Chapter 14
Computer-Assisted Repurposing of Existing Animations

Daniel Sýkora and John Dingliana

14.1 Introduction

Paper and pencil are the only tools that a skilled artist needs to create a fascinating world of cartoon animation. With these tools the artist has complete freedom, as there are no limitations apart from the size of the paper and length of the lead. However, this freedom is tempered by the enormous effort and time needed to complete the artwork especially in the case of colorful animation where hundreds of painted drawings are required.

In recent times, computer-assisted 3D animation systems have become very popular as they can save a great deal of manual work. Here, the key advantage is that the system already knows the structure and motion of an animated object, therefore the final artwork is simply created by an automated rendering algorithm without any additional effort. As a result, everything can be easily manipulated and modified. However, the compromise is that the artist loses a part of their freedom and expressivity. Moreover, the creation of fully consistent 3D models can become very tedious when compared to simple 2D drawing.

The aim of this chapter is to present a set of tools that enable ease of modification, manipulation, and rendering similar to 3D animation systems, whilst preserving the expressivity and simplicity of the original hand-drawn animation. To achieve this, it is necessary to infer a part of the structural information hidden in the sequence of hand-drawn images, namely the partitioning into meaningful segments, their topology variations, depth ordering, and correspondences. Since this inference can be very ambiguous and cannot be fully automated, we let the artist provide a couple of rough hints that make this problem tractable.

D. Sýkora (✉)
FEE, DCGI, CTU in Prague, Karlovo nám. 13, 121 35 Praha 2, Czech Republic
e-mail: sykorad@fel.cvut.cz

J. Dingliana
Trinity College Dublin, College Green, Dublin 2, Ireland
e-mail: John.Dingliana@scss.tcd.ie

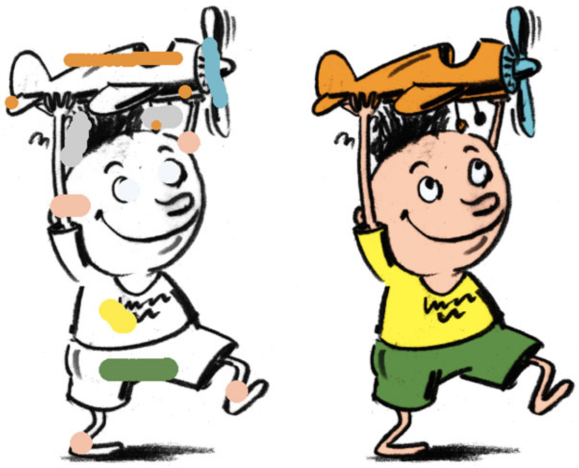


Fig. 14.1 Interactive segmentation of a hand-drawn image using LazyBrush [33]. The algorithm finds an optimal labelling based on a set of roughly placed positional constraints—scribbles. It automatically handles small gaps in outlines (note the small subaxillary gap), correctly maintains anti-aliasing, and is not sensitive to imprecise placement of scribbles (e.g., the large brown scribble over the plane). Reproduced with kind permission from Blackwell Publishing Ltd. © Anifilm + © EG & Blackwell. Used with permission

The rest of the chapter is organized as follows. First we introduce an interactive tool which enables quick partitioning of the image into a set of meaningful parts, Sect. 14.2. These play a crucial role in the depth assignment and layering framework, Sect. 14.3, which can further help to simplify deformation and retrieval of correspondences between animation frames, Sect. 14.4. Finally, we demonstrate how reconstructed structural information and correspondences can help to solve more complex problems such as auto-painting, example-based synthesis, temporally coherent texture mapping, or 3D-like shading, Sect. 14.5.

14.2 Segmentation

This section presents *LazyBrush*—an interactive tool for segmenting hand-draw images in various drawing styles [33], see Fig. 14.1. It addresses common limitations of area selection tools used in professional ink-and-paint systems (usually called *magic wand* or *bucket fill*). These are typically based on a variant of the flood-fill algorithm which works well for images containing large flat regions separated by continuous outlines. However, hand-drawn images are typically more complex and thus tedious manual corrections are necessary to obtain clean segmentation. Typical problems that can arise when one wants to segment a hand-draw image using flood-fill based tools are depicted in Fig. 14.2. These problems feature even in recent advanced image segmentation [3, 10] and colorization algorithms [20, 27, 30], see Fig. 14.3.

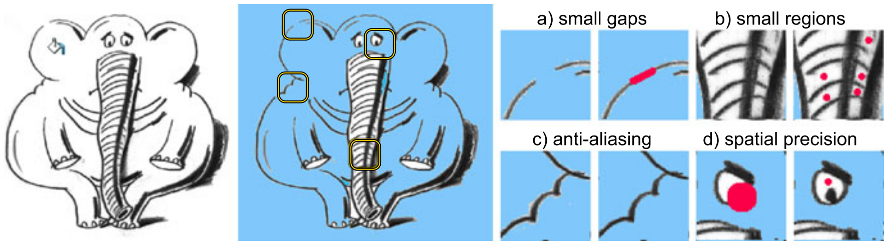


Fig. 14.2 Typical problems of flood-fill based tools when applied to a hand-drawn image: (**a**) small gaps cause leakage, the user has to retrieve them and draw a closure, (**b**) small regions require the user to perform many detailed mouse clicks, (**c**) anti-aliasing is not preserved well due to intensity thresholding mechanisms, the user has to tune the threshold to obtain better results, however, one single value is typically not sufficient for the whole image, (**d**) the user has to move the mouse pointer exactly inside the region of interest. © Anifilm. Used with permission

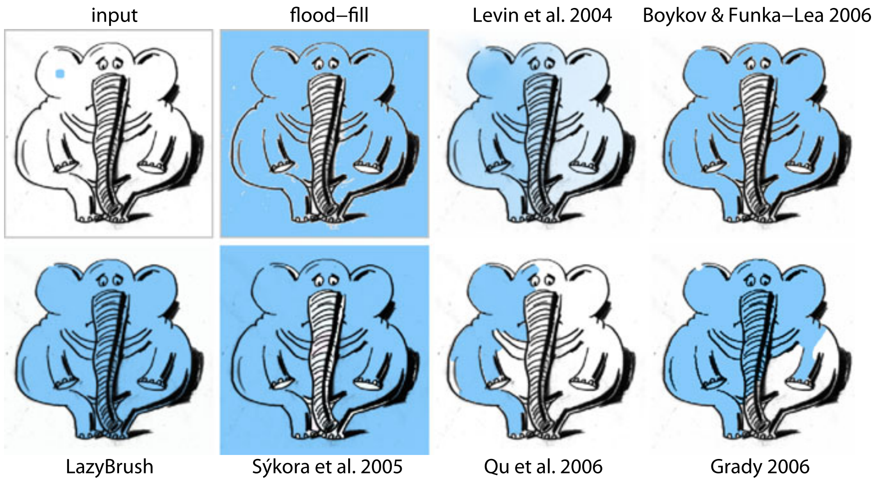


Fig. 14.3 LazyBrush [33] versus flood-fill and modern image segmentation and colorization algorithms: Levin et al. [20] suffers from leakage, Sýkora et al. [30] does not handle small gaps and small regions, Qu et al. [27] get stuck in a local minima, small regions when not arranged in a repetitive hatching pattern need to be handled individually, anti-aliasing is not supported, Grady [10] tends to produce weird boundaries and does not handle anti-aliasing, Boykov and Funka-Lea 2006 [3] have problems with small regions as well as anti-aliasing. Reproduced with kind permission from Blackwell Publishing Ltd. © Anifilm + © EG & Blackwell. Used with permission

LazyBrush uses a popular interaction metaphor called *scribbles*, see Fig. 14.1, which was originally used to perform interactive colorization of gray-scale images [20] and segmentation of photographs [3]. Instead of a single point click inside the region of interest, the user specifies a set of constrained pixels upon which the algorithm resolves the final labelling. Compared to previous approaches [3, 20], LazyBrush scribbles are not necessarily meant to be hard constraints, i.e., the user can overdraw the region of interest. The algorithm will recognize such inaccuracies and try to produce better labelling.

14.2.1 Problem Formulation

Similarly to recent advanced image segmentation and colorization techniques [3, 10, 20, 27] LazyBrush formulates segmentation as an energy minimization problem. It defines a new energy function that is custom tailored to hand-drawn images and thus can overcome issues depicted in Fig. 14.2 and Fig. 14.3.

14.2.1.1 Energy Function

As an input we consider a gray-scale image I consisting of pixels P in a 4-connected neighborhood system N. Each pixel $p \in P$ has an intensity $I_p \in \langle 0, 1 \rangle$. In addition to this, the user marks a subset of pixels using scribbles S. Each scribble $s \in S$ has a specific label ℓ_s taken from a set of possible labels L, see Fig. 14.1 left. The aim is to find an optimal labelling ℓ^*, i.e., the label-to-pixel assignment, see Fig. 14.1 right, that minimizes the following energy:

$$E(\ell) = \sum_{\{p,q\} \in N} V_{p,q}(\ell_p, \ell_q) + \sum_{p \in P} D_p(\ell_p) \tag{14.1}$$

where the *smoothness term* $V_{p,q}$ represents the energy of label discontinuity between two neighbor pixels p and q (i.e., when $\ell_p \neq \ell_q$ otherwise $V_{p,q} = 0$), and *data term* D_p the energy of assigning a label ℓ to a pixel p.

14.2.1.2 Smoothness Term

As the aim is to maintain anti-aliasing, discontinuities between two labels are preferred to appear at pixels p where the intensity I_p is low, i.e., inside dark outlines, see Fig. 14.4(A) left. Therefore we need to set $V_{p,q} \propto I_p$. This is a fundamental difference from standard image segmentation techniques [3, 10] where the aim is to push segment boundaries to pixels with maximal gradient. Such a setting is undesirable in our scenario as it reveals discontinuities at soft edges, Fig. 14.4(A) right.

Next we need to favor compact hole-free regions, see Fig. 14.4(B) left, therefore it is necessary to set $V_{p,q} > 0$, otherwise outlines with zero intensity will not influence the minimum of Eq. (14.1) therefore can easily produce disconnected holes in the final segmentation, Fig. 14.4(B) right. We avoid this by always adding 1 to the smoothness term, i.e., $V_{p,q} = 1 + I_p$. However, when the original image contains long creeks such simple additions can lead to unintended shortcuts, see Fig. 14.4(C) right. To suppress them, discontinuities going through the white pixels should have

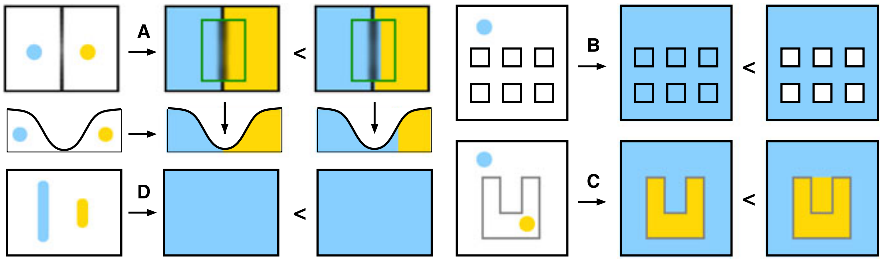


Fig. 14.4 The energy function used in LazyBrush satisfies the following inequalities: (A) a discontinuity inside the outline always has lower energy than a discontinuity on the edge (bottom inset depict intensity profile), (B) length of the discontinuity counts, i.e., compact hole-free regions have lower energy, (C) shortcut through the white areas has higher energy than a discontinuity along a creek even if the contrast of the outline is low, (D) soft scribbles respect rule of majority, i.e., the label of a scribble which occupies the largest area inside a homogeneous region will prevail. Reproduced with kind permission from Blackwell Publishing Ltd.

very high energy K. As the aim is to have overall energy of a shortcut higher than a sum of energies over a long creek, a good estimate for K can be a perimeter of I.

Another source of shortcuts is low contrast between homogeneous areas and outlines visualized by light gray in Fig. 14.4(C). This can happen, e.g., in unprocessed scans of soft pencil drawings. To overcome this, a nonlinear mapping that enhances contrast is required. We use a gamma correction: $V_{p,q} = 1 + K \cdot I_p^\gamma$, with, e.g., $\gamma = 5$. Similar preprocessing of input intensities is required when segmenting grayscale images. In this case outlines need to be emphasized first (e.g., using the negative response of Laplacian-of-Gaussian filter [30], see also Sect. 5.2.2) and then segmentation can be performed using the LazyBrush algorithm (cf. Fig. 14.18, for details see [33]).

To summarize the previous discussion, the smoothness term $V_{p,q}$ is defined as follows:

$$V_{p,q}(\ell_p, \ell_q) = \begin{cases} 1 + K \cdot I_p^\gamma & \text{for } \ell_p \neq \ell_q \\ 0 & \text{otherwise} \end{cases} \tag{14.2}$$

where K is the perimeter of I and $\gamma = 5$.

14.2.1.3 Data Term

In recent image segmentation and colorization algorithms, the data term D_p is usually set to reflect some image-based prior such as intensity [3] or a repetitive hatching pattern [27]. However, repetitive hatching or intensity variations are not typical for hand-made drawings and even if they are present, correspondences between intensity/pattern and a meaningful segment in the image are rare. To address this fact, LazyBrush uses only a user-driven data term. Among other properties, this setting ensures that all label segments are always connected to their initial scribbles. A similar approach is also used in [10], however, here a key difference is that LazyBrush

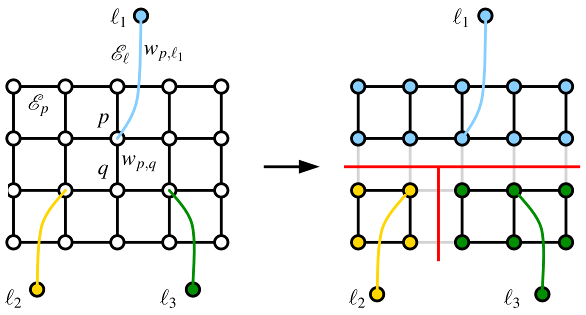


Fig. 14.5 Multiway cut—a basic structure of a graph $\mathcal{G}$ (*left*): pixels P (*white dots*), label terminals L (*color dots*), pixel edges $\mathcal{E}_p$ with weight $w_{p,q}$ (*black lines*), and links to label terminals $\mathcal{E}_\ell$ with weight $w_{p,\ell}$ (*color lines*). The resulting multiway cut and corresponding labelling of pixels (*right*). Reproduced with kind permission from Blackwell Publishing Ltd.

does not necessarily assume that all user-defined scribbles serve as hard constraints. It introduces a new rough positional constraint—*soft scribble,* which preserves the so called *rule of majority*, i.e., within a homogeneous area a label whose scribble occupies the majority of pixels will prevail, see Fig. 14.4(D). In other words, the overall energy Eq. (14.1) should be lower for the left labelling in Fig. 14.4(D) even thought all pixels under the yellow scribble have not received its label. This behavior can be accomplished using the following data term:

$$D_p(\ell_p) = \begin{cases} K & \text{no scribble} \\ 0.95 \cdot K & \text{soft scribble} \\ 0 & \text{hard scribble} \end{cases} \tag{14.3}$$

where K is the perimeter of the image I (i.e., the energy of a discontinuity at white pixels). For a derivation of this setting and a more detailed discussion about the rule of majority see [33].

14.2.2 Problem Solution

Once the energy function Eq. (14.1) is defined we can proceed to its minimization. Since the value of the smoothness term $V_{p,q}$ depends only on the case where two neighbor pixels have different labels, our energy satisfies the *Potts* model [26]. As shown in [5], minimizing such energy is equivalent to solving a *multiway cut* problem on a certain undirected graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ where $\mathcal{V} = \{P, L\}$ is a set of vertices and $\mathcal{E} = \{\mathcal{E}_p, \mathcal{E}_\ell\}$ a set of edges, see Fig. 14.5.

Vertices $\mathcal{V}$ consist of pixels P and terminals L. Each pixel $p \in P$ is connected to its 4 neighbors via edges $\mathcal{E}_p$ having weight equal to smoothness term $w_{p,q} = V_{p,q}$ for the case $\ell_p \neq \ell_q$. There are also auxiliary edges $\mathcal{E}_\ell$ that connect terminals L to marked pixels. Each $\mathcal{E}_\ell$ has a weight $w_{p,\ell} = K - D_p(\ell)$ (hard scribbles have $w_{p,\ell} = K$ and soft $w_{p,\ell} = 0.05 \cdot K$).

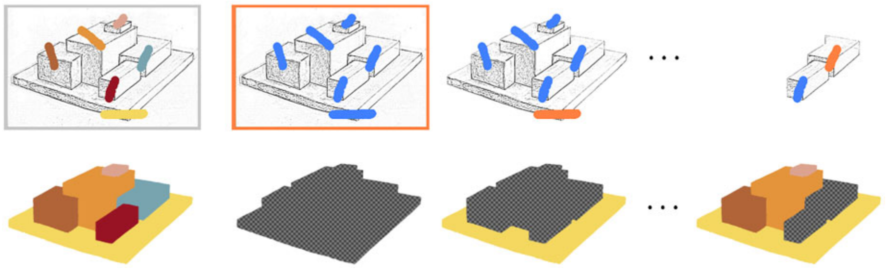


Fig. 14.6 Greedy approximation to multiway cut in progress—computing binary max-flow/min-cut subproblems on gradually reducing graphs (*top*), corresponding mask of already labelled pixels (*bottom*, checkerboard indicates unlabelled pixels). Reproduced with kind permission from Blackwell Publishing Ltd. © Ondřej Sýkora + © EG & Blackwell. Used with permission

A multiway cut with two terminals is equivalent to a max-flow/min-cut problem for which efficient algorithms exist [4, 12]. However, for three or more labels the problem becomes NP-hard [6]. Nevertheless, in practice a simple greedy approximation can quickly deliver a solution which is visually close to the global minimum. It is based on a sequence of binary solutions. The algorithm selects an arbitrary label as a first terminal and all other labels as a second terminal. Then it solves a binary max-flow/min-cut problem and removes a part of the graph associated to the first terminal. The same operation is repeated on a reduced graph with a reduced set of labels until there are only two different labels, see Fig. 14.6.

14.3 Adding Depth

In this section, an extension of the LazyBrush algorithm is presented that enables quick addition of depth information into hand-drawn images [34]. It is motivated by perceptual studies that have tried to understand how humans reconstruct depth from a single image [17, 18]. These studies show that humans typically fail to specify absolute depth values, however, are much more accurate in telling whether some part of the object is occluded by another and vice versa. Therefore the aim is to avoid inputs requiring knowledge of absolute depth and instead use a set of sparse *depth equalities* that are much easier to specify, see Fig. 14.7.

14.3.1 Depth from Depth Inequalities

Supposing that we have already partitioned the input image into a set of meaningful regions, see Fig. 14.8 left, and specified a set of depth inequalities which indicate their relative ordering in depth Fig. 14.8 middle. Such input can be represented as an oriented graph $G(V, E)$ where vertices V correspond to regions (red dots in Fig. 14.8) and oriented edges E represent depth inequalities (green arrows in Fig. 14.8). Now the task is to assign a depth value into each vertex $v \in V$ so

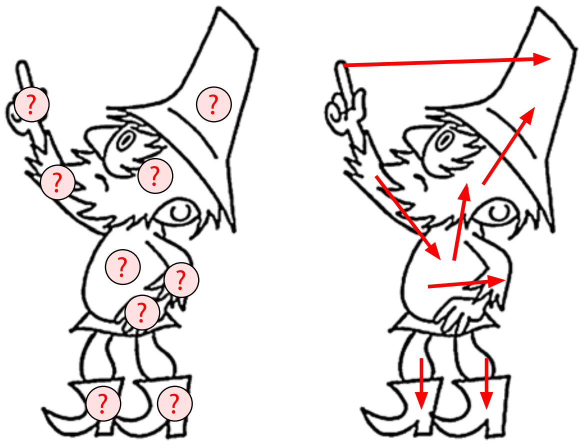


Fig. 14.7 If we ask a human to tell us what are the absolute depths of regions denoted by *question marks* (*left*) they immediately start to think in terms of pairwise above/under relationships (*right*) from which they reconstruct absolute depths. This tedious process can be automated so that the user can directly specify only these pairwise relationships (depth inequalities) and the system will resolve absolute depths automatically. © UPP & DMP. Used with permission

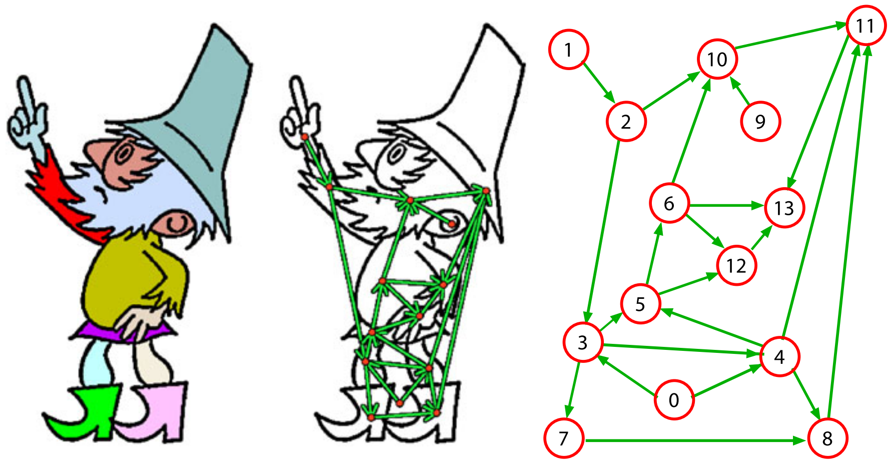


Fig. 14.8 Depth from depth inequalities—a user specifies meaningful segmentation (*left*) and a set of depth inequalities (*middle*). Based on this input, a graph is built (*right*) and its vertices are enumerated according to their topological order. Reproduced with kind permission from Blackwell Publishing Ltd. © UPP & DMP + © EG & Blackwell. Used with permission

that it satisfies all specified depth inequalities E (see the graph in Fig. 14.8 and the resulting depth map in Fig. 14.10).

This task is equivalent to a graph theoretical problem called *topological sorting* [15]. It can be solved by a simple algorithm, the input of which is graph $G(V, E)$, a set of vertices having no incoming edges S, and an empty set L. The algorithm repeats the following steps:

```
while $S \neq \emptyset$ do
    $S := S - \{n\}$   and   $L := L \cup \{n\}$
    for $\forall m \in V$ having edge $e : n \rightarrow m$ do
        $E := E - \{e\}$
        if $m$ has no other incoming edges then
            $S := S \cup \{m\}$
    endfor
endwhile
if $E \neq \emptyset$ then
    $G$ has at least one oriented cycle else $L$ contains topologically sorted nodes.
```

A topologically sorted list of vertices L is returned only if the graph G does not contain oriented loops. This situation can happen when a new depth inequality is added in the wrong direction or when the partitioning of the input image is insufficient. The system will inform the user about this problem and ask them to change the direction or refine the partition of the input image so that the loop is removed. This leads to an interactive process where segmentation and specification of depth inequalities is interchanged until the desired depth assignment is reached.

14.3.2 Outline-to-Region Assignment

The LazyBrush algorithm places segment boundaries inside the outline therefore it is not clear to which segment the outline actually belongs. This knowledge is crucial for applications where a precise extraction of individual parts is necessary. The task is equivalent to the *figure-ground separation* problem [25], which is non-trivial and requires additional semantic knowledge. However, an important part of this knowledge is already encoded in the absolute depth ordering. This information is sufficient to produce good results with only minor artifacts, see Fig. 14.9.

First we need to estimate local thickness of outlines using two distance maps [8]: D^1 computed from the boundaries of regions being expanded, red color in Fig. 14.9(b) and D^2 from all other regions, blue color in Fig. 14.9(b). Pixels where $D^1_p = D^2_p$ form a medial axis from which we can propagate estimates of outline thickness t to all other pixels, Fig. 14.9(c).

This propagation can be understood as a variant of *diffusion curves* [24] (see Sect. 8.2.3) and so can be formulated as a solution to the Laplace equation: $\nabla^2 t = 0$ with the following boundary conditions ($q \in N_p$):

$$\text{Dirichlet: } t_p = 2D^1_p \quad \Longleftrightarrow \quad D^1_p = D^2_p$$

$$\text{Neumann: } t'_{pq} = 0 \quad \Longleftrightarrow \quad D^1_p = 0 \quad \text{or} \quad D^2_p = 0$$

This formulation leads to a sparse system of linear equations which is solvable using simple Gauss–Seidel iterations or some more advanced techniques such as [13].

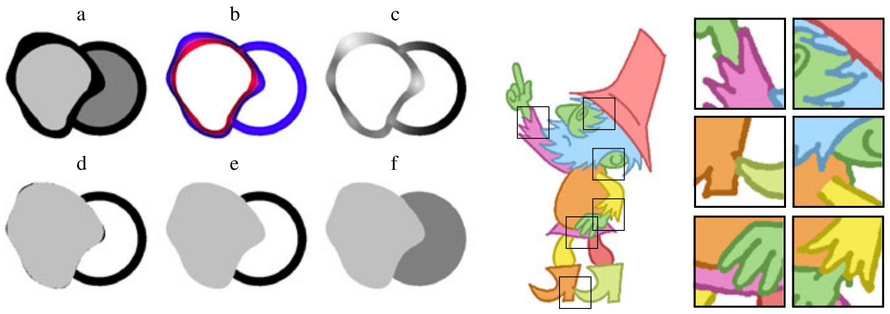


Fig. 14.9 Outline-to-region assignment—a synthetic example (*left*): (**a**) input depth map with dark outlines, (**b**) medial axis obtained using two distance maps computed from the active region (*red*) and all other regions (*blue*), (**c**) propagation of outline thickness from the medial axis to all other outline pixels, (**d**) outline-to-region assignment based on the local estimation of outline thickness, (**e**) filling in small gaps, (**f**) final expanded depth map. A practical example (*right*): several minor artifacts are depicted in selected zoom-ins. Reproduced with kind permission from Blackwell Publishing Ltd. © UPP & DMP + © EG & Blackwell. Used with permission

With the estimation of outline thickness we can expand the region to pixels where $d^1_p < t_p$, Fig. 14.9(d), and fill in small gaps by removing connected components whose size is below a predefined threshold, Fig. 14.9(e). The expanded region is then removed from the depth map and the same process is applied to all other remaining regions in a front-to-back order to obtain the resulting assignment, Fig. 14.9(f).

14.3.3 Smooth Depth Transitions

When the absolute depths and outline-to-region assignments are known, we can produce smooth depth transitions in areas where depth discontinuities between segments were enforced due to a depth assignment process based on topological sorting (note the depth discontinuity between body and arm in Fig. 14.10).

As we already know where the original depth inequalities were placed, we can use their endpoints to define a set of point constraints $U_\circ$ (see red dots in Fig. 14.10) from which we can smoothly propagate absolute depth values to the rest of the image. Values are taken from $\hat{d}$, which denotes the initial depth map produced by LazyBrush and the topological sorting algorithm described in Sect. 14.3.1, see Fig. 14.10 left.

This is again a problem similar to diffusion curves and can be solved using the Laplace equation: $\nabla^2 d = 0$ with the following boundary conditions ($q \in N_p$):

$$\text{Dirichlet: } d_p = \hat{d}_p \quad \Longleftrightarrow \quad p \in U_\circ$$

$$\text{Neumann: } d'_{pq} = 0 \quad \Longleftrightarrow \quad \hat{d}_p \neq \hat{d}_q \wedge I_p < \tau$$

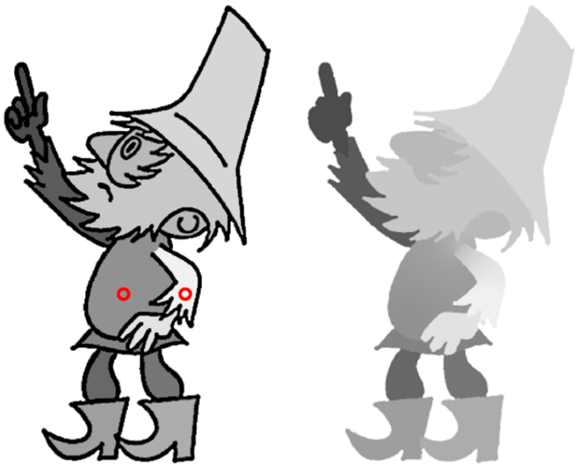


Fig. 14.10 Enforcing smooth depth transitions—the initial depth map (*left*) produced by Lazy-Brush (Sect. 14.2) and the topological sorting algorithm (Sect. 14.3.1) contains artificial depth discontinuities at pixels where the body and arm connects. This artifact can be removed (*right*) by calculating a smooth transition between the endpoints (*red dots*) of the original depth inequality which was used to specify depth ordering of these two regions (depicted in Fig. 14.8). Reproduced with kind permission from Blackwell Publishing Ltd. © UPP & DMP + © EG & Blackwell. Used with permission

Here Neumann conditions enforce zero derivative only at pixels where the depth discontinuity in the initial depth map $\hat{d}$ is valid, i.e., lies inside the outline (τ is a threshold for the intensity of outlines).

14.4 Deformation

Image deformation tools are invaluable for computer assisted production of hand-drawn cartoon animations since they allow for quick animation prototyping, example-based synthesis and can help to obtain rough correspondences between individual animation frames. Recently the as-rigid-as-possible deformation model [1] has become very popular due to its ability to produce plausible deformations with only little user intervention [11], see Fig. 14.11.

14.4.1 Rigid Square Matching

In this section we describe a simple yet effective algorithm called *rigid square matching* [37], which enables interactive as-rigid-as-possible shape manipulation and can be easily extended to perform fully automatic or supervised image registration [32].

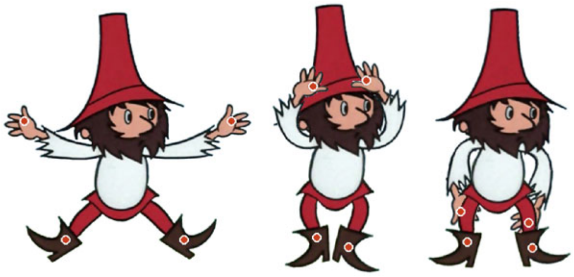


Fig. 14.11 As-rigid-as-possible shape manipulation—the user selects a few control points (*red dots*) on the pre-segmented image (*left*), drags them to a desired location (*middle* and *right*), and the algorithm deforms the image in a way that the rigidity of the original shape is preserved. © UPP & DMP. Used with permission

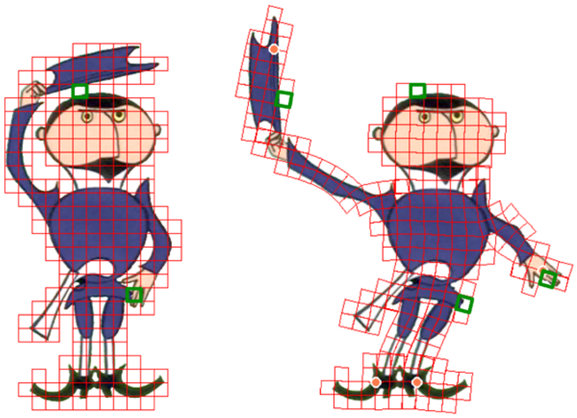


Fig. 14.12 Rigid square matching—the original image is embedded within a square lattice whose connectivity respects the initial segmentation and depth layering provided by the user (*left*). To avoid gluing of disconnected parts during the deformation (*right*) there can exist multiple collocated squares with different connectivity (examples are denoted in *green*). © UPP & DMP. Used with permission

An initial step of the rigid square matching algorithm is to embed the input image into a *control lattice*, see Fig. 14.12, consisting of interconnected squares whose topology respects segmentation and depth layering specified by the user. Each square is assigned a bitmap with corresponding pixels and a depth map to resolve the local layering problem when displaying self-occluded poses, see Fig. 14.11. This solution has an advantage over the triangulation used in [11] as it is much easier to implement and there is no need to approximate shape boundaries using piecewise linear segments.

To avoid gluing due to insufficient resolution of the control lattice, the algorithm allows several collocated squares with different connectivity, green squares in Fig. 14.12. For segments glued together due to occlusion, see hands or boots in Fig. 14.13(a), the user can additionally specify a subset of depth inequalities, blue arrows in Fig. 14.13(a), that will mark corresponding depth discontinuities as tears, Fig. 14.13(b). This modification is important namely for the image registration scenario where it can help to improve the accuracy of the final registration, Fig. 14.13(c).

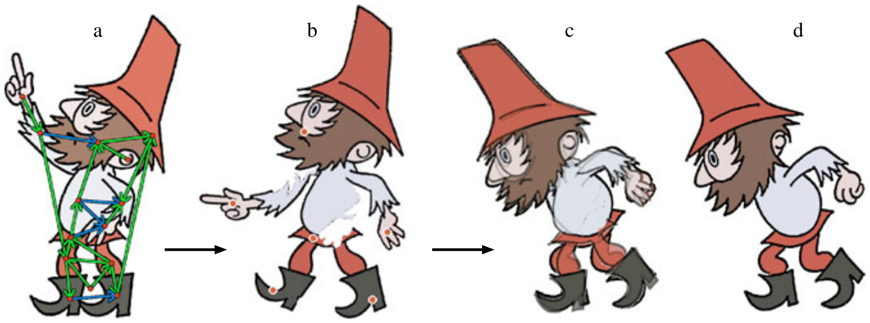


Fig. 14.13 Introducing tears—a subset of depth inequalities (*blue arrows*) is selected (**a**) to specify tears at corresponding depth discontinuities (**b**). This can help to improve the accuracy of the image registration algorithm (**c**) when a source image (**a**) is registered to a target image (**d**). Reproduced with kind permission from Blackwell Publishing Ltd. © UPP & DMP + © EG & Blackwell. Used with permission

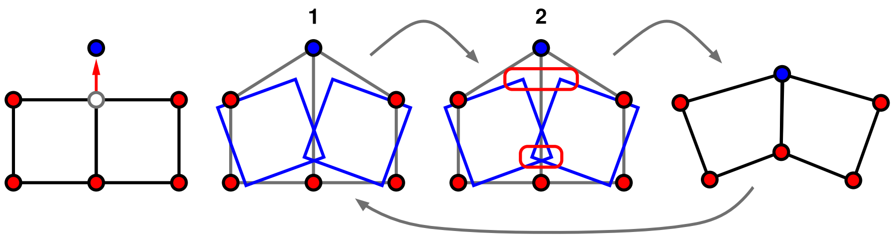


Fig. 14.14 Schematic overview of two simple steps repeated by the rigid square matching algorithm: (1) computation of rigid transformation for each square and (2) moving vertices towards centroid computed from their instances in connected squares

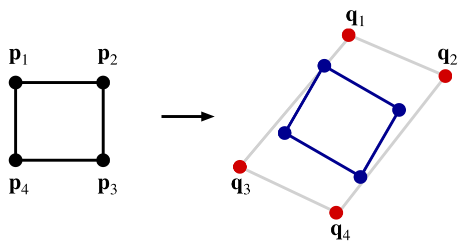
14.4.1.1 Algorithm

When the control lattice is created, the user selects a subset of its vertices called *control points* and moves them to arbitrary locations to define the desired deformation, see Fig. 14.11 and Fig. 14.12. The aim of the rigid square matching algorithm is to move all remaining vertices on the lattice so that deformation of the corresponding squares will be as close as possible to a rigid transformation, i.e., just rotation $\mathbf{R}^*$ and translation $\mathbf{t}^*$. This is accomplished by iterating the following two simple steps, see Fig. 14.14:

1. For each square, compute optimal rigid transformation $(\mathbf{R}^*, \mathbf{t}^*)$ and use it to transform its vertices.
2. Move each vertex to the centroid of its transformed instances in all interconnected squares.

Step (1) can be computed as follows:

The aim is to find the optimal rigid transformation ($\mathbf{R}^*$,$\mathbf{t}^*$) that moves the vertices of the original square $\mathbf{p}_i$ so that the sum of squared distances to the corresponding vertices in the desired pose $\mathbf{q}_i$ is minimized:



$$\left(\mathbf{R}^*, \mathbf{t}^*\right) = \arg\min_{\mathbf{R},\mathbf{t}} \sum_i |\mathbf{R} \cdot \mathbf{p}_i + \mathbf{t} - \mathbf{q}_i|^2 \tag{14.4}$$

A simple closed form solution exists to this least square problem in 2D [28]. It can be shown that the optimal rigid transformation ($\mathbf{R}^*$,$\mathbf{t}^*$) always moves the centroid $\mathbf{p}_c$ of the source vertices to the centroid $\mathbf{q}_c$ of the target vertices. When we align source and target vertices in a way that their centroids are in the origin (i.e., $\hat{\mathbf{p}}_i = \mathbf{p}_i - \mathbf{p}_c$ and $\hat{\mathbf{q}}_i = \mathbf{q}_i - \mathbf{q}_c$) then the optimal rotation $\mathbf{R}^*$ can be computed as follows:

$$\mathbf{R}^* = \frac{1}{\mu} \sum_i \begin{pmatrix} \hat{\mathbf{p}}_i \\ \hat{\mathbf{p}}_i^{\perp} \end{pmatrix} \begin{pmatrix} \hat{\mathbf{q}}_i^T & \hat{\mathbf{q}}_i^{\perp T} \end{pmatrix} \tag{14.5}$$

where

$$\mu = \sqrt{\left(\sum_i \hat{\mathbf{q}}_i \hat{\mathbf{p}}_i^T\right)^2 + \left(\sum_i \hat{\mathbf{q}}_i \hat{\mathbf{p}}_i^{\perp T}\right)^2} \tag{14.6}$$

T denotes transposition, and the operator $\perp$ denotes the perpendicular vector, i.e.: $(x, y)^{\perp} = (y, -x)$. Once the rotation matrix $\mathbf{R}^*$ is known, the translation vector $\mathbf{t}^*$ can be computed directly:

$$\mathbf{t}^* = \mathbf{q}_c - \mathbf{R}^* \cdot \mathbf{p}_c \tag{14.7}$$

14.4.2 As-Rigid-As-Possible Image Registration

The knowledge of correspondences between individual hand-drawn animation frames is crucial for many applications described in this chapter. However, obtaining them automatically is a challenging task. The problem is that each animation frame is unique and when compared to a previous frame, it typically undergoes a large amount of free-form deformation and notable change in appearance, see Fig. 14.15.

Popular computer vision techniques based on local similarity (*SIFT keys* [21]) or global context (*shape contexts* [2]) typically fail on such images since they rely on unique local features or stable global configurations, which are common in photographs but rare in hand-drawn images. A more powerful approach—*deformable image registration* based on discrete optimization [9, 29] allows retrieval of correspondences even in the presence of local/global free-form deformation, however, it

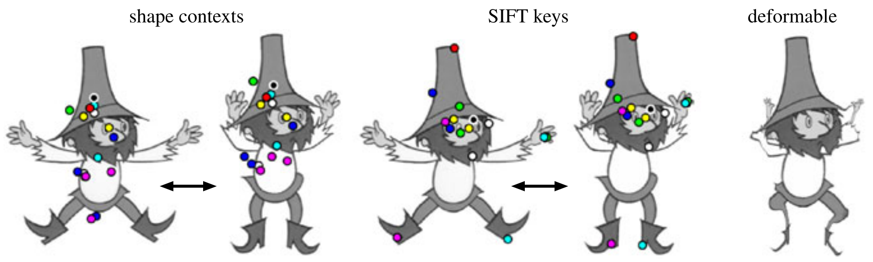


Fig. 14.15 Automatic retrieval of correspondences between two hand-drawn images—SIFT keys [21] or shape contexts [2] fail as there are only a few distinct local features and the global context is not preserved. Deformable image registration [9] cannot handle large displacements. © UPP & DMP. Used with permission

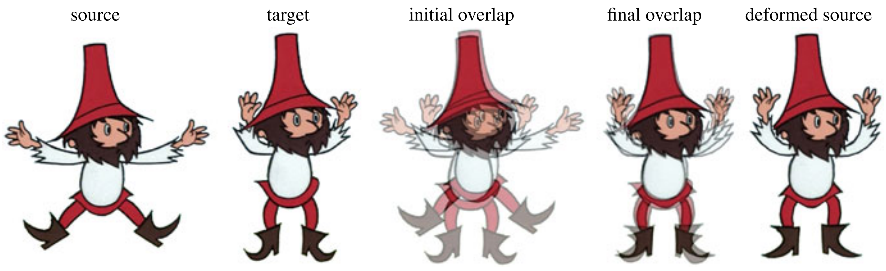


Fig. 14.16 Result of fully automatic as-rigid-as-possible image registration—the aim is to register source and target image with depicted initial overlap, after several push/regularize iterations the source is deformed so that it approximately matches the target (cf. final overlap and deformed source). © UPP & DMP. Used with permission

becomes computationally intractable for larger displacements due to an exponentially increasing state space.

In this section we describe a simple yet effective extension of the rigid square matching algorithm that enables fully automatic or supervised deformable image registration [32]. As the deformation model employed enforces local rigidity and respects the original shape articulation, the algorithm is more robust to larger displacements, see Fig. 14.16. Moreover, due to its iterative nature, it allows the user to inspect the registration process and intervene when necessary.

The algorithm iterates two simple steps until a stable configuration is reached, see Fig. 14.17:

1. **Push** all vertices to locations with minimal visual difference.
2. **Regularize** control lattice using the rigid square matching algorithm.

The aim of the push phase is to find a new location for each vertex on the embedding lattice that minimizes visual difference in its local neighborhood. To do this we can utilize a simple *block matching* algorithm, which guarantees a globally optimal shift within a predefined search area.

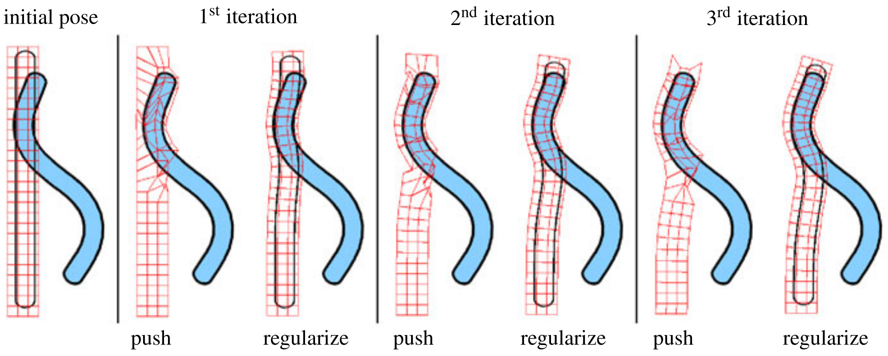
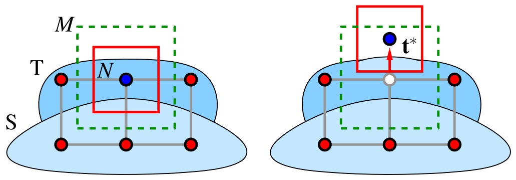


Fig. 14.17 As-rigid-as-possible image registration in progress—in each iteration two steps are executed: first all vertices are pushed towards locations with minimal visual difference (*left*) and then the shape is regularized using the rigid square matching algorithm (*right*). For clarity, the source shape is filled with a transparent color and the control lattice is visualized

Formally, the aim is to find an optimal shift vector $\mathbf{t}^*$ within a local search area M that minimizes the sum of square differences over a neighborhood N, i.e.:



$$\mathbf{t}^* = \arg\min_{\mathbf{t}\in M} \sum_{\mathbf{p}\in N} \left|\mathrm{S}(\mathbf{p}) - \mathrm{T}(\mathbf{p}-\mathbf{t})\right|^2 \qquad (14.8)$$

where S denotes the source and T the target image. Note that in spite of shift optimization, the overall image registration algorithm is not limited to pure translation, since S is slightly deformed during the regularization phase and local neighborhoods of vertices gradually adapt to more complicated deformations, see Fig. 14.17.

In addition to the block matching algorithm the user can also intrude into the push phase by specifying their own positional constraints similarly to the shape manipulation scenario, Fig. 14.11, or just quickly guide the process by simply dragging selected vertices towards desired locations. The key difference from the stand-alone rigid square matching algorithm is that during the push phase *all* vertices are moved (not only those representing user-defined constraints). As can be seen in Fig. 14.17 this leads to temporarily inconsistent configurations, however, in the regularization phase, vertex positions are immediately relaxed by a couple of rigid square matching iterations that enforce local rigidity and make the overall shape consistent.

14.5 Applications

The techniques described in previous sections can now be used as basic building blocks in various practical applications enabling repurposing of existing or creation

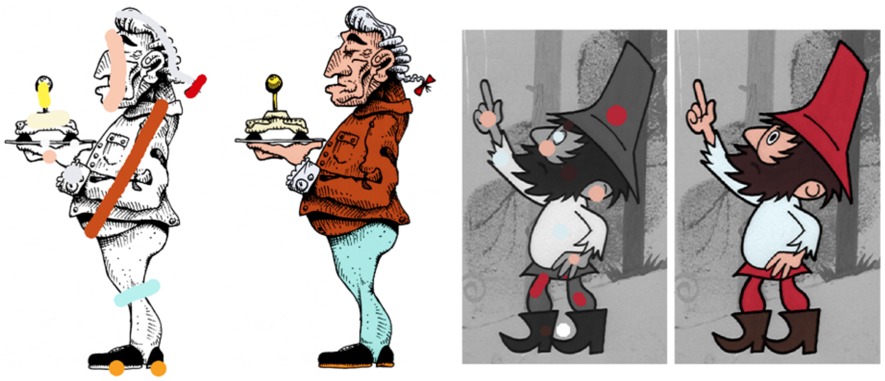


Fig. 14.18 Examples of interactive painting (*left*) and colorization (*right*) of hand-drawn images in various drawing styles using the LazyBrush algorithm. Reproduced with kind permission from Blackwell Publishing Ltd. © Lukáš Vlček + © UPP & DMP + © EG & Blackwell. Used with permission

of new hand-drawn cartoon animations with a look that is distinct from traditional techniques.

14.5.1 Painting, Colorization and Texture Mapping

The first straightforward application of segmentation and registration is *painting* and *colorization*, see Fig. 14.18 and also Sect. 17.5. Here desired colors or color components are assigned to the resulting segments and, in each pixel, multiplied/combined with the original gray-scale intensity. The greedy multi-label segmentation algorithm presented in Sect. 14.2.2 is fast enough to enable interactive response when working in PAL resolution.

To avoid repeated specification of scribbles for all animation frames, as-rigid-as-possible image registration, Sect. 14.4.2, can be used to register the first frame to the following frame, transfer the scribbles, and use the LazyBrush algorithm to obtain the segmentation, see Fig. 14.19. As the LazyBrush algorithm is robust to imprecise positioning of scribbles, small mismatches in the registration are allowed. However, for scenes where detailed painting is required (e.g., many small regions with different colors), the user may need to specify additional correction scribbles to keep the segmentation consistent.

Instead of a single color, the user can also specify a texture and make the region filling more visually rich. However, in contrast to a single color there is an additional problem: the texture should follow the motion and/or deformation of its corresponding regions in the subsequent frames to preserve temporal coherency. This can be problematic in hand-drawn animation as it is typically impossible to obtain one-to-one correspondence between individual frames, see Sect. 14.4.2.

Fortunately, as noted in [39] the human visual system tends to focus more on visually salient regions, while devoting significantly less attention to other, less vi-

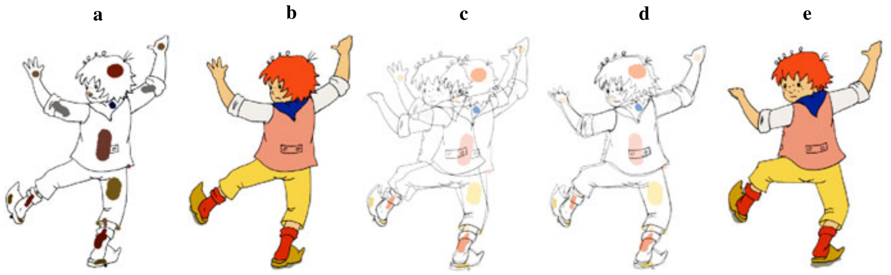


Fig. 14.19 Auto-painting—color scribbles (**a**) are transferred from already painted (**b**) to yet unpainted frames (**c**) using as-rigid-as-possible image registration algorithm (**d**). LazyBrush is then utilized to compute the final painting (**e**). Reproduced with kind permission from ACM. © Anifilm + © ACM. Used with permission

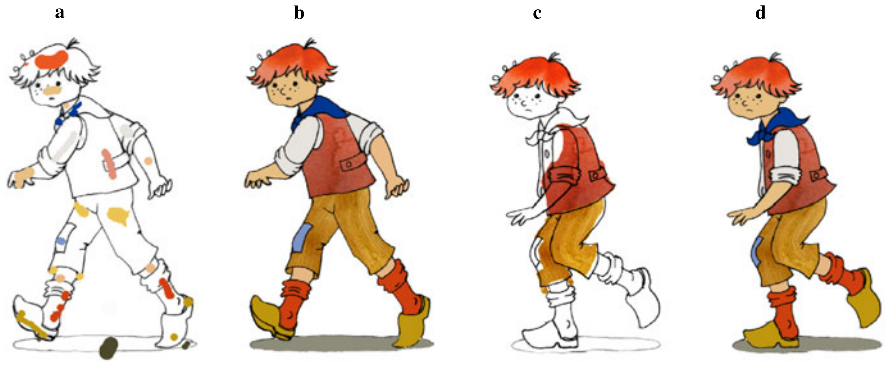


Fig. 14.20 Texture mapping with approximate temporal coherence—scribbles used to paint the first frame (**a**), textures were applied to selected regions (**b**), texture transfer to a new frame using a deformation field obtained by as-rigid-as-possible image registration algorithm (**c**), final painting of the new frame (**d**), small gaps were filled using extrapolation of texture coordinates. Reproduced with kind permission from ACM. © Anifilm + © ACM. Used with permission

sually important, areas. In hand-drawn animations contours are the salient features. Textures are typically less salient and thus attract considerably less attention [36]. Exploiting this property, an illusion of temporal coherent animation can be created using only rough correspondences obtained by an as-rigid-as-possible image registration algorithm [35], see Fig. 14.20.

14.5.2 Simulation of 3D-Like Effects

In this section we describe techniques which allow artists to simulate 3D-like effects common for computer-generated movies entirely in the 2D domain without the need to reconstruct and render a 3D model, see Fig. 14.23(d). They are based on

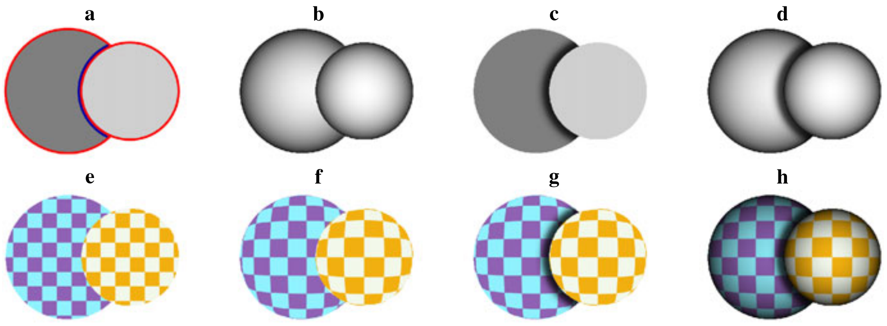


Fig. 14.21 Simulation of 3D-like effects: (**a**) depth map with Dirichlet (*red*) and Neumann (*blue*) boundary conditions, (**b**) Lumo shading, (**c**) simulation of ambient occlusion, (**d**) simulation of ambient occlusion with Lumo shading, (**e**) texture mapping using flat UV coordinates, (**f**) texture rounding based on shading, (**g**) texture rounding with ambient occlusion, (**h**) texture rounding with shading and ambient occlusion

the segmentation and depth obtained using the algorithms described in Sect. 14.2 and Sect. 14.3.

14.5.2.1 Ambient Occlusion

Ambient occlusion is a popular technique that can approximate smooth light attenuation on diffuse surfaces caused by occlusion. Its key advantage is that it can enhance the perception of depth in the image [19]. In our setting with known segmentation and depth order this effect can be simulated by unsharp masking the depth buffer [22] or by simply superimposing a stack of regions with blurred boundaries in a back-to-front order, see Fig. 14.21(c) and Fig. 14.23(c).

14.5.2.2 Shading

Another popular technique that can profit from knowledge of segmentation and depth is *Lumo* [14]. The method approximates the normal field inside a region using the 2D normals computed on its boundaries. The main idea is that on the silhouette of an object the normal component in the viewing direction is always equal to zero, hence the normal is completely specified by its x and y components. Furthermore, the gradient of the image intensity is orthogonal to the silhouette, giving exactly the required normal components. This simple rule holds only when the target shape contains silhouette pixels, i.e., for interior strokes, depth discontinuities should be taken into account. In the original method the user had to trace over the region boundaries and then manually specify an over-under assignment map to produce correct results.

In this section we describe a new formulation of Lumo [35], which exploits algorithms described in this chapter in order to obtain similar results with much less

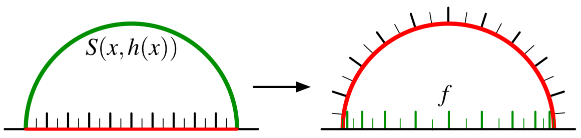


Fig. 14.22 Texture rounding (1D example): linearly interpolated texture coordinates on the surface S and their back projection f to the plane

effort, see Fig. 14.21(b) and Fig. 14.23(c). The formulation is analogous to diffusion curves. The resulting normal field is obtained by solving the Laplace equation $\nabla^2 f = 0$ (where f is either the x or y component of the normal vector n) with the following boundary conditions, see Fig. 14.21(a):

$$\begin{aligned} &\text{Dirichlet: } f_p = d'_{pq} \quad &\Longleftrightarrow \quad & d_p > d_q \\ &\text{Neumann: } f'_{pq} = 0 \quad &\Longleftrightarrow \quad & d_p < d_q \end{aligned} \tag{14.9}$$

where q is a neighboring pixel to p, d'_{pq} is the derivative of the depth map at pixel p in the direction pq and f'_{pq} is the derivative of the normal component (n_x or n_y). This leads to a sparse system of linear equations with two different right hand sides (n_x and n_y). As in the original method n_z is computed using n_x and n_y components via the sphere equation:

$$n_z = \sqrt{1 - n_x^2 - n_y^2} \tag{14.10}$$

14.5.2.3 Texture Rounding

Values of n_z can be further utilized to simulate a *texture rounding* effect, i.e., when the curvature of the surface generates an area distortion and causes the texture to scale. *Parallax mapping* [16] is typically used to simulate this scaling [38], however, the disadvantage here is that it does not preserve UV coordinates at region boundaries thus produces noticeable texture sliding when used in animation [35]. This artifact can be avoided by interpolating texture coordinates on a virtual 3D surface $S = (x, y, h(x, y))$, where the height h is taken from the z component of the extrapolated normal: $h(x, y) = n_z(x, y)$, see Fig. 14.22.

Such interpolation can be computed directly in 2D by solving the *inhomogeneous Laplace equation*: $\nabla_w^2 f = 0$ where ∇_w^2 is the *Laplace–Beltrami* operator, which measures actual distances on the surface S. This yields another large sparse system of linear equations, now with an irregular Laplacian matrix where the weights w_{ij} between pixels i and j are computed as the inverted length of the edge connecting their corresponding 3D vertices on S:

$$w_{ij} = \frac{1}{\sqrt{1 + (h_i - h_j)^2}} \tag{14.11}$$

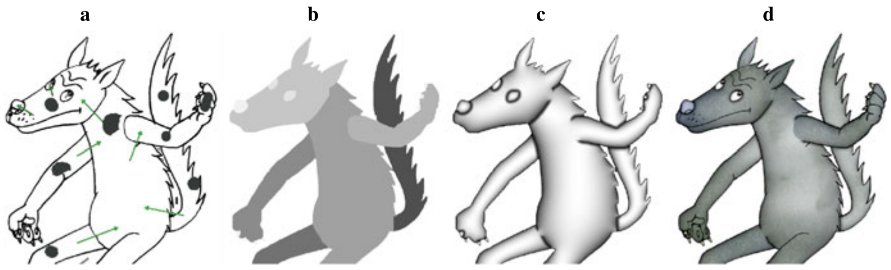


Fig. 14.23 Simulation of 3D-like effects (real example): (**a**) original image with LazyBrush scribbles and depth inequalities, (**b**) depth map, (**c**) ambient occlusion and Lumo shading, (**d**) final composition: original image, textures, ambient occlusion, Lumo shading and texture rounding. Reproduced with kind permission from ACM. © Anifilm + © ACM. Used with permission

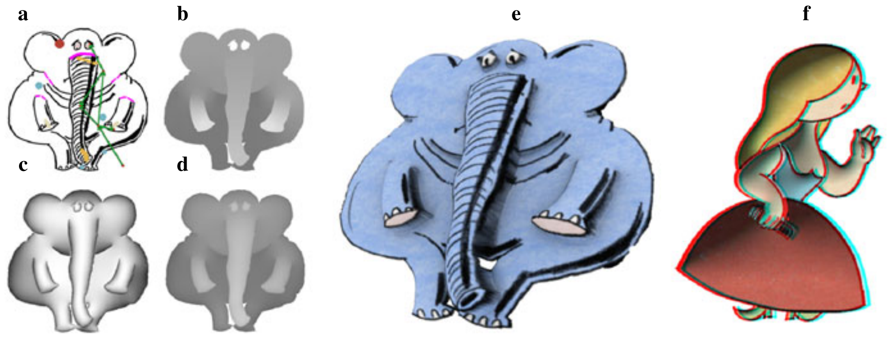


Fig. 14.24 Approximate 3D model and stereo: (**a**) original image with LazyBrush scribbles and depth inequalities, (**b**) depth map, (**c**) Lumo shading, (**d**) depth map after shape-from-shading applied on the Lumo shading, (**e**) approximate 3D mode with texture, (**f**) anaglyph stereo. Reproduced with kind permission from Blackwell Publishing Ltd. © Anifilm + © UPP & DMP + © EG & Blackwell. Used with permission

Solving this inhomogeneous system with Dirichlet boundary conditions and two different right hand sides yields texture coordinates for the surface S projected on the plane, see Fig. 14.21(f) and Fig. 14.23(d).

14.5.2.4 Approximate 3D Model and Stereo

We can further combine the depth map with Lumo shading and apply shape-from-shading [7] to reconstruct an approximation of the 3D surface, see Fig. 14.24(a–e). Such a simple 3D model can be further refined in some modelling tool or rendered from two different viewpoints to obtain stereoscopic images, Fig. 14.24(f).

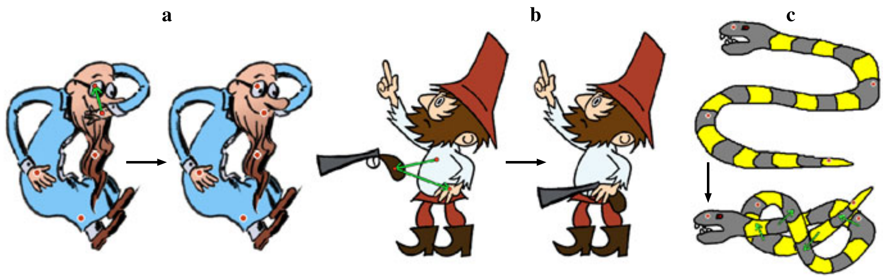


Fig. 14.25 Local layering—depth inequalities can be used to obtain desired visibility during the interactive shape manipulation (**a**) and fragment composition (**b**), the approach can handle complex self-occlusions (**c**). Reproduced with kind permission from Blackwell Publishing Ltd. (**a**). © Anifilm + © UPP & DMP + © EG & Blackwell. Used with permission

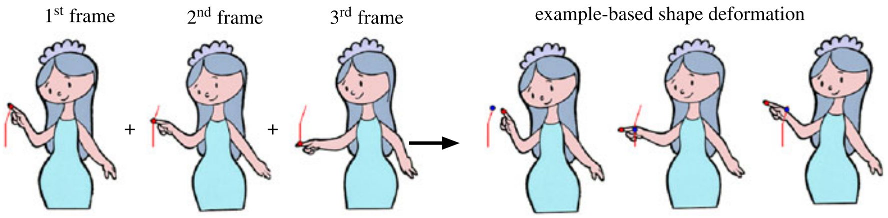


Fig. 14.26 Example-based shape manipulation—by registering several consecutive frames (*left*) a smooth sequence of intermediate frames can be generated. This can be utilized for a synthesis of new poses satisfying a user-given positional constraint (*right*): the current position of the dragged point (*red dot*) is projected (*blue dot*) on its original motion trajectory (*red curve*) to retrieve the corresponding intermediate frame, which is subsequently deformed to match the current position of the dragged control point. Reproduced with kind permission from ACM. © UPP & DMP + © ACM. Used with permission

14.5.3 Shape Manipulation and Example-Based Synthesis

Depth maps generated by the algorithm described in Sect. 14.3 can be used to resolve the visibility of occluded parts during interactive shape manipulation, see Fig. 14.11. The user can freely interact with the shape and modify the visibility on the fly using additional depth inequalities, Fig. 14.25(a). A similar problem arises in systems where the user extracts and composes fragments of images [31]. Here depth inequalities allow quick reordering of regions to obtain correct composition, see Fig. 14.25(b). This operation is also known as *local layering* for which alternative techniques exist [23], however, the approach presented in this chapter is more general as it handles complex self-occlusions, Fig. 14.25(c).

The knowledge of correspondences between consecutive animation frames allows the creation of smooth intermediate transitions, see Fig. 14.26. For this task sub-pixel accurate registration is required. We can use the results of the as-rigid-as-possible image registration, Sect. 14.4.2, as an initial guess for a more precise algorithm with a flexible deformation model, e.g., [9]. Then, intermediate frames can be obtained by interpolating positions of vertices on the control lattice and performing several shape regularization iterations to enforce rigidity. Inside each square, sub-pixel accurate source-target and target-source deformation fields together with pixel blending help to produce the final smooth transition.

The process of inbetweening can be additionally controlled by the user. This extension can be viewed as an example-based shape manipulation which respects the original animation but is more flexible than simple inbetweening. In this scenario, the user can drag a specific vertex on the control lattice and move it to a different location. By projecting this new location on its inbetweening trajectory we can generate the closest transition frame and deform it to match the user-specified constraint, see Fig. 14.26, for details refer to [32].

Acknowledgements This work has been supported by the Marie Curie action ERG, No. PERG07-GA-2010-268216 and partially by the Technology Agency of the Czech Republic under the project TE01010415 (V3C—Visual Computing Competence Center). Hand-drawn images used in this chapter are courtesy of UPP & DMP, Anifilm, Lukáš Vlček, and Ondřej Sýkora.

References

1. Alexa, M., Cohen-Or, D., Levin, D.: As-rigid-as-possible shape interpolation. In: ACM SIGGRAPH Conference Proceedings, pp. 157–164 (2000)
2. Belongie, S., Malik, J., Puzicha, J.: Shape matching and object recognition using shape contexts. IEEE Trans. Pattern Anal. Mach. Intell. **24**(24), 509–522 (2002)
3. Boykov, Y., Funka-Lea, G.: Graph cuts and efficient N-D image segmentation. Int. J. Comput. Vis. **70**(2), 109–131 (2006)
4. Boykov, Y., Kolmogorov, V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. IEEE Trans. Pattern Anal. Mach. Intell. **26**(9), 1124–1137 (2004)
5. Boykov, Y., Veksler, O., Zabih, R.: Markov random fields with efficient approximations. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 648–655 (1998)
6. Dahlhaus, E., Johnson, D.S., Papadimitriou, C.H., Seymour, P.D., Yannakakis, M.: The complexity of multiway cuts. In: Proceedings of ACM Symposium on Theory of Computing, pp. 241–251 (1992)
7. Ecker, A., Jepson, A.D.: Polynomial shape from shading. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 145–152 (2010)
8. Felzenszwalb, P.F., Huttenlocher, D.P.: Distance transforms of sampled functions. Tech. Rep. TR2004-1963, Cornell University (2004)
9. Glocker, B., Komodakis, N., Tziritas, G., Navab, N., Paragios, N.: Dense image registration through MRFs and efficient linear programming. Med. Image Anal. **12**(6), 731–741 (2008)
10. Grady, L.: Random walks for image segmentation. IEEE Trans. Pattern Anal. Mach. Intell. **28**(11), 1768–1783 (2006)
11. Igarashi, T., Moscovich, T., Hughes, J.F.: As-rigid-as-possible shape manipulation. ACM Trans. Graph. **24**(3), 1134–1141 (2005)
12. Jamriška, O., Sýkora, D., Hornung, A.: Cache-efficient graph cuts on structured grids. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 3673–3680 (2012)
13. Jeschke, S., Cline, D., Wonka, P.: A GPU Laplacian solver for diffusion curves and Poisson image editing. ACM Trans. Graph. **28**(5), 116 (2009)

14. Johnston, S.F.: Lumo: illumination for cel animation. In: Proceedings of International Symposium on Non-photorealistic Animation and Rendering, pp. 45–52 (2002)
15. Kahn, A.B.: Topological sorting of large networks. Commun. ACM **5**(11), 558–562 (1962)
16. Kaneko, T., Takahei, T., Inami, M., Kawakami, N., Yanagida, Y., Maeda, T., Tachi, S.: Detailed shape representation with parallax mapping. In: Proceedings of International Conference on Artificial Reality and Telexistence, pp. 205–208 (2001)
17. Koenderink, J.J.: Pictorial relief. Philos. Trans. R. Soc. Lond. **356**(1740), 1071–1086 (1998)
18. Koenderink, J.J., van Doorn, A.J., Kappers, A.M.L.: Pictorial surface attitude and local depth comparisons. Percept. Psychophys. **58**(2), 163–173 (1996)
19. Langer, M.S., Buelthoff, H.H.: Depth discrimination from shading under diffuse lighting. Perception **29**(6), 649–660 (2000)
20. Levin, A., Lischinski, D., Weiss, Y.: Colorization using optimization. ACM Trans. Graph. **23**(3), 689–694 (2004)
21. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vis. **60**(2), 91–110 (2004)
22. Luft, T., Colditz, C., Deussen, O.: Image enhancement by unsharp masking the depth buffer. ACM Trans. Graph. **25**(3), 1206–1213 (2006)
23. McCann, J., Pollard, N.S.: Local layering. ACM Trans. Graph. **28**(3), 84 (2009)
24. Orzan, A., Bousseau, A., Winnemöller, H., Barla, P., Thollot, J., Salesin, D.: Diffusion curves: a vector representation for smooth-shaded images. ACM Trans. Graph. **27**(3), 92 (2008)
25. Pao, H.K., Geiger, D., Rubin, N.: Measuring convexity for figure/ground separation. In: Proceedings of IEEE International Conference on Computer Vision, pp. 948–955 (1999)
26. Potts, R.: Some generalized order-disorder transformation. In: Proceedings of Cambridge Philosophical Society, vol. 48, pp. 106–109 (1952)
27. Qu, Y., Wong, T.T., Heng, P.A.: Manga colorization. ACM Trans. Graph. **25**(3), 1214–1220 (2006)
28. Schaefer, S., McPhail, T., Warren, J.: Image deformation using moving least squares. ACM Trans. Graph. **25**(3), 533–540 (2006)
29. Shekhovtsov, A., Kovtun, I., Hlaváč, V.: Efficient MRF deformation model for non-rigid image matching. Comput. Vis. Image Underst. **112**(1), 91–99 (2008)
30. Sýkora, D., Buriánek, J., Žára, J.: Colorization of black-and-white cartoons. Image Vis. Comput. **23**(9), 767–782 (2005)
31. Sýkora, D., Buriánek, J., Žára, J.: Sketching cartoons by example. In: Proceedings of Eurographics Workshop on Sketch-Based Interfaces and Modeling, pp. 27–34 (2005)
32. Sýkora, D., Dingliana, J., Collins, S.: As-rigid-as-possible image registration for hand-drawn cartoon animations. In: Proceedings of International Symposium on Non-photorealistic Animation and Rendering, pp. 25–33 (2009)
33. Sýkora, D., Dingliana, J., Collins, S.: LazyBrush: flexible painting tool for hand-drawn cartoons. Comput. Graph. Forum **28**(2), 599–608 (2009)
34. Sýkora, D., Sedlacek, D., Jinchao, S., Dingliana, J., Collins, S.: Adding depth to cartoons using sparse depth (in)equalities. Comput. Graph. Forum **29**(2), 615–623 (2010)
35. Sýkora, D., Ben-Chen, M., Čadík, M., Whited, B., Simmons, M.: TexToons: practical texture mapping for hand-drawn cartoon animations. In: Proceedings of International Symposium on Non-photorealistic Animation and Rendering, pp. 75–83 (2011)
36. Walther, D., Koch, C.: Modeling attention to salient proto-objects. Neural Netw. **19**(9), 1395–1407 (2006)
37. Wang, Y., Xu, K., Xiong, Y., Cheng, Z.Q.: 2D shape deformation based on rigid square matching. Comput. Animat. Virtual Worlds **19**(3–4), 411–420 (2008)
38. Winnemöller, H., Orzan, A., Boissieux, L., Thollot, J.: Texture design and draping in 2D images. Comput. Graph. Forum **28**(4), 1091–1099 (2009)
39. Yarbus, A.L.: Eye Movements and Vision. Plenum, New York (1967)