Vision, Modeling, and Visualization (2016)
M. Hullin, M. Stamminger, and T. Weinkauf (Eds.)

A Framework for Interactive Realtime Image Editing

Benjamin Hell, Moritz Mühlhausen, Marcus Magnor (TU Braunschweig)

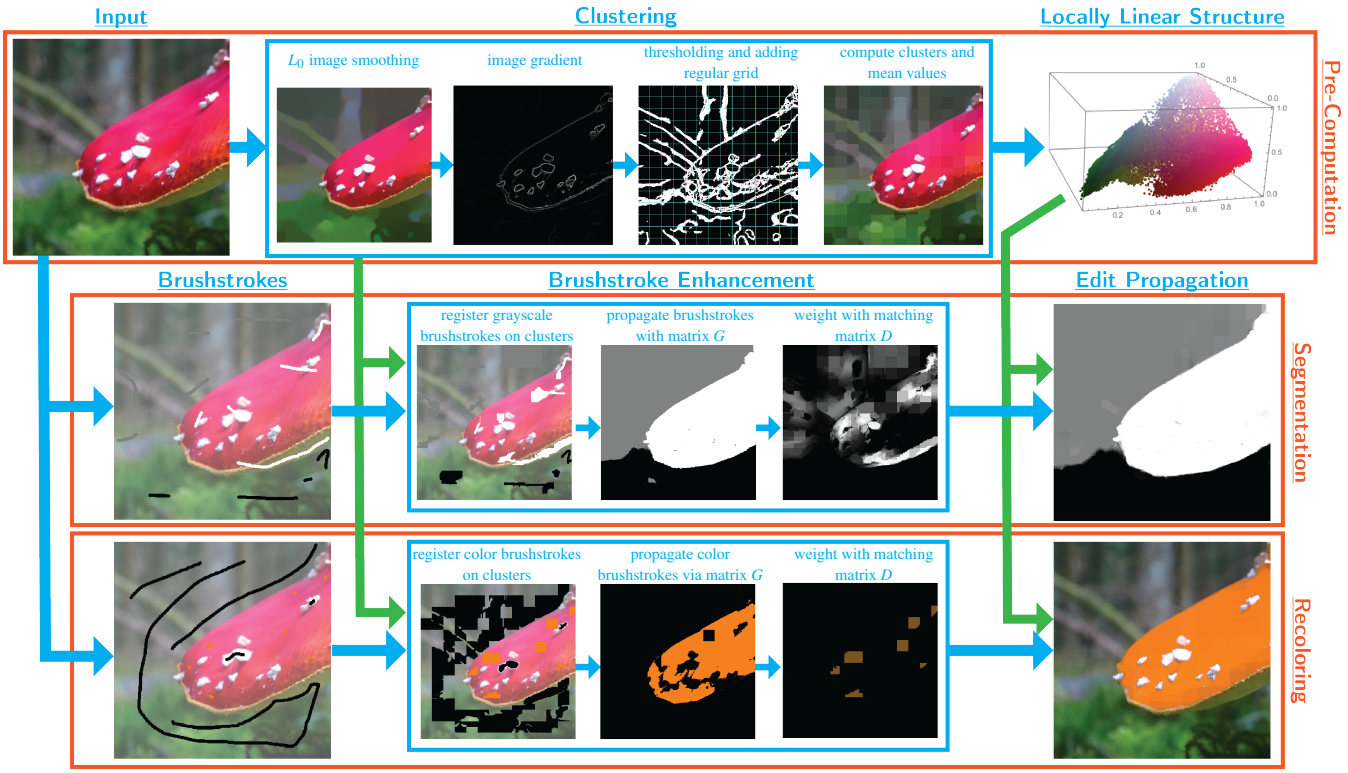


Figure 1: ***Pipeline Overview****: The illustration shows an overview of our complete pipeline. The first row corresponds to the major steps in the pre-computation part, which has to be executed once in the beginning. The second and third row show the interactive part, which comes into play when working with user input, where the second row shows the application of image segmentation, while the third one shows image recoloring based on colored user input.*

Abstract

We present a complete framework for interactive image editing with a focus on quickly obtaining results based on user input via brushstrokes. The goal is to show feedback to changing input in realtime while maintaining high quality output. We achieve this by extracting vital information in the form of local color clusters and a corresponding locally linear structure from the input image. With this information the problem of propagating any sort of features provided by user input can be executed conveniently and very efficiently. One of the major merits of our approach is that the locally linear structure does not impose any restrictions on the cluster structure. The fully automatically created structure essentially allows for a flexible scaling of the overall computational cost for propagating user edits. If needed, a more detailed output can be obtained by enforcing the creation of smaller clusters.

Categories and Subject Descriptors (according to ACM CCS): I.4.6 [IMAGE PROCESSING AND COMPUTER VISION]: Segmentation—Edge and feature detection

1. Introduction

The task of Image editing is required to change a still picture or video to suit the viewers needs or extract vital information from it. In some cases algorithms can generate desired results autonomously, but for many practical applications additional user input is required to specify semantic connections that algorithms cannot find or to make choices when there is no clear intention spec-

DOI: 10.2312/vmv.20161348

ified otherwise (like choosing specific color tones to change, regions to select, etc.). A common way to provide this information is by using brushstrokes, as popularized by Levin et al. [LLW04], to mark certain pixels in an image and provide additional color information. Using brushstrokes and spreading this information in the image regarding its underlying structure is commonly referred to as edit propagation.

In many situations it is highly beneficial for the user to get fast feedback to the input that has been provided. One reason for that is the fact that brushstrokes usually have to be adjusted or extended multiple times to achieve the desired result. Classical edit propagation approaches like Levin et al. [LLW04] and Lischinski et al. [LFUS06] and more recent approaches like Farbman et al. [FFLS08], Chen et al. [CZZT12] and Xu et al. [XYJ13] achieve high quality output but are not very efficient in terms of computation time and most importantly are not designed with the goal of being highly parallelizable. Nevertheless, many of their ideas are applicable in a parallel setting or can be accelerated using other speedup techniques. Our idea is to provide a scalable cluster structure that establishes a high symbiosis when being combined with Locally Linear Embedding, which has been used with much success by Chen et al. [CZZT12] and was introduced by Roweis and Saul [RS00].

The remainder of this paper is structured in the following way: First, we complete the introductory part by mentioning additional related work and our contributions. Section 2 then goes into detail about all the mathematical modeling aspects of our pipeline presented in Figure 1. Section 3 gives more details on how to actually solve the problems introduced before and provides pseudocode for several algorithms, which is also intended to give an overview of the programmatic design of our overall pipeline. Section 4 provides results and computation times for our approach and finally Section 5 concludes the paper.

1.1. Related Work

Originally user brushstrokes were propagated locally according to image gradients. A framework with user brushstrokes as input for edit propagation algorithms was first introduced by Levin et al. [LLW04] and was used for the purpose of colorization of images and videos. Later Lischinski et al. [LFUS06] extended the framework for tonal adjustments using image gradients for edge detection purposes. Pellacini and Lawrence [PL07] introduced a brushstroke based framework to edit measured materials. It is important to note that these local propagation methods have problems with fragmented image regions, requiring more user scribbles to generate pleasing results. Li et al. [LAA08] added pixel classification to address this problem. At each pixel an affine combination of different brushstrokes is considered. In contrast to that, An et al. [AP08] propagated the user input with regard to all pixel pairs. Although this solves the problem of disconnected regions, it is inefficient in terms of computation time and memory usage. Liu et al. [LSS09] came up with an edge-aware user guided selection with brushstrokes, considering the local user interaction as a progressive process to create the desired selection step by step. Solving a series of local optimization problems in the user's scribble direction, they achieved real time interaction. Meanwhile, Xu et al. [XLJ*09] accelerated edit propagation algorithms using k-d trees: Rectangular pixel clusters are introduced by dividing the image with a k-d tree. Li et al. [LJH10] formulated the edit propagation problem as a radial basis functions interpolation problem. This resulted in better memory and computation efficiency than in preceding work. Chen et al. [CZZT12] introduced a two step edit propagation: Capturing the manifold structure by using locally linear embedding [RS00] and propagating the user input while maintaining this structure afterwards. Although computation times are higher than in the work of Li et al. [LJH10], the results are better in regions with color blending. Extending the work of Chen et al. [CZZT12], Ma and Xu [MX14] were able to accelerate the computation by adapting the number of neighbors for each pixel based on local complexity. Furthermore, the authors used a similar built k-d tree to [XLJ*09], solving the edit propagation only on node corners and interpolating the values for other pixels. However, this approach needs additional user input if some details fall inside of k-d tree rectangles not touching any node. This work is partially based on the approach of Chen et al. [CZZT12] as well. In contrast to [MX14] our approach introduces clustering of pixels that adapt to the local complexity. This clustering is done before calculating the local linear dependencies to lower the dimensionality of the problem.

1.2. Contributions

This paper contains the following contributions:

- We establish a gradient independent cluster based image structure for very fast edit propagations.
- We show a way to propagate user input in the form of brushstrokes to incorporate global information in the edit propagation workflow.
- We present a full pipeline for a cluster based interactive editing process, which is fast enough to allow for realtime feedback to user input.

1.3. Notation and Abbreviations

The following notation will be used in this paper.

$\Omega \subset \mathbb{R}^2$:	image domain
I :	color feature vector in $\mathbb{R}^3$
$\|.\|_{L_p(\Omega)}, \|.\|_p$:	L_p-norm, p-norm, $\|.\|$ is 2-norm
$\tilde{.}$:	marks full 5D feature vector (color $\times$ location)
$\hat{.}$:	denotes obtained optimal result
r :	total number of clusters
C_i :	i-th cluster, i.e. i-th set of accumulated pixels
$\|C\|$:	number of pixels contained in cluster C
x_C :	pivot (location) of cluster C
g^C :	set of all brushstroke feature vectors on clusters
S^C :	set of all marked clusters
N_C^m :	indices of all clusters in m-neighborhood of cluster C
$\tilde{N}_C^k$:	indices of all brushstroke clusters in k-neighborhood of cluster C
n :	dimension of brushstroke feature vector and final output
$W = (\hat{w}_{ij})$:	weighting matrix representing locally linear structure
G :	matrix of gaussian mixture model weights
D :	diagonal matching matrix
E_l :	$l \times l$ identity matrix
$0_{\mathbb{R}^l}$:	zero vector in $\mathbb{R}^l$
$\mathrm{BV}(\Omega)$:	set of functions of bounded variation on Ω

2. The Mathematical Model

This section deals with the modeling aspect of our pipeline shown in Figure 1. Every single specific design choice we made is based on computation time, scalability and output quality. It is important to note that we put a special emphasis on fast computability, which will also shine through in the simplicity of the introduced optimization problems. The key to do this is the combination of the cluster structure and the locally linear image structure. We make use of two very important properties: Firstly, the locally linear structure is preserved when going from pixel level to cluster level (cf. Figure 2) and secondly, it is independent of image gradients and does not need any regular grid structure.

In the following we explain the mathematical details of every single subproblem that is part of our overall workflow shown in Figure 1. We take a look at all the parts that can be pre-computed before any user interaction takes place and introduce the way we deal with user input in the form of brushstrokes afterwards.

2.1. Pre-Calculation

The pre-calculation step consists of all the computations that are not dependent on any user input, hence only need to be computed once in the beginning (cf. first row in Figure 1). The first step in this process is to analyze the structure of the image for local accumulations of pixels with approximately the same intensity I. These pixels will then be bundled into so called superpixels or clusters. Afterwards a graph representing linear dependencies between these clusters with regard to their pivot x_C and mean color intensity I_C is established, which yields a structure independent of any sort of image gradients and hence any sort of regular grid, which could limit numerical computations later on.

2.1.1. Subdividing into Clusters

The goal of this section is to subdivide the image into local clusters with respect to their 5D feature vectors (color $\times$ location). The motivation behind finding locally coherent clusters containing pixels of approximately the same color becomes evident when looking at the following steps in our pipeline 1. For the color cohesion property of each individual cluster we use a very efficient edge detection or image smoothing technique presented in [XLXJ11]. Basically we solve the following L_0-optimization problem

$$\begin{aligned} \min_{z\in \mathrm{BV}(\Omega)} \lambda_{L_0} \|\nabla z\|_{L_0(\Omega)} + \|z-I\|^2_{L_2(\Omega)} &\overset{\text{on grid}}{\approx} \\ \lambda_{L_0} \sum_{x\in\Omega} \|\nabla z(x)\|_0 + \sum_{x\in\Omega} \|z(x)-I(x)\|^2 & \end{aligned} \tag{1}$$

where minimization of the L_0-norm guarantees that the output image represented by z is a thoroughly smoothed version of the input image I. Dependent on the parameter λ_{L_0}, regions of actual intensity change are vastly reduced. We consider the remaining locations with actual intensity change the hard edges of the image that clusters shall not pass. For local coherence we simply subdivide the image into a regular grid and limit each cluster to the corresponding grid cells. Combining the grid cells with the edge map obtained from the L_0-optimization process yields a cluster structure with the aforementioned desired property.

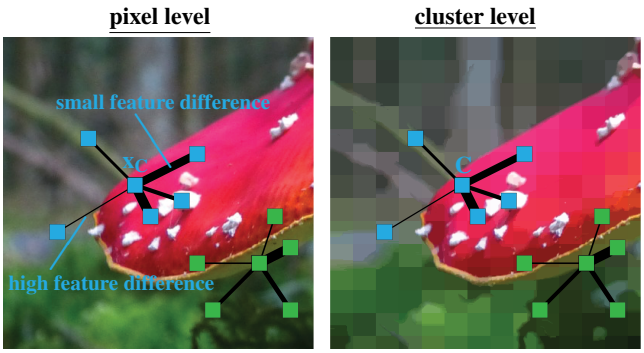


Figure 2: ***Locally Linear Structure Preservation****: Observation of the local neighborhood shows that the image structure in terms of 5D feature vectors $\tilde{I}$ (left) and $\tilde{I}^C$ (right), as defined in (4), is vastly preserved by the clustering process. This means that the locally linear structure of an image (cf. Section 2.1.2) is also preserved, which is the foundation of our clustering-based edit propagation scheme.*

We finally assign to each cluster C its pivot $x_C \in \mathbb{R}^2$ as its location vector and the mean intensity value of all pixels contained in the cluster as its intensity I_C:

$$|C| := \sum_{x\in C} 1, \quad I_C := \frac{1}{|C|}\sum_{x\in C} I(x), \quad x_C := \frac{1}{|C|}\sum_{x\in C} x \tag{2}$$

2.1.2. Extracting a Locally Linear Image Structure

The goal of this section is to provide the mathematical details on extracting structural information from 5D image data (pixel location and color information), which should be preserved when propagating user edits later on. For our purposes it is paramount that this structure can be computed without forcing our previously introduced clusters to form any sort of regular grid or have other restrictions like gradient based image measures would do (cf. Figure 2). We found the locally linear structure presented in [RS00] to be perfectly suited for this task.

As we want to treat location information and color information differently we define 5D feature vectors in the following way

$$\tilde{I} = \tilde{I}(x) := \begin{pmatrix} I(x) \\ 0_{\mathbb{R}^2} \end{pmatrix} + \tau \begin{pmatrix} 0_{\mathbb{R}^3} \\ x \end{pmatrix} = \begin{pmatrix} I(x) \\ \tau\, x \end{pmatrix} \tag{3}$$

where τ obviously balances the influence of color and location information. For feature vectors of clusters we define the obvious extension

$$\tilde{I}_C := \frac{1}{|C|}\sum_{x\in C} \tilde{I}(x) = \begin{pmatrix} I_C \\ \tau\, x_C \end{pmatrix}, \quad \tilde{I}^C := \{\tilde{I}_{C_1}, \ldots, \tilde{I}_{C_r}\} \tag{4}$$

with x_C being the pivot of cluster C and r the number of clusters.

The problem of finding the locally linear structure at each cluster C_i is given by the following optimization problem

$$\begin{aligned} \underset{w_i=(w_{i1},\ldots,w_{im})\in\mathbb{R}^m}{\operatorname{argmin}} & \left\| \tilde{I}_{C_i} - \sum_{j\in N^m_{C_i}} w_{ij}\, \tilde{I}_{C_j} \right\|^2 = \left\| \sum_{j\in N^m_{C_i}} w_{ij}\, \left(\tilde{I}_{C_i} - \tilde{I}_{C_j}\right) \right\|^2 \\ & \text{s.t.} \sum_{j\in N^m_{C_i}} w_{ij} = 1 \end{aligned} \tag{5}$$

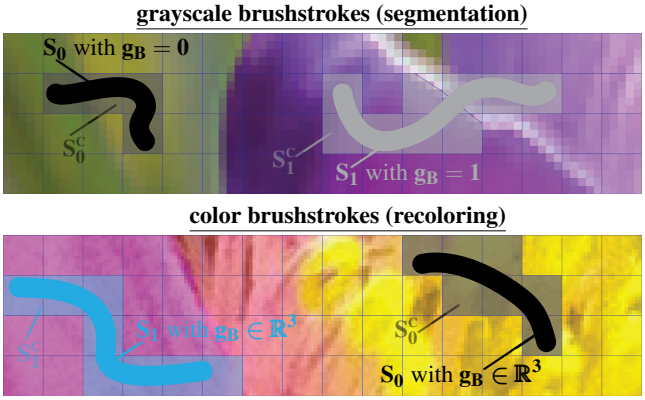


Figure 3: ***Interpreting Brushstrokes (Illustration)****: In the top part grayscale brushstroke values g_B are registered on a pixel level with S_0 (black) and S_1 (gray) and extended to adjacent clusters to form the sets S_0^c and S_1^c. The same concept is the shown at the bottom for brushes carrying color information. Note that for recoloring purposes a black brushstroke, i.e. $g_B = (0,0,0)^T$, is used to keep regions of the picture untouched.*

with $N_{C_i}^m$ being the set of indices of all clusters in the neighborhood of cluster C_i. The neighborhood of a cluster contains the m nearest clusters with respect to the euclidean distance of the clusters' corresponding 5D feature vectors $\tilde{I}^C$, i.e.

$$\begin{aligned} N_{C_i}^m &:= \{j_1,\dots,j_m\} \subset \{1,\dots,r\}\backslash\{i\} \\ &\text{with the property that for } k = 1,\dots,m \\ \|\tilde{I}_{C_i} - \tilde{I}_{C_{j_k}}\| &\leq \|\tilde{I}_{C_i} - \tilde{I}_{C_l}\| \;\forall l \in \{1,\dots,r\}\backslash\{j_1,\dots,j_m\} \end{aligned} \tag{6}$$

which implies $|N_{C_i}^m| = m$ $(i = 1,\dots,r)$.

Solving the optimization Problem (5) for each cluster C_i $(i = 1,\dots,r)$ yields optimal weights $\hat{w}_{ij}$ for $j \in N_{C_i}^m$. These weights represent the locally linear structure we were looking for. We store them in the $r \times r$ matrix $W := (\hat{w}_{ij})$ by explicitly setting $\hat{w}_{ij} := 0$ for $j \in \{1,\dots,r\}\backslash N_{C_i}^m$:

$$W_{ij} := \begin{cases} \hat{w}_{ij} & j \in N_{C_i}^m \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

Note that because of $r \gg m = |N_{C_i}^m|$ the matrix W is sparse.

2.2. Interactive Edit Propagation

Doing all our computations on the cluster structure instead of on the image pixel grid allows us to vastly reduce the dimensionality of the problem. All further design choices have been made with two goals in mind: Keeping computations efficient and being able to execute them completely on the cluster level. This section will derive our final edit propagation model, which is fully based on the cluster structure and can be solved by computing the solution to a linear system of equalities, which can be efficiently done on a GPU.

2.2.1. Interpretation of Brushstrokes

We register brushstrokes on a per pixel level and then map this information onto the corresponding clusters. The information carried by a specific kind of brushstroke can be any sort of feature, in this

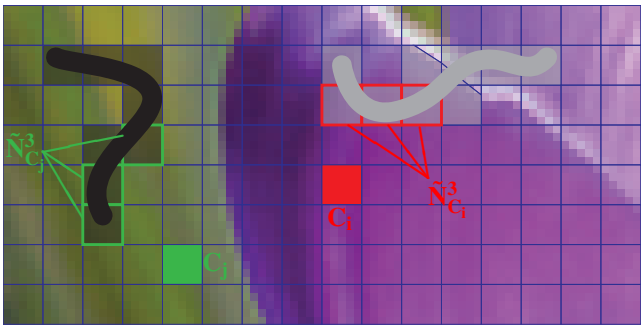


Figure 4: ***Brushstroke Clusters and Neighbors****: The image shows two clusters C_i and C_j, with their corresponding brushstroke cluster 3-neighborhoods $\tilde{N}_{C_i}^3$ and $\tilde{N}_{C_j}^3$. The criteria for choosing neighborhoods depends on the 5D image feature vectors I^C and is given in (6).*

paper we focus on grayscale values for image segmenation purposes and color values for image color editing. For each pixel x we set the corresponding brushsstroke intensity g as

$$g(x) := \begin{cases} g_B & \text{if marked by brushstroke with feature vector } g_B \\ 0_{\mathbb{R}^n} & \text{otherwise} \end{cases} \tag{8}$$

with $g_B \in \mathbb{R}^n$ being an appropriate feature vector, where we consider the cases $n = 1$ for segmentation or grayscale images and $n = 3$ for color editing. We define B to be the set of all l different occurring brushstroke feature vectors g_B, i.e.

$$B := \{g_{B_1},\dots,g_{B_l}\} \tag{9}$$

Having obtained the brushstroke information on a pixel grid level we want to assign appropriate intensity values to the individual clusters C_i $(i = 1,\dots,r)$ generated in Section 2.1.1. Due to the fact that clusters have been generated with a limit on their spacial extent, it is safe to assume that one cluster contains only marked pixels by one single kind of brushstroke. To avoid problems in the rare case of this assumption being violated, we just have to pick

$$g_{C_i} := g(y) \text{ with } y = \operatorname*{argmax}_{x \in C_i} \|g(x)\| \tag{10}$$

which means that we have the brushstroke intensity $g_{C_i} = 0_{\mathbb{R}^n}$ for clusters that do not contain any pixels marked by any brushstrokes. We also define

$$g^C := (g_{C_1},\dots,g_{C_r})^T \in \mathbb{R}^{r \times n} \tag{11}$$

to be the vector of all brushstroke cluster intensities.

Finally, we define S_i $(i = 1,\dots,|B|)$, with B being the set of all occurring brushstroke feature vectors defined in (9), to be the set of all pixels marked by a brushstroke with the same intensity and $S := \bigcup_{i=1,\dots,|B|} S_i$ the set of all marked pixels. Once again we extend the whole concept to clusters and obtain the corresponding sets of marked clusters S_i^C and $S^C = \bigcup_{i=1,\dots,|B|} S_i^C$. For an illustration of these definitions see Figure 3.

2.2.2. Brushstroke Propagation

As mentioned before, we have assigned the brushstroke intensity $g_{C_i} = 0_{\mathbb{R}^n}$ (cf. (10)) for clusters that do not contain any pixels marked by any brushstrokes. In order to influence those unmarked

clusters, especially if they are in close proximity to marked clusters, it makes sense to propagate brushstrokes in the image, as has been shown in [XYJ13]. The discussion about property (b) in Section 2.2.3 will also go into more details about that.

To do so we apply a Gaussian Mixture Model (GMM) like weighting process to the assigned brushstroke intensity values g^C analogous to [XYJ13]. Our weighting process is solely influenced by local color and position information given by the 5D feature vectors $\tilde{I}^C$ defined in (4). The individual Gaussian mixture weights are defined as

$$\alpha_{ij} := \exp(-\|\tilde{I}_{C_j} - \tilde{I}_{C_i}\|^2/\sigma^2) \quad (i,j = 1,\dots,r) \tag{12}$$

Given the brushstrokes and their corresponding features represented by g^C it makes sense to define the "probability" for an arbitrary brushstroke intensity distribution as

$$p\left(\bar{g}_{C_i}|g^C\right) := \frac{1}{\sum_{j\in\tilde{N}_{C_i}} \alpha_{ij}} \sum_{j\in\tilde{N}_{C_i}} \alpha_{ij} \exp\left(-\|g_{C_j} - \bar{g}_{C_i}\|^2/\sigma^2\right) \tag{13}$$

with $\tilde{N}^k_{C_i}$ being the set of the k nearest clusters (with respect to color and position) contained within the set of brushstroke clusters S^C being defined analogous to (6). For an illustration of $\tilde{N}^k_{C_i}$ see Figure 4.

It can be shown that the vector maximizing probability (13) is

$$\bar{g}_{C_i} := \frac{1}{\sum_{j\in\tilde{N}^k_{C_i}} \alpha_{ij}} \sum_{j\in\tilde{N}^k_{C_i}} \alpha_{ij}\, g_{C_j} \tag{14}$$

which we will consequently choose for brushstroke intensity values on clusters that have not been explicitly marked, i.e. for all clusters $C \notin S^C$. For a simpler matrix form notation we define the Gaussian mixture matrix G to represent the weighting process (14) by

$$G_{ij} := \begin{cases} \frac{\alpha_{ij}}{\sum_{l\in\tilde{N}^k_{C_i}} \alpha_{il}} & j \in \tilde{N}^k_{C_i} \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

which has a similar structure to the locally linear structure matrix W defined in (7) and hence is also sparse. This allows us to write (14) in matrix form by considering each channel of g^C, denoted by g^C_i $(i = 1,\dots,n)$, separately:

$$\bar{g}^C_i = G g^C_i \tag{16}$$

2.2.3. Propagating User Edits

When propagating brushstrokes we want the result $z^C := (z_{C_1},\dots,z_{C_r})^T \in \mathbb{R}^{r\times n}$ to have two basic properties:

(a) The propagated intensity values z^C should exhibit the same locally linear structure as the underlying image, which is given by the structure matrix W (cf. (5, 7)).
(b) Clusters in proximity (with respect to color and location) of marked clusters should take on similar intensity values z^C to the ones specified by the corresponding brushstroke, i.e. g^C (cf. (10, 11)).

In order to obtain the desired results we once again consider a minimization problem, this time with two energy terms representing properties (a) and (b) described above.

Let us consider property (a) first and assume that the optimal weights have already been computed according to (5) and are stored in matrix W according to (7). Then in order to obtain an output z^C, which exhibits the same locally linear structure, we just have to minimize the energy term

$$T_1(z^C) := \sum_{i=1}^{r} \left\| z_{C_i} - \sum_{j\in N^m_{C_i}} w_{ij}\, z_{C_j} \right\|^2 = \sum_{i=1}^{n} \left\| (E_r - W)\, z^C_i \right\|^2 \tag{17}$$

with $z^C_i = \{(z_{C_1})_i,\dots,(z_{C_r})_i\}$ $(i = 1,\dots,n)$, where $(z_C)_i$ denotes the i-th component of the vector $z_C \in \mathbb{R}^n$. Note that $n = 1$ corresponds to grayscale output and $n = 3$ to color output. Furthermore E_r is the $r \times r$ identity matrix and $N^m_{C_i}$ is the cluster m-neighborhood of cluster C_i as defined in (6).

We now turn our attention to property (b). A simple and straightforward way would be to consider the energy term

$$\sum_{C\in S^C} \|z_C - g_C\|^2 \tag{18}$$

which is considered in [CZZT12] (on a pixel level instead of clusters) and basically guarantees that the output z^C won't deviate much from the user input on clusters marked via brushstrokes given by the set S^C defined in Section 2.2.1. Obviously only clusters marked by brushstrokes are influenced by this energy term. Unfortunately, it turns out that in the case of sparse brushstrokes or a coarse cluster structure this term leads to problems when relying on the locally linear structure to spread the brushstroke information (property (a)). This has to do with the fact that the locally linear structure, which is always represented by a network built from the neighborhoods of every single cluster, might not connect all clusters with at least one of the brushstroke clusters. If a cluster is not connected via the aforementioned network, there is no information from the brushstroke available to propagate to this cluster.

To overcome this shortcoming we first propagate brushstrokes according to Section 2.2.2, which can be done separately for each channel with the propagation matrix G as shown in (16). When matching the output z_{C_i} with the propagated brushstroke intensity $\bar{g}_{C_i}$ at cluster C_i we want to weight this matching operation according to the cluster's feature vector $\tilde{I}_{C_i}$ (see definition (4)) in relation to the feature vectors of the brush neighborhood $\tilde{N}^k_{C_i}$ (see Figure 4). The weight for cluster C_i is set to

$$d_{C_i} := \frac{1}{|\tilde{N}_{C_i}|} \sum_{j\in\tilde{N}_{C_i}} \alpha_{ij} \tag{19}$$

with α_{ij} being defined in (12). This means that clusters with more similar color information or in closer proximity to brushstroke clusters will be given more incentive to match the propagated brushstroke information $\bar{g}^C$.

The corresponding diagonal matching matrix D is defined as

$$D = (D_{ij})_{i,j=1,\dots,r} \text{ with } D_{ii} := d_{C_i} \text{ and } D_{ij} := 0\ (i \neq j) \tag{20}$$

The overall matching process is then given by

$$T_2(z^C) := \sum_{i=1}^{n} \left(z^C_i - G g^C_i\right)^T D \left(z^C_i - G g^C_i\right) \tag{21}$$

Note that (18) is a special case of (21) and can be obtained by restricting D to be the indicator matrix for the brushstroke cluster set S^C and setting G to be the identity matrix.

By combining (17) and (21) we arrive at the minimization problem for the edit propagation step, i.e.

$$\underset{z^C \in \mathbb{R}^{r\times n}}{\operatorname{argmin}} \sum_{i=1}^{n} \lambda \left(z_i^C - G g_i^C\right)^T D \left(z_i^C - G g_i^C\right) + \left\| \left(E_r - W\right) z_i^C \right\|^2 \tag{22}$$

with weighting factor $\lambda > 0$, G being defined according to (15), D according to (20) and the locally linear structure weighting matrix W being defined in (7). As is immediately evident, this problem can be solved separately for each channel $i = 1, \ldots, n$.

3. Numerical Solution and Algorithm

In this section we show several numerical aspects of the individual parts of our overall algorithm. We will also sum up certain key aspects of algorithms introduced in other papers in order to provide a concise overview and introduce certain parameters our image editing pipeline depends on.

3.1. Solving the Pre-Calculation Problems

The first step in the pre-calculation step is to solve the L_0-minimization Problem (1) in order to detect relevant edges in the image. Following the ideas of [XLXJ11], this is done by splitting the problem into two separate subproblems. To do the separation process a binding energy term is introduced into Problem (1), i.e.

$$\min_{z,v \in \mathrm{BV}(\Omega)} \sum_{x \in \Omega} \left(\lambda_{L_0} \|v(x)\|_0 + \beta \|v(x) - \nabla z(x)\|^2 + \|z(x) - I(x)\|^2 \right) \tag{23}$$

This problem is then solved by alternating optimization in the two optimization variables z and v. Thresholding the gradient of the output image yields the desired edge map.

Algorithm 1 L0 Edge Detection

```
procedure DETECTEDGES(image, λ_{L_0}, thresh)
    set β := 0.03, κ := 2, maxIter := 100, β_max := 10000, I := image and initialize z := I
    while i < maxIter && β < β_max do
        v ← argmin Problem (23) with z fixed
        z ← argmin Problem (23) with v fixed
        set β = κ · β,  i ← i + 1                          ▷ increase beta continuously
    edgeMap := (||∇z|| > thresh)                           ▷ threshold gradient map
    return (edgeMap, z)
```

Having the information of relevant edges in the image available allows us to build appropriate clusters. In our case appropriate means locally coherent and edge aware. One additional requirement for fast computation is once again parallelizability, which can be easily obtained by splitting the image in a regular grid and computing clusters in each cell in a separate thread, which also yields local coherence of a cluster by restricting it to one grid cell (cf. Figure 1).

Algorithm 2 Create Clusters

```
procedure CREATECLUSTSERS(edgeMap, cellSize)
    build regular grid = {cell_i} with cell_i of size cellSize^2 and set clusters := ∅    ▷ initialize
    for i = 1, ..., |grid| do                                  ▷ this can be executed in parallel
        while cell_i \ clusters ≠ ∅ do
            pick pixel x ∈ cell_i \ clusters and create new cluster C = {x}
            while C does not extend over edge in edgeMap do
                pick pixel x ∈ cell_i \ clusters according to edgeMap and set C ← C ∪ {x}
            set C ← C \ {x} and clusters ← clusters ∪ C
    return clusters
```

From the cluster structure we can then extract the corresponding locally linear structure by solving Problem (5). As this problem is a constrained convex quadratic problem we are able to obtain one solution (there might be infinitely many) by solving the corresponding KKT conditions, which are first order optimality conditions. Due to the special structure of the problem this boils down to just solving a sparse linear system of equalities. The procedure is described in detail in [SR00], which is unfortunately lacking a precise derivation of the aforementioned linear system of equations from the KKT conditions.

Algorithm 3 Locally Linear Structure Computation

```
procedure LINEARSTRUCTURE(clusters, image, m, τ)
    compute 5D features Ī^C on clusters according to (4)
    for i = 1,...,r do                                     ▷ can be done in parallel
        compute m-neighborhood N_{C_i}^m according to (6)
        ŵ_i ← argmin Problem (5)
    from ŵ_i (i = 1, ..., r) create matrix W according to (7)
    return W
```

Algorithm 1, Algorithm 2 and Algorithm 3 constitute the full pre-calculation part of our pipeline, hence they only have to be executed once in the beginning.

3.2. Interactive Edit Propagation Algorithm

The processing of the user input starts with registering brushstrokes and propagating them. This procedure has already been described in detail in Section 2.2.1 and Section 2.2.2. The corresponding algorithm is trivial and in a short form looks like

Algorithm 4 Brushstroke Enhancement

```
procedure ENHANCEBRUSHSTROKES(strokes, clusters, τ)
    register graylevels g according to (8) from strokes
    g^C ← extend g to clusters following (10)
    α_ij ← compute Gaussian weights according to (12)
    G ← with α_ij obtain the Gaussian mixture matrix from (15)
    D ← with α_ij obtain the diagonal matching matrix from (19) and (20)
    return (g^C, G, D)
```

The final step in our pipeline is the edit propagation itself. This means that we have to solve Problem (22) separately for each channel $i = 1, \ldots, n$. Taking a closer look reveals that it is an unconstrained quadratic convex problem, which can easily be solved by computing the solution to its first order optimality conditions. As the problem is quadratic, the first order optimality conditions are just a huge system of linear equations:

$$\begin{aligned} \lambda D \left(\hat{z}_i^C - G g_i^C\right) + \left(E_r - W\right)^T \left(E_r - W\right) \hat{z}_i^C &= 0_{R^r} \\ \left(\lambda D + \left(E_r - W\right)^T \left(E_r - W\right)\right) \hat{z}_i^C &= \lambda D G g_i^C \end{aligned} \tag{24}$$

Note that we have successfully vastly reduced the number of equations by transferring the problem from a per pixel level to our cluster structure. The edit propagation step is essentially the reason for doing all the mentioned speedup measures (precomputation steps, especially clustering, and brushstroke adjustments), because it is the part of our pipeline that has to be executed every time the user input is changed. Also important to note is that due to the local aspect of the neighborhoods $N_{C_i}^m$ the matrix W exhibits diagonal block structure, which means that $\left(E_r - W\right)^T \left(E_r - W\right)$ is still a sparse matrix, which makes solving system (24) more efficient in terms of

computation time and memory usage. As the interactive edit propagation part of the algorithm will be called every time the user input changes (live editing), it makes sense to precompute this matrix and store it in GPU memory. The edit propagation algorithm is

Algorithm 5 Edit Propagation

1: **procedure** EDITPROPAGATION(g^C, G, D, W, z^C_{init})
2: use precomputed sparse matrix $(E_r - W)^T (E_r - W)$
3: **for** $i = 1, \ldots, n$ **do**
4: solve equation (24) for $\hat{z}^C_i$ with iterative method using initial solution z^C_{init}
5: **return** $\hat{z}^C$

Overall, Algorithm 4 and Algorithm 5 are called every time the user input in form of brushstrokes changes. As the drawing of brushstrokes by the user is a continuous incremental process, the user input will only deviate by a little bit in its new iteration. This means that it is a good idea to reuse the most recently computed solution $\hat{z}^C$ as the initial solution z^C_{init} for Algorithm 5 and use an iterative linear solver.

4. Evaluation and Results

In this section we show the application of our algorithm in various use cases, measure computation times in a realistic user input driven environment and compare our approach to state of the art methods in the field. All statistics were measured on an Intel Core i5-4460, 3.20 GHz and 16 GB RAM PC with a NVIDIA GeForce GTX 970 graphics card running Windows 10 64-bit with code foundation written in C++ CUDA, running completely on the GPU.

As described in Section 2 and Section 3 our full clustering based framework depends on the choice of certain parameters. The good news is that most of these parameters can be fixed and all the others only have to be adjusted for the specific kind of application (like segmentation, recoloring, etc.) or if there is a certain user preference for the behavior of brushstrokes. For the following experiments we fixed $\lambda_{L_0} := 0.015$, $\beta := 0.03$, $\kappa := 0.03$, $maxIter := 100$, $\beta_{max} := 10000$ (Algorithm 1), $cellSize := 10$ (Algorithm 2) and $\sigma := 0.1$, which turned out to be good choices in general. Setting brush neighbors $k := 3$ (Algorithm 4) is a good choice in general, too. This leaves only a few parameters that are worth mentioning for actual variation by the user. One of the parameters users might want to adjust depending on the image is τ (Algorithm 3 and Algorithm 4), which balances the influence of color and location information on the final output with respect to the given brushstrokes. Theoretically it might make sense to choose different values for τ in Algorithm 3 and Algorithm 4, but in practice we found this to be unnecessary. In most cases we chose $\tau := 0.2$. In addition to that we do have the threshold parameter $thresh$ (Algorithm 1) and the number of neighbors m (for computing the locally linear structure), which balance visual quality and computation time, depending on the image at hand. Once again, $thresh := 0.01$ and $m := 30$ are good choices for most cases. Depending on the amount of brushstrokes, users might also want to change the parameter σ, which determines a brushstroke's influence on its surroundings.

Our approach can be used for segmenting images by using a discrete set of graylevel brushstrokes. In the notation of this paper this means that the brushstroke intensities $g_{C_i} \in \mathbb{R}$ $(i = 1, \ldots, r)$ (cf. Section 2.2.1). In Figure 5 we show segmentation in multiple distinct regions with sparse and dense input and compare results with the approach of Chen et al. [CZZT12]. Due to the additional brushstroke propagation step (cf. Section 2.2.2) our approach shows better performance with sparser input and regions that are not explicitly marked in comparison to [CZZT12]. Additionally, due to the lack of fine details in the input image (no hair, etc.), our clustering process has no detrimental influence on the visual quality of the segmented output at all.

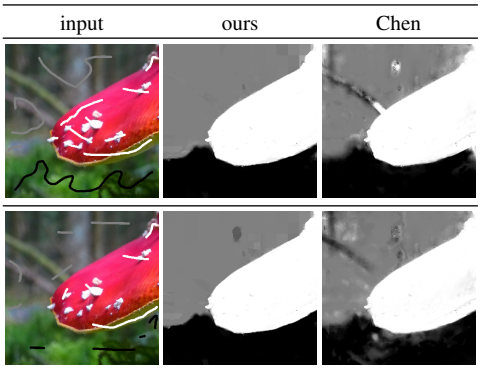


Figure 5: ***Image Segmentation:*** *Results with dense (first row) and sparse (second row) input compared to the approach of Chen et al. [CZZT12].*

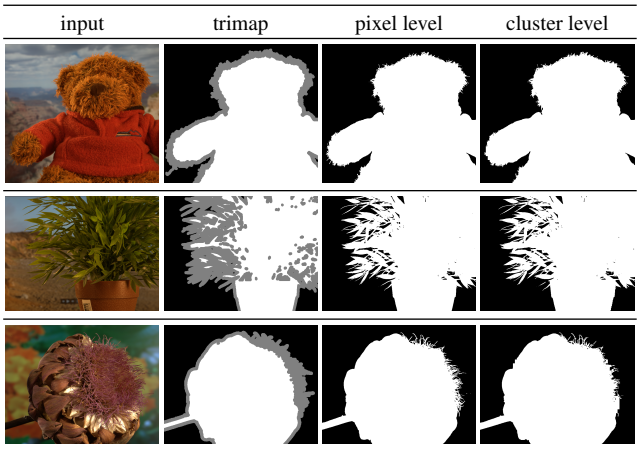


Figure 6: ***Image Matting:*** *Our approach for classical image matting on the alphamatting dataset [RRW*09]. Pixel level means the output of our approach without clustering. The results show the thresholded output of our algorithm. On cluster level, problems occur when the intensity change from foreground to background is almost nonexistent.*

The presented approach is also applicable in the classic case of image matting using a trimap as input as shown in Figure 6. To incorporate trimaps into our workflow one simply has to specify the black area and white area as brushstrokes of the corresponding graylevel, so the only unmarked area is the gray region of the trimap. It should be obvious that the clustering component of our approach won't be able to deliver much of a speedup as the trimap already specifies vast regions that do not have to be computed and the gray area usually exhibits rather delicate detail, which automatically leads to a fine cluster structure according to Section 2.1.1.

Figure 7: ***Image Recoloring:*** *Colored brushstrokes mark the pixels that will be replaced with the specified color. Black strokes mark regions that should be kept untouched. Pixel level is our approach without clustering.*

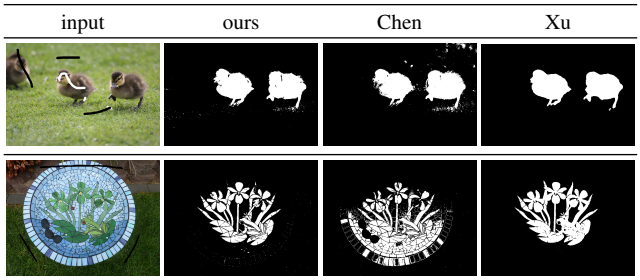
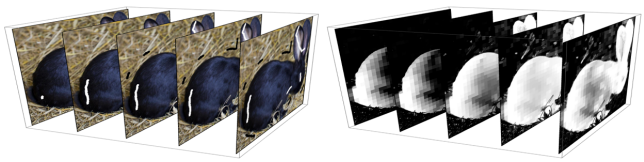


Figure 8: ***Qualitative Comparison:*** *Two challenging scenarios taken from [XYJ13]. Both only provide sparse input and overlapping color vectors. Global brush information propagation (used by our approach and even more extensively in the approach of Xu) helps greatly in these scenarios. Pixel level means the output of our approach without clustering. The results show the thresholded output of the algorithms.*

We can also achieve high quality output in case of image recoloring as shown in Figure 7. Recoloring is done by choosing color brushes, i.e. $g_{C_i} \in \mathbb{R}^3$ $(i = 1,\dots,r)$ (cf. Section 2.2.1), with black user input corresponding to regions that should be kept untouched. As usual, we solve Problem (22) for $i = 1,\dots,3$ and given g^C but additionally solve the problem as a graylevel segmentation problem with all color information treated as white, i.e. $g_B = 1$, and black as $g_B = 0$. The segmentation part yields a mask that we apply to the recolored result to reintroduce the original pixel information from the input image in the background. We see our approach applicable in a scenario where the user wants immediate feedback. According to the results in Figure 7 a coarser cluster structure usually only lacks the more detailed nuances.

Amongst the qualitatively best approaches for interactive image segmentation are the approaches of Chen et al. [CZZT12] and especially Xu et al. [XYJ13]. So for further output quality comparison we show results compared to the aforementioned approaches in



detailed timings for low resolution case ($\approx 75K$ pixels)

Testcase	ours				Chen				Xu (estimated)			
	first	min	max	mean	first	min	max	mean	first	min	max	mean
spring_flower	0.149	0.008	0.146	0.029	0.390	0.008	0.377	0.159	1.191	0.027	1.146	0.484
teddybear	0.115	0.006	0.125	0.018	0.491	0.011	0.462	0.128	1.503	0.033	1.362	0.390
leaves	0.110	0.083	0.109	0.096	1.035	0.257	0.389	0.323	3.213	0.810	1.197	1.004
rabbit	0.087	0.006	0.178	0.046	1.058	0.013	0.494	0.305	3.156	0.036	1.482	0.903

mean timings for different image resolutions

Testcase	≈75K pixels			≈200K pixels			≈400K pixels		
	ours	Chen	Xu	ours	Chen	Xu	ours	Chen	Xu
spring_flower	0.029	0.159	0.484	0.060	0.487	1.451	0.084	1.289	3.770
teddybear	0.018	0.128	0.390	0.036	0.620	1.909	0.045	2.833	8.323
leaves	0.096	0.323	1.004	0.089	0.886	2.667	0.198	2.857	8.625
rabbit	0.046	0.305	0.903	0.108	0.779	2.323	0.157	4.130	11.950

Figure 9: ***Computation Time:*** *Evaluation of the interactive part presented in Section 2.2.3. We use whole sequences of continuous brushstroke input to simulate real user input. The first table shows execution times in seconds for each data set (four images with approx. 75K pixels each, thresh for Algorithm 1 set to 0.01 and number of neighbors m set to 30) divided in first (first time executing the edit propagation), min (minimum execution time in sequence), max (maximum execution time in sequence) and mean (mean value throughout sequence without first strokes). Using the same parameters, the second table shows mean timing values (without first strokes) throughout the sequences for varying image resolutions.*

Figure 8. Results from Xu et al. [XYJ13] are qualitatively superior, but need much more computation time (cf. Figure 9) in contrast to our realtime approach presented in this paper.

At last we show essential computation time measurements. To achieve relevant measurements we simulate continuous user input by passing a sequence of brushstrokes, constituting incremental stroke updates, to our algorithm and evaluate different sequences on four images (spring flower, teddybear leaves, rabbit), which have been shown in different figures in this section. Figure 9 shows an excerpt of the rabbit sequence and the central computation time table for the interactive part of our algorithm. The pre-calculation part always takes about a few seconds depending on the size and structure of the input image. We compare our approach to the approach of Chen et al. [CZZT12], which we have implemented in the same way as our algorithm. As memory consumption is much higher for the latter approach than for ours we evaluate the algorithms on medium sized images (spring_flower: 300×200 pixels, rabbit: 300×277 pixels, leaves: 300×223 pixels, teddybear: 300×270 pixels) to guarantee a fair evaluation, although our approach does scale much better with increasing image resolution. Additionally, we give an estimate of the computation time for the approach of Xu et al. [XYJ13] by taking into account that its computational complexity is identical to applying our approach on a pixel level three to four times (according to the information provided by the authors of the paper).

5. Conclusion

In this paper we have shown how to create a very fast image editing pipeline by using a locally linear image structure on clusters to extract information from an input image. We have also shed some light on how to actually propagate user input using the aforementioned data and sacrifice little to none of the visual quality by using the cluster structure. For future work we are interested in considering further ideas of the approach of Xu et al. [XYJ13] and how to incorporate them in an efficient way in our existing framework. We believe that making the brushstroke propagation considered in Section 2.2.2 dependent on the actual output of the edit propagation is one key component for an even higher quality result.

Acknowledgment

The research leading to these results has received funding from the European Unions Seventh Framework Programme FP7/2007-2013 under grant agreement no. 256941, Reality CG.

References

[AP08] AN X., PELLACINI F.: Appprop: All-pairs appearance-space edit propagation. *ACM Trans. Graph. 27*, 3 (Aug. 2008), 40:1–40:9. 2

[CZZT12] CHEN X., ZOU D., ZHAO Q., TAN P.: Manifold preserving edit propagation. *ACM Trans. Graph. 31*, 6 (Nov. 2012), 132:1–132:7. 2, 5, 7, 8

[FFLS08] FARBMAN Z., FATTAL R., LISCHINSKI D., SZELISKI R.: Edge-preserving decompositions for multi-scale tone and detail manipulation. In *ACM SIGGRAPH 2008 Papers* (New York, NY, USA, 2008), SIGGRAPH '08, ACM, pp. 67:1–67:10. 2

[LAA08] LI Y., ADELSON E., AGARWALA A.: ScribbleBoost: Adding Classification to Edge-Aware Interpolation of Local Image and Video Adjustments. *Computer Graphics Forum* (2008). 2

[LFUS06] LISCHINSKI D., FARBMAN Z., UYTTENDAELE M., SZELISKI R.: Interactive local adjustment of tonal values. *ACM Trans. Graph. 25*, 3 (July 2006), 646–653. 2

[LJH10] LI Y., JU T., HU S.-M.: Instant Propagation of Sparse Edits on Images and Videos. *Computer Graphics Forum* (2010). 2

[LLW04] LEVIN A., LISCHINSKI D., WEISS Y.: Colorization using optimization. *ACM Trans. Graph. 23*, 3 (Aug. 2004), 689–694. 2

[LSS09] LIU J., SUN J., SHUM H.-Y.: Paint selection. *ACM Trans. Graph. 28*, 3 (July 2009), 69:1–69:7. 2

[MX14] MA L.-Q., XU K.: Efficient manifold preserving edit propagation with adaptive neighborhood size. *Computers & Graphics 38* (2014), 167–173. 2

[PL07] PELLACINI F., LAWRENCE J.: Appwand: Editing measured materials using appearance-driven optimization. *ACM Trans. Graph. 26*, 3 (July 2007). 2

[RRW*09] RHEMANN C., ROTHER C., WANG J., GELAUTZ M., KOHLI P., ROTT P.: A perceptually motivated online benchmark for image matting. In *Proceddings of the IEEE Conference on Computer Vision and Pattern Recognition* (2009). 7

[RS00] ROWEIS S. T., SAUL L. K.: Nonlinear dimensionality reduction by locally linear embedding. *SCIENCE 290* (2000), 2323–2326. 2, 3

[SR00] SAUL L. K., ROWEIS S. T.: *An Introduction to Locally Linear Embedding*. Tech. rep., test, 2000. 6

[XLJ*09] XU K., LI Y., JU T., HU S.-M., LIU T.-Q.: Efficient affinity-based edit propagation using k-d tree. *ACM Transactions on Graphics 28*, 5 (2009), 118:1–118:6. 2

[XLXJ11] XU L., LU C., XU Y., JIA J.: Image smoothing via l0 gradient minimization. *ACM Trans. Graph. 30*, 6 (Dec. 2011), 174:1–174:12. 3, 6

[XYJ13] XU L., YAN Q., JIA J.: A sparse control model for image and video editing. *ACM Trans. Graph. 32*, 6 (Nov. 2013), 197:1–197:10. 2, 5, 8, 9