>
average seconds per click	8s
average clicks per frame	5
average stroking per frame	13s
operator name	<i>I. Gobelová</i>
operator speed	1.2x (60 fph)
overall application usage	46%





Contour free region boundaries which have to be virtually painted using retouching brush. Fast motion effect visualised by white smudges sometimes completely broken character contours. Lots of retouching work. Complex movement of three different characters with camera panning.

movie shot	2nd episode, cut 23b
shot difficulty	<i>hard</i>
total frames	<b>227</b>
total time	9323s ( <b>02:35:23</b> )
total break time	1431s ( <b>00:23:51</b> )
total stroking time	2006s (21%)
total marker clicks	1281
maximal frame delay	445s
maximal interaction delay	26s
average seconds per frame	<b>41s</b>
average seconds per click	7s
average clicks per frame	5
average stroking per frame	8s
operator name	<i>O. Sýkora</i>
operator speed	1.5x (60 fph)
overall application usage	42%



## 6.3 Results

Presented experiments show us that inking process based on COL application is still far from the ideal of example driven unsupervised inking. The average processing speed vary from 20 to 40 seconds per frame while the main slowdown is usually caused by simply unpredictable extensive character movement, correction of large number of small foreground regions and retouching of tight regions, broken contours and contours inside background area. If we do not take in account pre-processing and post-processing phases, one user is able to apply ink on whole episode in two weeks of full time inking. If we include background restoration and other post-production tasks we should conclude that one episode could be completed in one month.

It is now possible to make raw comparison of above presented processing speed with time complexity of original technology based only on hi-end film editing post-production hardware driven by experienced user. During testing phase on this technology was proved that one user is able to apply ink with the same or longer duration, but with lower image quality and much higher costs due to expensive machine time. Thanks to parallelism on cost effective workstations, COL application allow us to speed-up this time consuming process, increase the final image quality and above all, dramatically decrease the outgoings of whole project.

However we have to consider that we are focussed on high quality of final image sequence when the main slowdown is usually caused by correction of small regions and by complex retouching of contours in background. It is hard even for human to produce correct inking. Small regions are usually badly inked because of user inattention due to perceptual capabilities of human eye in front of still images. Bugs usually come up later if we review whole image sequence using original frame rate. If we leave these minor mistakes alone the final image quality will be still acceptable and the processing speed is going to be even more than two times faster.

# 7



---

## Conclusion

The main motivation of this thesis was to speed up and make cheaper original slow and expensive inking technology. Unfortunately objective decision of fulfilment of the basic motivation is not possible due to really different properties of purposed inking technologies. The main difference lie in the final visual quality which is much more better relative to that obtained by luminance keys driven technology used in hi-end film editing systems. Accomplished speed up is also discussable while we enable feasibility of parallelism which was not initially accessible. The main reason why we develop purposed inking technology was due to cost untenability of original technology. Only fulfilment of this main task denote to success of our application.

It was proved that role of hi-end film editing system is invaluable, however we are able using COL application to increase processing speed, final visual quality and take in account possible usage of cost effective PC workstations. This cooperation stay beyond final product, so we could not simply say if our application is faster or better than hi-end film editing system. What we have to say is fact that our application was successfully used as important part of inking pipeline and practically enables the feasibility of whole project.

### 7.1 Future work

We still did not exploit additional information stored inside background layer. This layer can be used for separation of foreground regions from background. Unfortunately due to incorrect global camera motion tracking it is not possible to simply fit new layer to proper position and compute alpha mask of foreground using straightforward image difference. More advanced comparison method insensitive to local pixel shifts have to be introduced.

Another problem is connected with low resolution: contours of tight regions are usually merged together and form one bold contour which has to be manually retouched by special type of semi-automatic stroke. We already saw that sub-pixel resolution is able to solve this problem. However faster method that works with original resolution have to be invented.

The weak point of COL application is also prediction. Although current implementation based on simple intensity and position prediction is able to significantly speed up whole inking process, it is possible to spent large effort by development of advanced prediction technique based on robust statistic models in connection with *Bayesian* framework (widely used on the filed of pattern recognition).

Small regions introduce fundamental and troublesome problem. The number of samples inside these regions is usually below the limit with that we are able to produce usable prediction statistics. This causes e.g. still unsolved temporal luminance fluctuation of small regions with unique color index. If we consider the fact that more than half of user driven correction work is still connected with small regions, we are not able to significantly speed up inking process by better prediction technique. The more robust estimation of region intensity and shape statistics have to be introduced.

---

This work was supported by *Universal Production Partners (UPP)* and *Digital Media Production (DMP)*.



# References

---

- [Blinn96] J. F. Blinn and A. R. Smith. Blue screen matting. In *SIGGRAPH 1996 Conference Proceedings*, pages 259–268, August 1996.
- [Borgefors86] G. Borgefors. Distance transformations in digital images. *Computer Vision, Graphics, and Image Processing*, 34(3):344–371, 1986.
- [Canny86] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.
- [Chen87] J. S. Chen, A. Huertas, and G. Medioni. Fast convolution with Laplacian-of-Gaussian masks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(4):584–590, July 1987.
- [Chen02] Junqing Chen, Thrasyvoulos N. Pappas, Aleksandra Mojsilovic, and Bernice Rogowitz. Adaptive image segmentation based on color and texture. In *Proceedings of IEEE International Conference on Image Processing*, September 2002.
- [Cheng00] Heng-Da Cheng and Ying Sun. A hierarchical approach to color image segmentation using homogeneity. *IEEE Transactions on Image Processing*, 9(12):2071–2082, December 2000.
- [Chuang01] Yung-Yu Chuang, Brian Curless, David H. Salesin, and Richard Szeliski. A Bayesian approach to digital matting. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 264–271, December 2001.
- [Clark89] J. J. Clark. Authenticating edges produced by zero-crossing algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(1):43–57, 1989.
- [Deng01] Yining Deng and B. S. Manjunath. Unsupervised segmentation of color-texture regions in images and video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(8):800–810, August 2001.
- [Ding01] Lijun Ding and Ardeshir Goshtasby. On the Canny edge detector. *IEEE Transactions on Computer Vision and Pattern Recognition*, 34(3):721–725, March 2001.
- [Driessen91] J. N. Driessen, L. Boroczky, and J. Biermond. Pel-recursive motion field estimation from image sequences. *Journal of Visual Communication and Image Representation*, 2(3):259–280, September 1991.
- [Gonzalez87] Rafael C. Gonzalez and Paul Wintz. *Digital Image Processing*. Addison-Wesley, Reading, Massachusetts, 2nd edition, 1987.
- [Haenselmann00] Thomas Haenselmann, Claudia Schremmer, and Wolfgang Effelsberg. Wavelet-based semi-automatic segmentation of image objects. In *Proceedings of IEEE International Conference on Signal and Image Processing*, pages 387–392, November 2000.
- [Haralick92] Robert M. Haralick and Linda G. Shapiro. *Computer and Robot Vision*, volume 1. Addison Wesley, New York, NY, USA, 1992.
- [Haris98] K. Haris, S. N. Estradiadis, N. Maglaveras, and A. K. Katsaggelos. Hybrid image segmentation using watersheds and fast region merging. *IEEE Transactions on Image Processing*, 7(12):1684–1699, 1998.
- [Heath96] M. D. Heath, S. Sarkar, T. Sanocki, and K. W. Bowyer. Comparison of edge detectors: A methodology and initial study. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 143–148, 1996.
- [Hertzmann01] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image analogies. In *SIGGRAPH 2001 Conference Proceedings*, pages 327–340, 2001.

- [Hlaváč94] V. Hlaváč, M. Šonka, and R. Boyle. *Image Processing, Analysis and Machine Vision*. Chapman and Hall Computing, 1994.
- [Hrbek02] Štěpán Hrbek. Suppression of global luminance fluctuation in aged image sequences. Technical report, Department of Mathematics and Physics, Charles University, Prague, October 2002.
- [Hug99] Johannes Hug, Christian Brechbühler, and Gábor Székely. Tamed snake: A particle system for robust semi-automatic segmentation. In *Proceedings of 2nd International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 106–115, Cambridge, UK, September 1999.
- [Joyeux01] L. Joyeux, S. Boukir, B. Besserer, and O. Buisson. Reconstruction of degraded image sequences. Application to film restoration. *Image and Vision Computing*, 19:503–516, September 2001.
- [Kalra97] S. Kalra. A new auto-regressive model-based algorithm for motion picture restoration. In *Processings of International Conference on Acoustics, Speech, and Signal Processing*, volume 4, 1997.
- [Kégl102] Balázs Kégl and Adam Kryžak. Piecewise linear skeletonization using principal curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(1):59–74, January 2002.
- [King82] D. King. Implementation of the Marr-Hildreth theory of edge detection. Technical Report ISG-102, The University of Southern California, October 1982.
- [Koepfler94] G. Koepfler, C. Lopez, and J. M. Morel. A multiscale algorithm for image segmentation by variational method. *SIAM Journal on Numerical Analysis*, 31(1):282–299, February 1994.
- [Kokaram93] A. C. Kokaram. *Motion picture restoration*. PhD thesis, Cambridge University, 1993.
- [Kokaram98] A. C. Kokaram. *Motion Picture Restoration: Digital Algorithm for Artefact Suppression in Degraded Motion Picture Film and Video*. Springer, London, UK, 1998.
- [Lee83] C. H. Lee. Resotring spline interpolation of ct images. *IEEE Transactions on Medical Imaginig*, 3:142–149, 1983.
- [Li96] S. Z. Li. Bayesian image restoration and segmentation by constrained optimization. In *Processings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–6, 1996.
- [Liang01] Lin Liang, Ce Liu, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics*, 20(3):127–150, July 2001.
- [Liu93] B. Liu and A. Zaccarin. New fast algorithms for the estimation of block motion vectors. *IEEE Transactions on Circuits and Systems for Video Technology*, 3(2):148–57, 1993.
- [Marr80] D. Marr and E. C. Hildreth. Theory of edge detection. In *Proceedings of Royal Society*, volume B207, pages 187–217, 1980.
- [Martinez86] D. M. Martinez. *Model-based motion estimation and its application to restoration and interpolation of motion pictures*. PhD thesis, Massachusetts Institute of Technology, 1986.
- [Meyer90] F. Meyer and S. Beucher. Morphological segmentation. *Journal of Visual Communication and Image Representation*, 1(1):21–46, September 1990.
- [Mitchell88] O. Robert Mitchell and Edward P. Lyvers. Precision edge contrast and orientation estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(6):927–937, November 1988.

- [Mrázek01] Pavel Mrázek. *Nonlinear Diffusion for Image Filtering and Monotonicity Enhancement*. PhD thesis, Center for Machine Perception, Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University, Prague, Czech Republic, December 2001.
- [Nieminen87] A. Nieminen, P. Heinonen, and Y. Neuvo. A new class of detail-perserving filters for image processing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9:74–90, 1987.
- [Nishihara81] H. K. Nishihara and N. G. Larson. Towards a real-time implementation of the Marr and Poggio stereo matcher. In *Proceedings of DARPA Image Understanding Workoshop*, pages 114–120, April 1981.
- [Nitzberg93] Mark Nitzberg, David Mumford, and Takahiro Shiota. *Filtering, segmentation, and depth*, volume 662 of *Lecture Notes in Computer Science*. Springer-Verlag Inc., New York, NY, USA, 1993.
- [Pank98] Bob Pank. *The Digital Fact Book*. Quantel Limited, Newbury, UK, 1998.
- [Patras01] Ioanus Patras and R. L. Lagendijk. Video segmentation by MAP labeling of watershed segments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(3), March 2001.
- [Pitas93] I. Pitas. *Digital Image Processing Algorithms*. Prentice Hall International, London, UK, 1993.
- [Prewitt70] J. M. S. Prewitt. Object enhancement and extraction. In *Picture Processing and Psychopictorics*, pages 75–149. B. S. Lipkin et al., Eds. Academic Press, 1970.
- [Reinhard01] Erik Reinhard, Michael Ashikhmin, Bruce Gooch, and Peter Shirley. Color transfer between images. *IEEE Transactions on Computer Graphics and Applications*, 21(5):34–41, September/October 2001.
- [Rosenfeld82] A. Rosenfeld and A. C. Kak. *Digital Picture Processing*, volume 1. Academic Press, Orlando, USA, 1982.
- [Ruderman98] D. L. Ruderman. Statistics of cone responses to natural images: Implications for visual coding. *Journal of Optical Society of America*, 15(8):2036–2045, 1998.
- [Schneider00] M. K. Schneider, P. W. Fieguth, W. C. Karl, and A. S. Willsky. Multiscale methods for the segmentation and reconstruction of signals and images. *IEEE Transactions on Image Processing*, 9(3):456, March 2000.
- [Serra93] Jean Serra. *Image Analysis and Mathematical Morphology*, volume 1. Academic Press, Oval Road, London, UK, 4th edition, 1993.
- [Shen86] J. Shen and S. Castan. An optimal linear operator for edge detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 109–114, June 1986.
- [Shen00] Fei Shen and Han Wang. Real time gray level corner detector. In *Proceedings of the 6th International Conference on Control, Automation, Robotics and Vision*, Singapore, December 2000.
- [Smith97] S. M. Smith and J. M. Brady. SUSAN – a new approach to low-level image processing. *International Journal of Computer Vision*, 32(1):45–78, May 1997.
- [Sotak89] G. E. Sotak and K. L. Boyer. The Laplacian-of-Gaussian kernel: a formal analysis and design procedure for fast, accurate convolution and full-frame output. *Computer Vision, Graphics, and Image Processing*, 48(2):147–189, November 1989.
- [Steger98] Carsten Steger. Evaluation of subpixel line and edge detection precision and accuracy. In *International Archives of Photogrammetry and Remote Sensing*, volume 32, pages 256–264, 1998.

- [Suzuki86] S. Suzuki and K. Abe. Sequential thinning of binary pictures using distance transformation. In *Proceedings of the 8th International Conference on Pattern Recognition*, pages 289–292, 1986.
- [Tomasi00] C. Tomasi and M. A. Ruzon. Alpha estimation in natural images. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 18–25, June 2000.
- [Vincent91] L. Vincent and P. Soille. Watersheds in digital spaces: an efficient algorithm based on immersion simulations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):583–598, June 1991.
- [Welsh02] Tomihisa Welsh, Michael Ashikhmin, and Klaus Mueller. Transferring color to greyscale images. In *SIGGRAPH 2002 Conference Proceedings*, pages 277–280, 2002.
- [Witkin86] A. P. Witkin. Scale space filtering. In A. P. Pentland, editor, *From Pixels to Predicates: Recent Advances in Computational and Robot Vision*, pages 5–19, Norwood, NJ, USA, 1986. Ablex.
- [Wyszecki82] G. Wyszecki and W. Stiles. *Color Science: Concepts and Methods, Quantitative Data and Formulas*. Wiley, New York, NY, USA, 1982.
- [Yezzi97] A. Yezzi, S. Kichenassamy, A. Kumar, P. Olver, and A. Tannenbaum. Geometric snakes for edge detection and segmentation of medical imagery. *IEEE Transactions on Medical Imaginig*, 16(2):199–209, April 1997.
- [Zhou99] Yong Zhou and Arthur W. Toga. Efficient skeletonization of volumetric objects. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):196–209, July/September 1999.
- [Ziou97] D. Ziou and S. Tabbone. Edge detection techniques - An overview. Technical Report 195, Department of Mathematics and Informatique, Universit de Sherbrooke, October 1997.

# Appendix

---

## Contents of supplied CD-ROM

Following papers cited in this thesis are located on CD-ROM in PDF or PS format (in section *References*): [Clark89], [Deng01], [Ding01], [Haenselmann00], [Haris98], [Heath96], [Hertzmann01], [Hug99], [Chen02], [Cheng00], [Chuang02], [Koepfler94], [Kokaram93], [Mrázek01], [Patras01], [Reinhard01], [Shen00], [Steger98], [Welsh02], [Yezzi97], [Ziou97]. Additionally several related but not cited papers and full text of this diploma thesis in PDF format (print and preview version) are also available on the supplied CD-ROM. Besides the full *DirectX* version of COL application with one example image sequence from cartoon “*O loupežníku Rumcajsovi*” is also included together with user reference manual in czech language. It is possible to try apply color and inspect final color images in full quality. Moreover in section *Experiments* there is over 10 minutes of grey-scale and color sequences selected from 5th, 6th and 7th episode of the same cartoon encoded using *DivX* video codec (also available on supplied CD-ROM). Unfortunately source code of COL application is not available due to exclusive rights of *Digital Media Production*.