The Visual Computer

An approach to computer-supported cartooning

Joaquim S. Madeira[1,*], André Stork[2], Markus H. Groß[3]

Visual Computing Group, Computer Graphics Center, Wilhelminenstraße 7, D-64283 Darmstadt, Germany

Abstract. An approach to computer-supported cartooning is described that aims at optimizing the image-related working process by introducing computer-support in the drawing and painting stages of traditional cartoon production, in particular, allowing automatic coloring of a sequence of digitized images. This is achieved by using a shape-matching algorithm to evaluate the similarity of image regions and by performing an optimum region assignment to identify the corresponding ones and propagate the color information through the image sequence. In order to maintain the use of traditional drawing tools, the first step in the proposed system architecture is the processing of scanned animators' drawings to enhance their quality and extract meaningful information. Two different system modules allow either manual coloring of images or computer-assisted automatic painting of an image sequence. Additional modules allow the construction of a vector representation for the images, the generation of in-betweens, and the composing of each cartoon frame. The first stages of the system's architecture – image preprocessing, painting and vectorization – are presented. Special emphasis is placed on the fundamental ideas behind the computer-assisted painting and vectorization steps. The competitiveness of the approach, which requires no special hardware or high-performance workstation, is shown.

1 Introduction

Since the beginning of the 1970s various techniques have been developed to support the production of classical cartoons. Some of them attempted to automate the generation of drawings using key-frames and interpolation methods (Baecker 1969; Burtnyk and Wein 1971). Others tried to support the entire production process using vector input (Levoy 1977) or raster graphics (Stern 1979). To overcome some of the problems of the earlier in-betweening approaches, e.g., the shrinkage associated with rotation, additional and more complex methods for visually pleasing key-frame animation were developed (e.g., Burtnyk and Wein 1976; Reeves 1981; Kochanek and Bartels 1984; Sederberg and Greenwood 1992; Sederberg et al. 1993). Clearly, none of them can solve the basic problem of the lack of information inherent in the images.

Catmull (1978) carried out a precise analysis of the problems of computer-supported animation, while Thomas and Johnston (1981) stressed the artistic side of animation; as a result, the limitations of fully automatic approaches to cartooning became quite clear. In the 1980s research and development concentrated on 3D animation tools, some of them taking the principles of classical animation (Lasseter 1987) into account. Although the 3D tools revolutionized certain market segments – e.g., special effects, TV commercials – some attempts to use them for traditional cartooning were less successful commercially, either because of their complexity, lack of user-friendliness or simply as a result of their costs and requirements, which sometimes made

Key words: Computer-supported cartooning – Shape-matching – Coloring – Painting – Vector representation – Image structuring

Permanent addresses:
[1] Mathematics Department, University of Coimbra Coimbra, Portugal
[2] Fraunhofer Institute for Computer Graphics Darmstadt, Germany
[3] Institute for Information Systems, Computer Science Department, Swiss Federal Institute of Technology, Zurich, Switzerland
*E-mail: jmadeira@igd.fhg.de
Correspondence to: J. Madeira

them uncompetitive regarding manual production. For this reason, the expectations raised by early researchers have not yet been completely fulfilled.

Therefore, dedicated tools are needed that do not limit the animation artists in their creativity; they need to be easy to use and allow cheaper cartoon production. Based on these requirements, a prototype was developed that allows automatic painting of a sequence of images by combining methods from computer graphics, image processing and scene analysis, and integrating them into the usual working procedure, thus optimizing it. Our approach acts as a bridge between traditional and computer-supported production, making it easily acceptable to animation artists, and strikes a balance between the amount of user interaction and the execution time of the associated algorithms.

Before presenting the overall architecture of the system, a short review of the traditional cartoon production process is given in the next section. Afterwards, the image preprocessing, manual painting and computer-assisted painting stages of the prototype are presented. For the latter the basic algorithms used are thoroughly described. The fundamental ideas associated with the image vectorization step, which is essential for the later generation of in-betweens, are also discussed.

2 Traditional cartooning

The production process of a cartoon consists of several major steps (Catmull 1978). The ones that are of direct interest in the context of this paper are briefly described below.

The fundamental cartoon production information is defined by:

- The story-board, in which the story is split into scenes and sequences
- The model sheets, which show all cartoon characters in various poses
- The exposure sheet, which contains a precise formal description on how to build up each frame in a sequence by stacking different images (i.e., layers) in the defined order

For each cartoon sequence the animators draw first the key layers that contain (at least) one cartoon character – or a part of it – in characteristic or extreme positions. Between each pair of key layers, in-betweeners of different skill levels draw the intermediate layers that make up the sequence. These outline drawings are then transferred to transparencies and colored. Afterwards the different layers are combined into frames according to the exposure sheet. Finally, the film is shot frame by frame.

Considering the thousands of layers that have to be drawn and painted for just 1 min of animated film, these two stages make up approx. 60% of the labor required in traditional cartooning (Durand 1991) and are well suited for the introduction of computer-support. Time-consuming painting can especially be accelerated with appropriate methods.

3 Overview of the system's architecture

Since the animation artists' creative freedom must not be limited in any way and since there is no input device that satisfies all of the artists' needs, we decided to have hand-drawn images as input, thus enabling the artists to continue to use their convenient tools: specific pencils and paper.

Aiming at supporting the drawing and painting stages of the traditional cartoon production, the system consisting of the modules shown in Fig. 1 was laid out. The information flow between the different modules, which comprise almost all of the image-related tasks in cartoon production, is also depicted.

The specific purpose of each module is roughly described below:

- *Preprocessing* – the scanned line drawings are cleaned, skeletonized and segmented. Features like the regions' contours, and neighbors are extracted for later use.
- *Painting* – a digital image is painted interactively by assigning colors to its regions. Simultaneously, the image is hierarchically structured into objects consisting of sets of regions. It is intended that only the layers of the first frame of each sequence be colored manually; for the remaining ones the computer-assisted painting module is used.

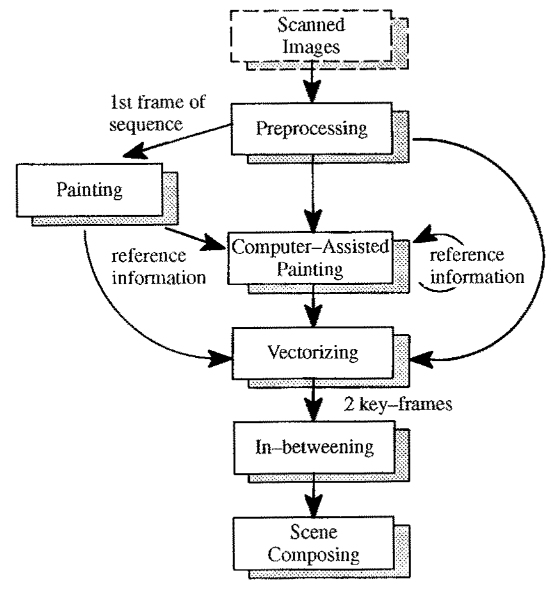


Fig. 1. The architecture of the global system

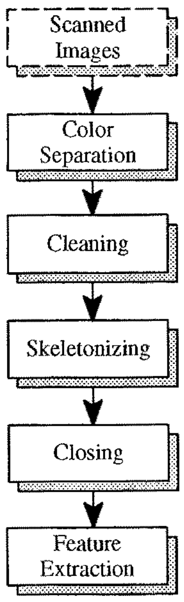


Fig. 2. The preprocessing pipeline

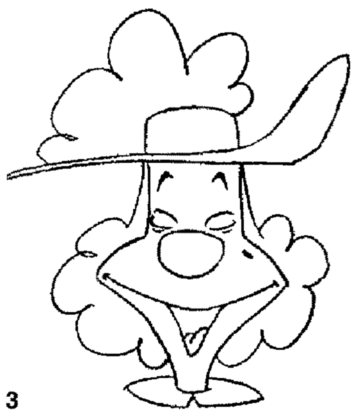
- *Computer-assisted painting* – given an already painted image, its successor in the image sequence will be (semi-)automatically painted, using shape-matching and region-assignment algorithms to identify corresponding regions and propagate the color information. Then, the just-colored image becomes the reference one, and the process continues.
- *Vectorizing* – the raster images are vectorized and represented in a topologically correct way. The thickness of the image lines plays an important role in cartoon images and is associated with the vector representation.
- *Interactive in-betweening* – between two vectorized key layers additional intermediate layers are (semi-)automatically generated and, if needed, interactively modified. This further reduces the number of images that have to be manually drawn.
- *Scene composing* – using a computerized exposure sheet and sequences of painted layers, each cartoon frame is composed by layer stacking. The entire frame sequence can then be animated for testing purposes with a near real-time display rate.

4 Preprocessing

Using a scanner to digitize the line drawings entails the need for an image preprocessing stage, which is an additional one in comparison to the traditional procedure. Aiming at off-line processing, only fast algorithms that can run independently from any user intervention were selected. A pipeline grouping them together was defined as shown in Fig. 2.

Since the scanned images usually contain black and blue lines – the latter will be painted with the fill color of the region they enclose – the preprocessing starts with color separation and generation of two binary images. This is done by weighted color-component averaging and thresholding. To remove noise and enhance image quality the binary images are then cleaned. Because of the line-shaped nature of the image contents, mathematical morphology operations (Haralick et al. 1987) turned out to be inappropriate for such a task. Instead, a spike/blob removal procedure – namely, connected component suppression (Haralick and Shapiro 1992) – is applied to the images.

Figure 3 shows a binarized cartoon image, scanned with 150 dpi and containing approx. 250,000 pixels. Clearly, when handling the large


3

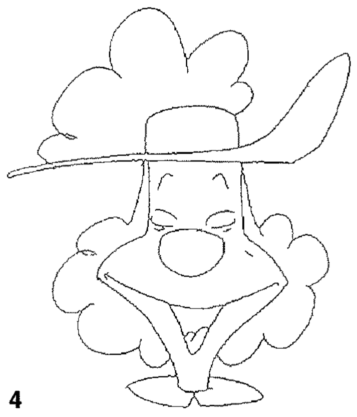

4

Fig. 3. A scanned image (original image by courtesy of France Animation)

Fig. 4. The image skeleton

number of digital images needed for a cartoon, a compromise has to be found between the resolution necessary for appropriate image quality and storage space. For an entire cartoon this can only be achieved by using image-compression methods.

To enable the extraction of image features the skeleton of the image lines is computed. A distance-based skeletonizing algorithm (Niblack et al. 1992) was selected since its execution time does not depend on the largest-occurring line thickness. The distance map is calculated using a 3–4 metric because of its relatively small approximation error ($<10\%$) to the Euclidean distance (Leymarie and Levine 1992), which allows accurate reconstruction of the line thicknesses. Since the selected skeletonization algorithm does not guarantee construction of a minimal skeleton – for which removing just one more pixel would yield a loss of connectedness – a post-thinning step is also applied to the image. This two-phase process generates the result presented in Fig. 4.

Afterwards a gap-closing step can be applied, which uses the Gestalt laws (Wertheimer 1921) as a basis for a set of heuristics and attributes to each possible closing a measure of goodness based on the distance between endpoints and the line direction in the vicinity of each endpoint.

The skeleton allows the extraction of features like adjacency relationships between regions, as well as the recognition of endpoints and simplifies the later vectorization process. Clearly, before adjacency relationships between regions can be extracted the image regions have to be identified. This segmentation is done by flood-filling (Heckbert 1990) each image region with an individual label.

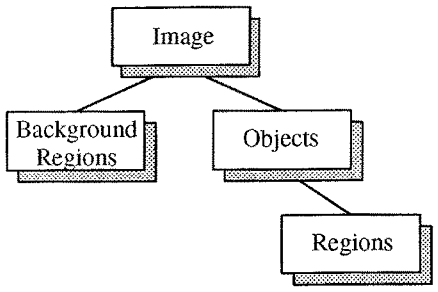


Fig. 5. The hierarchical image structure

The example image shown in Fig. 3, containing 532×516 pixels, passed the entire preprocessing pipeline in approx. 27 s on a SUN SPARCstation 10, Model 30, with 86.1 MIPS, 10.6 MFLOPS and 32 MB RAM; the code is implemented in C++.

5 Painting

The painting module can be used to color all images like a conventional paint tool, but it is intended to be used only for the layers that will be a reference for their successors in the computer-assisted painting stage. The user paints the different image regions by assigning a color to each of them. Since the image is already segmented, the painting is reduced to just a manipulation of the look-up table; this speeds up the interactive process.

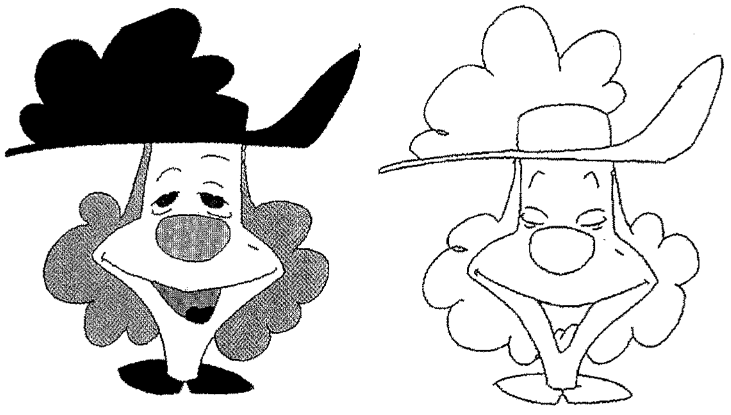


Fig. 6. Example situation before the start of the assisted painting (original images by courtesy of France Animation)

While painting the image, its contents are structured hierarchically into objects consisting of sets of regions. An object can be arbitrarily defined by the user, but the intention is that it represents a cartoon character or a part of it. The structuring is done with almost no additional workload on the part of the user and supports efficiently the assisted painting. Figure 5 illustrates the structure that is built up as a basis for the assisted painting.

Furthermore, the painting module places a comfortable gap-closing facility at the user's disposal. If the user notices a color spreading, he can select two line endpoints and the system automatically draws a line between them, checks if the closing was successful and updates the underlying data structure.

6 Computer-assisted painting

All the processing steps presented so far aim at building up a basis for the computer-assisted painting process. Its goal is to propagate the color information from an already-painted image to its uncolored successor in the image sequence. An initial situation, before the start of the assisted painting process, is depicted in Fig. 6.

Although the images composing a cartoon sequence are quite close to each other – there are only 1/25 s between two succeeding images – region deformations and partial or global occlusions usually occur. Thus, a (semi-)automatic painting method must be robust against these effects, as well as against region translation, rotation and scaling. Given these requirements, simple position-bounded approaches often present in today's commercial tools are inadequate.

Our approach can be described as follows:

A topologically guided shape matching is carried out to evaluate the similarity of regions, considering specific constraints and heuristics, and region correspondences are established using linear optimization to determine a local best region assignment.

When shape matching is applied, one fundamental task is to find a description for a region's shape that incorporates all of the previously mentioned requirements. One promising approach is to use the curvature along a region's contour. The curvature itself is robust regarding translation, rotation and uniform scaling, but in the case of raster images a precise determination of curvature is not possible; only approximations can be used for generating a shape description. A further task is the appropriate construction of such shape descriptions to enable efficient shape comparisons.

The region-matching and assignment process starts by letting the user select one region – region R – in the reference (i.e., previously painted) image and its corresponding region – region P – in the image to be painted. This initial match was introduced to speed up the matching procedure by avoiding an automatic search for two regions that constitute a best first match.

Then a set of matching candidate regions is determined for each image. For the reference image the object structure is taken into account: only neighboring regions – R_i, $i = 1, 2, \dots, m$ – of R that belong to the same object and do not yet have a corresponding region in the image to be painted are considered to be possible matching candidates. For the image to be painted all neighboring regions – P_j, $j = 1, 2, \dots, n$ – of P that do not yet have a corresponding region in the reference image are possible candidates.

Afterwards, for every pair of candidate regions (R_i, P_j), a measure of the dissimilarity between R_i and P_j is computed, using curvature-based shape descriptions. This is a critical step in terms of execution time; thus, we decided to use a set of heuristics to restrict a priori the number of possible candidate region pairs and minimize the number of dissimilarity values that have actually to be computed.

Using linear optimization, an assignment of candidate regions is then determined that minimizes the sum of dissimilarity measures. The color information is propagated according to the region assignments and the corresponding object structure is replicated in the image to be painted. Using each pair of assigned regions the process runs recursively until no assignable regions remain.

Summing up, the two steps that have to be accomplished for each image region prior to the matching and assignment process are:

- Compute a shape description for each image region.
- Code each description as a character string to allow shape comparisons.

The steps carried out for region matching and assignment are:

- Determine the set of pairs of candidate regions and filter it with a set of heuristics to limit the number of comparisons.
- Compare the string descriptions associated with each surviving pair of candidate regions to determine the dissimilarity measures.
- Perform the region assignment, given the dissimilarity values associated with the pairs of candidate regions.

We now explain each of these steps. The algorithms for the computation of a shape description for each region, string-based shape coding and string comparison are based on Liu and Srinath (1992).

6.1 Shape description

The first step towards describing a region's shape is taken by applying the Sobel operator (Gonzalez and Wintz 1987) to the skeletonized and segmented image to estimate the gradient at each one of the region's contour pixels. The gradient of an image $f(x, y)$ at pixel location (x, y) is defined as the two-dimensional vector $\nabla f(x, y) = (\partial f/\partial x, \partial f/\partial y)$. The Sobel operator is usually applied to gray-level images and estimates the gradient by inspecting the eight neighbors of (x, y), taking into account their gray values.

Let $P_{x,y}$ represent the bit value at location (x, y). Using the Sobel operator, the gradient component in the x direction is approximated as:

$$\Delta x = (P_{x+1,y-1} + 2P_{x+1,y} + P_{x+1,y+1}) - (P_{x-1,y-1} + 2P_{x-1,y} + P_{x-1,y+1}) \qquad (1)$$

and in the y direction as

$$\Delta y = (P_{x-1,y+1} + 2P_{x,y+1} + P_{x+1,y+1}) - (P_{x-1,y-1} + 2P_{x,y-1} + P_{x+1,y-1}). \qquad (2)$$

The gradient direction is then defined as

$$\Theta = \tan^{-1}\left(\frac{\Delta y}{\Delta x}\right). \qquad (3)$$

To exemplify how the Sobel operator works, an example is given in Fig. 7; the gradient direction is determined along the region's contour, represented by the pixels in darker gray. Figure 8 depicts the gradient direction as a function.

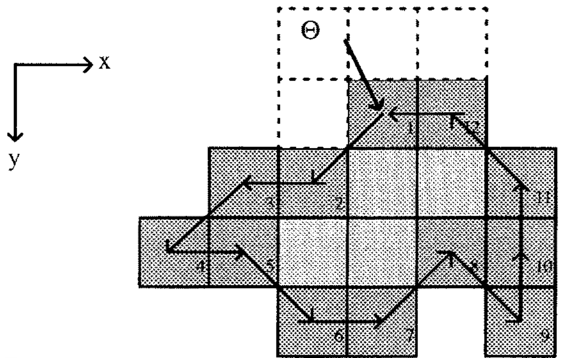
Once the gradient direction has been estimated, a measure of the curvature variation c_i at each contour pixel P_i, $i = 1, 2, \ldots, m$, can be obtained by simply computing the difference between two succeeding gradient direction values:

$$c_i = \Theta_{(i \bmod m)+1} - \Theta_i \qquad (4)$$

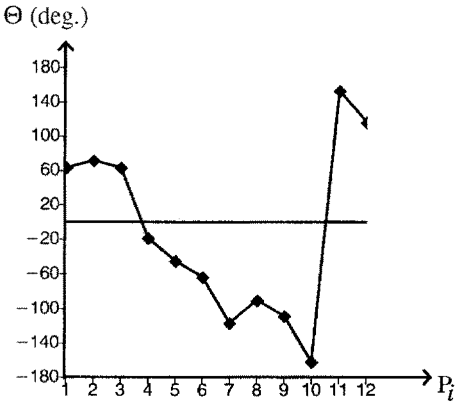
The resulting contour description function for the example region is shown in Fig. 9.

6.2 String-based shape description

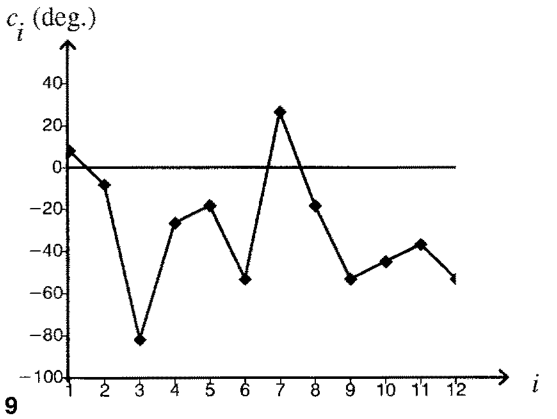
So far, the contour descriptions are not sufficiently invariant against affine transformations, e.g., their length is directly related to the length of the contour. Each contour description is at this stage coded into a string aiming at increasing its



7



8



9

Fig. 7. Applying the Sobel operator

Fig. 8. The gradient direction at each contour pixel P_i

Fig. 9. The contour description function

scaling invariance and compactness, as well as accelerating the later comparison of any two shape descriptions.

According to Liu and Srinath (1992) a string can be built from a shape description by introducing quantization levels and using the sign of each value to determine the character inserted in the string: negative values are coded into one or more '*d*' (down) and positive ones into one or more '*u*' (up). The number of times each character is inserted in the string is computed by taking each function value and performing integer division by the amplitude of the quantization level. Figure 10 illustrates this; the character(s) successively inserted in the string are placed near to their function points; the resulting string is: *dddddd*.

This example shows the compression effect of the string coding. The contour description containing 12 curvature measures is coded into 7 characters. Clearly, the coding does not work without quantization errors, e.g., the positive curvature values are not represented in the coding since they are smaller than the quantization level.

Moreover, this simple kind of string coding has the essential drawback that all convex regions generate a string consisting exclusively of '*d*' characters; thus, they are not distinguishable. Note that if a character is inserted multiple times at any given step, it is not possible to reconstruct the contour shape from the string since the same substring can correspond to different contour pieces.

To overcome this problem we decided to use more than two characters for string coding: when building the string, instead of using only two characters and inserting them multiple times, different characters are used to express the function's absolute value and sign. Assuming that each quantization level in the positive direction is coded with characters from the beginning of the alphabet starting with '*a*,' and the negative ones are coded starting backwards from '*z*,' the resulting string is: *yzzzzz*. This coding has the advantage of expressing the rate of change in curvature direction by different characters, instead of just inserting one character multiple times to accomplish this.

String coding has one crucial parameter that has to be properly chosen: the size of the quantization level. This is illustrated in Fig. 11, where the above curvature function is string coded using smaller quantization levels. Now the resulting string is the following: *vzzxazxyyx*.

Note that one of the positive values is now represented in the string, which means that the concavity at the bottom of the region depicted in Fig. 7 is detected. Thus, smaller quantization levels correspond to a finer contour description, but are also more sensitive against noise. Since the strings are constructed to facilitate region matching, a balance has to be found between description accuracy and string length, otherwise the execution time of the string comparison algorithm will be prohibitively high. In the current implementation the function range is divided into ten 36° quantization levels.

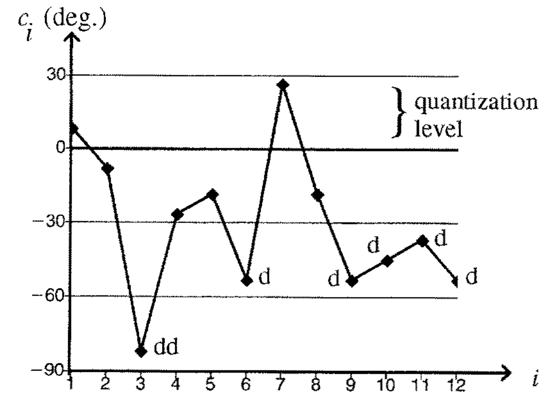
6.3 Comparison heuristics

Since the shape of a region's contour is represented by a string, the dissimilarity of two regions can be measured by comparing their respective shape coding strings. As mentioned before, the string comparison is a crucial time step of the assisted painting process. A way to avoid performing unnecessary comparisons is to use a set of heuristics to sort out which pairs of candidate regions need to be compared and which can be outright rejected. However, one should not forget that the use of heuristics decreases somewhat the invariance regarding image transformations.

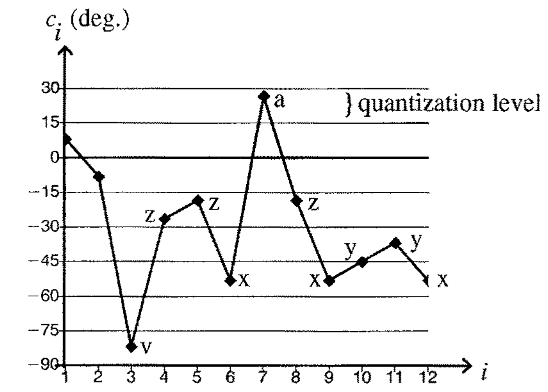
Given two regions to be compared and their respective shape-coding strings, one filtering step is to check if one of the strings is judged to be much longer than the other; in this case no string comparison is performed and a large default dissimilarity value is assigned to this pair of regions. This heuristic restricts the invariance regarding image scaling, but large, sudden scale changes do not often occur in most cartoon sequences, so the restriction is not too imposing.

Another step is to verify if there is too large a difference in the relative position of the centroids of the regions to be compared – say R_i and P_j – regarding the centroids of their previously matched neighboring regions R and P, in the reference image and in the image to be painted, respectively. The allowed changes in relative position depend on the distances between the centroids: the greater the distances, the smaller the allowed angular variations. If the relative positions of the two matching candidates are judged to be too different, again a default dissimilarity value is assigned and no string comparison is performed. This heuristic restricts the invariance regarding image rotation and translation, but again it is not too imposing.

A further purely topological heuristic is also used: according to their neighborhood information, regions are classified as *containers* (i.e. they contain one or more regions), as *contained-in* (i.e.,



10



11

Fig. 10. Coding the function into a string

Fig. 11. Division of the function range using smaller quantization levels

surrounded by a single region) or as none of both. Regions that are *contained-in* in the reference image can only be assigned to regions of the same type. Again, some comparisons are avoided and a default dissimilarity value is assigned. However, in some cases this heuristic prevents correct matches: for instance, when a region is moving from the interior of its container towards its border and starts crossing it, it ceases to be a *contained-in* region and can no longer be assigned to its corresponding region, which still is a *contained-in* region in the reference image.

6.4 Evaluating region dissimilarity

For each pair of regions that still have to be compared it is necessary to compute a measure of their shape dissimilarity. This is accomplished using the respective shape-coding strings and computing the cost of transforming one of the strings into the other. The conversion costs between the two strings give a measure of the dissimilarity of the regions: low costs correspond to high similarity and vice versa.

The aim is then to determine the least costly sequence of character substitutions, insertions, and deletions that convert one of the strings into the other. Usually the costs of the different operations are written into a matrix and a minimum cost path through the matrix defines the best sequence of substitutions, insertions or deletions needed to transform the origin string into the destination string. If the matrix rows are associated with the origin string and the columns with the destination string, horizontal moves in any matrix path correspond to the insertion of characters into the destination string, vertical moves to the deletion of characters from the origin string and diagonal moves to character substitution. The costs associated with each possible move have to be defined as a function of the characters and the probability of their substitution, insertion, or deletion taking place.

The string comparison algorithm used (Wagner and Fischer 1974; Liu and Srinath 1992) is as follows:

1. Given an origin string $A = A_1A_2 \ldots A_m$ and a destination string $B = B_1B_2 \ldots B_n$, a cost matrix d of size $(m+1)\times(n+1)$ is created.
2. The first element of the matrix, which corresponds to the cost of transforming a null string into itself, is zero – i.e. $d_{00} = 0$.
3. The remaining elements of the first row are computed as:

$$d_{0j} = \sum_{r=1}^{j} \gamma(\lambda \rightarrow B_r), j = 1,2,\ldots,n \tag{5}$$

   where $\gamma(\lambda \rightarrow B_r)$ represents the cost of inserting character B_r in the destination string. Each element d_{0j} represents the cost of transforming the null string into successively larger substrings of B.
4. The remaining elements of the first column are computed as:

$$d_{i0} = \sum_{r=1}^{i} \gamma(A_r \rightarrow \lambda),\ i = 1,2,\ldots,m \tag{6}$$

   where $\gamma(A_r \rightarrow \lambda)$ represents the cost of deleting character A_r from the original string. Each element d_{i0} represents the cost of deleting successively larger substrings of A.
5. The remaining matrix elements are computed as:

$$\begin{aligned} d_{ij} = \min(&d_{i-1\,j-1} + \gamma(A_i \rightarrow B_j),\\ &d_{i-1\,j} + \gamma(A_i \rightarrow \lambda), d_{i\,j-1} + \gamma(\lambda \rightarrow B_j)) \end{aligned} \tag{7}$$

   for all $i = 1,2,\ldots,m$ and $j = 1,2,\ldots,n$, where $\gamma(A_i \rightarrow B_j)$ represents the cost of replacing character A_i in the origin string by character B_j in the destination string. Each element d_{ij} represents the cost of transforming the substring $A_1A_2 \ldots A_i$ into the substring $B_1B_2 \ldots B_j$ – the last operation being either a character substitution, deletion or insertion.
6. The total cost of transforming the origin string into the destination string is given by d_{mn}.

When comparing a pair of regions without any knowledge about eventual region transformations, it is possible that the starting points of both string-coded shape descriptions do not match in the sense that their relative position in each region's contour is not the same. Then, the only way of correctly comparing both strings is to successively shift one of them, one character at a time, perform successive string comparisons (i.e., compute their conversion costs), and keep the minimum value. In the end the minimum value is the rotation invariant measure of the strings' dissimilarity (Liu and Srinath 1992). Since such a sequence of conversion costs neither increases nor decreases monotonically – instead, the values are usually oscillating –, it cannot be decided, at any shifting position, if the minimum value has already been found or if one is moving downhill towards the minimum conversion cost. However, an analysis of the sequences of conversion costs associated with a set of test images showed that the consecutive values do not vary in a wide range and are rather close to the minimum.

One optimization regarding execution efficiency – given that the conversion cost at any string shifting position is relatively close to the minimum one – is not to perform string shifting. Although the conversion costs obtained are less precise, the previous filtering of the possible matching pairs through the set of heuristics compensates for the lack of precision introduced in the string matching.

The shape-dissimilarity values assigned to the pairs of matching candidate regions through the heuristics – a default value stating that no final assignment is allowed – and the string comparison algorithm are used as input to the region assignment algorithm. This algorithm establishes the region correspondences between the sets of matching candidates by minimizing the sum of the dissimilarity values associated with the assigned regions.

6.5 Region assignment

The aim of the region assignment can be stated as follows: given two sets of regions $\{R_i\}$ – belonging to the reference image – and $\{P_j\}$ – belonging to the image to be painted – with m and n elements, respectively, and given the corresponding $(m \times n)$ dissimilarity values, determine the $\min\{m,n\}$ assignment pairs in such a way that no assignment sequence exists whose sum of dissimilarity values is smaller.

Several methods exist to solve this assignment problem. A simple successive assignment using *row search*, *column search* or *matrix search* was rejected since the results vary depending on the assignment order, and an optimum assignment is usually not found. The best method is to formulate the assignment as a linear programming problem:

Let the dissimilarity between any two regions R_i and P_j be denoted by d_{ij}, and let x_{ij} have the value 1 if region P_j is assigned to region R_i, and be 0 otherwise. To determine the assignments in such a way as to minimize the sum of the dissimilarities associated with the assigned regions, the problem is formulated as follows (Gass 1975):

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} d_{ij} x_{ij} \tag{8}$$

subject to

$$\sum_{j=1}^{n} x_{ij} = 1,\ i = 1, 2, \ldots, m \tag{9}$$

$$\sum_{i=1}^{m} x_{ij} = 1,\ i = 1, 2, \ldots, n \tag{10}$$

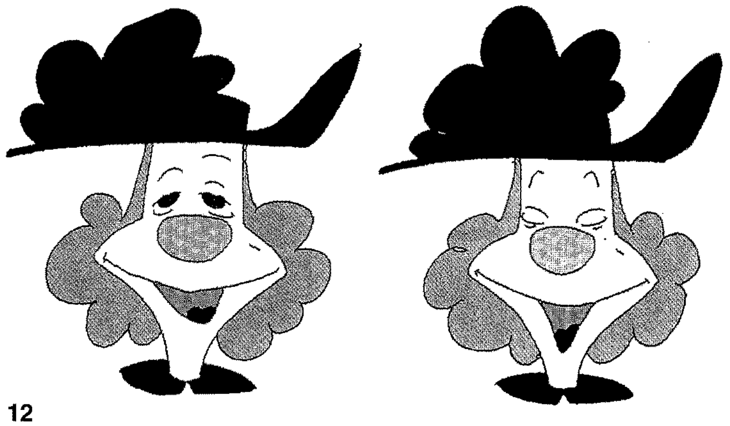
$$x_{ij} \in \{0, 1\} \tag{11}$$

The assignment problem is a special case of the well-known transportation problem and can be solved using the appropriate version of the simplex method (Gass 1975). Since the basic solutions to the assignment problem are highly degenerate, a slightly perturbed problem has to be set up to facilitate the computational procedure; from its solution the corresponding solution of the original problem is inferred. When $m \neq n$, fictitious regions and default dissimilarity values are introduced to transform the problem into a square one.

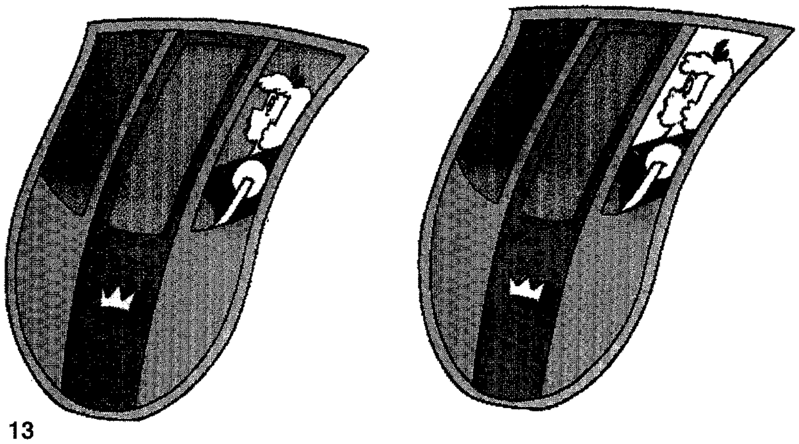
As mentioned before, after having computed a best assignment between the set of regions in the reference image and the set of regions in the image to be painted, the color information associated with each pair of matched regions is automatically established. The assisted painting process proceeds then recursively using each pair of matched regions as a basis for a new region matching and assignment step.

6.6 Results

The speed with which an image is colored using the assisted painting approach depends on the



12



13

Fig. 12. Assisted painting result for the example in Fig. 6

Fig. 13. Assisted painting: reference image (*left*) and automatically painted image (*right*) (original images by courtesy of France Animation)

two corresponding regions that are chosen to start the region matching and assignment process, since they determine the sequence in which the successive sets of matching candidate regions are determined and assigned. Therefore, the performance of the assisted painting can only be expressed in terms of average execution times, taken for different pairs of initial corresponding regions.

Figure 12 shows the result of the assisted painting stage for the example of Fig. 6. The image on the right was correctly painted, although some regions are missing (e.g., the eyes are closed) and others are significantly deformed (e.g., the hat's feather). The average execution time for these two images is approx. 4 s on a SUNSPARCstation10.

An additional example (see Fig. 13) shows the applicability and efficiency of our approach. Both images are now larger (approx. 1 million pixels each) and more complex. The average execution time is approx. 17 s, and the result has 92% correct region assignments. Note that a region associated with part of the right window pane was not painted at all in the second image; this is due to the fact that the corresponding part of the window pane in the reference image is made up of two regions; therefore, no region matching and assignment can be correctly achieved in this case, without bringing further information into play.

In comparison to commercially available tools, our approach is more robust, needing only a few interactions. It is also not too demanding in terms of computing performance, so it can be used on low-cost hardware. Although no error-free results can be guaranteed, the execution times associated with the developed approach, which is 2–4 times faster than a fully interactive painting, leave enough free room for corrections. Thus, our approach is competitive regarding the traditional interactive painting.

7 Vectorization

The vectorization of raster data is one of the most relevant data acquisition techniques. Since our approach starts with the scanning of hand-drawn images, a fully automatic vectorization stage is needed to allow easy and comfortable editing of the digitized images, the generation of in-betweens, the merging of digitized data with additional vector input, higher data compression and fast image rendering.

In the case of cartoon images two main requirements can be formulated: the topology of the image's contents has to be preserved, and the line thicknesses have to be approximated with a given error bound. The former is needed especially for edition purposes. The latter is essential to preserve artistic features, since animators often express the characters' moods by drawing the lines in a certain way.

To ease the vectorization it was decided to base it on already existing processing steps needed for the assisted painting:

- Computation of the distance map, which allows extraction of line-thickness information
- Distance-based skeletonizing, allowing accurate vectorization since the skeletal lines are placed on (or near) the medial axis of the original lines
- Automatic segmentation, allowing extraction of region adjacency relationships

The two steps are needed for image vectorization:

- Extraction of the topological information associated with the image and its representation in an appropriate data structure
- Approximation of the line shape information

7.1 Data structure

The two-dimensional cartoon drawings can be represented in a winged-edge data structure reflecting the topology of their contents. The face-edge data structure (Weiler 1985) is a particularly efficient one. The correspondence between the image entities and data structure elements is shown in Fig. 14.

The data structure is automatically and consistently built up from a thinned and segmented image, whose regions and contours are now associated with faces and oriented loops. Skeleton endpoints and branchpoints (i.e., knots) are represented by vertices, and a contour piece between two knots is represented by an edge. Since it can be traversed in the two opposite directions, such a directed contour piece is called "edgehalf". The buildup of the data structure is carried out by following the contours associated with each image region to identify their edgehalves and vertices and to establish the appropriate adjacency relationships.

Although the image is represented in Weiler's data structure, a vector representation is not yet achieved. Based on the previously extracted skeletal coordinates and the associated thickness values, an edgewise approximation to the original lines now has to be computed.

7.2 Approximation

The basic idea is to compute a three-dimensional curve that approximates, within a given tolerance, the data points defined by the sequences of (x, y) coordinates and thickness values associated with each edge. Each approximating curve is constructed as a piecewise Bézier curve (Farin 1990) with certain continuity conditions at its joints. Without any loss of generality, curves of degree three are used to explain the approach taken, which is a canonical extension of the works of Schneider (1990) and Pudet (1992) to three dimensions.

A Bézier curve of degree three is defined as

$$Q(u) = \sum_{i=0}^{3} P_i B_{i,3}(u),\ u \in [0,1] \tag{12}$$

where the P_i are the curve's control points and the $B_{i,3}(u)$ are the Bernstein polynomials of degree three. Equation (12) can be rewritten as

$$\begin{aligned} Q(u) = {} & P_0 B_{0,3}(u) + (P_0 + \sigma_0 t_0) B_{1,3}(u) \\ & + (P_3 - \sigma_1 t_1) B_{2,3}(u) + P_3 B_{3,3}(u) \end{aligned} \tag{13}$$

where t_0 and t_1 are the unit tangent vectors at P_0 and P_3, respectively, and σ_0 and σ_1 are positive constants defined by

$$\sigma_0 t_0 = (P_1 - P_0) \tag{14}$$

$$\sigma_1 t_1 = (P_3 - P_2) \tag{15}$$

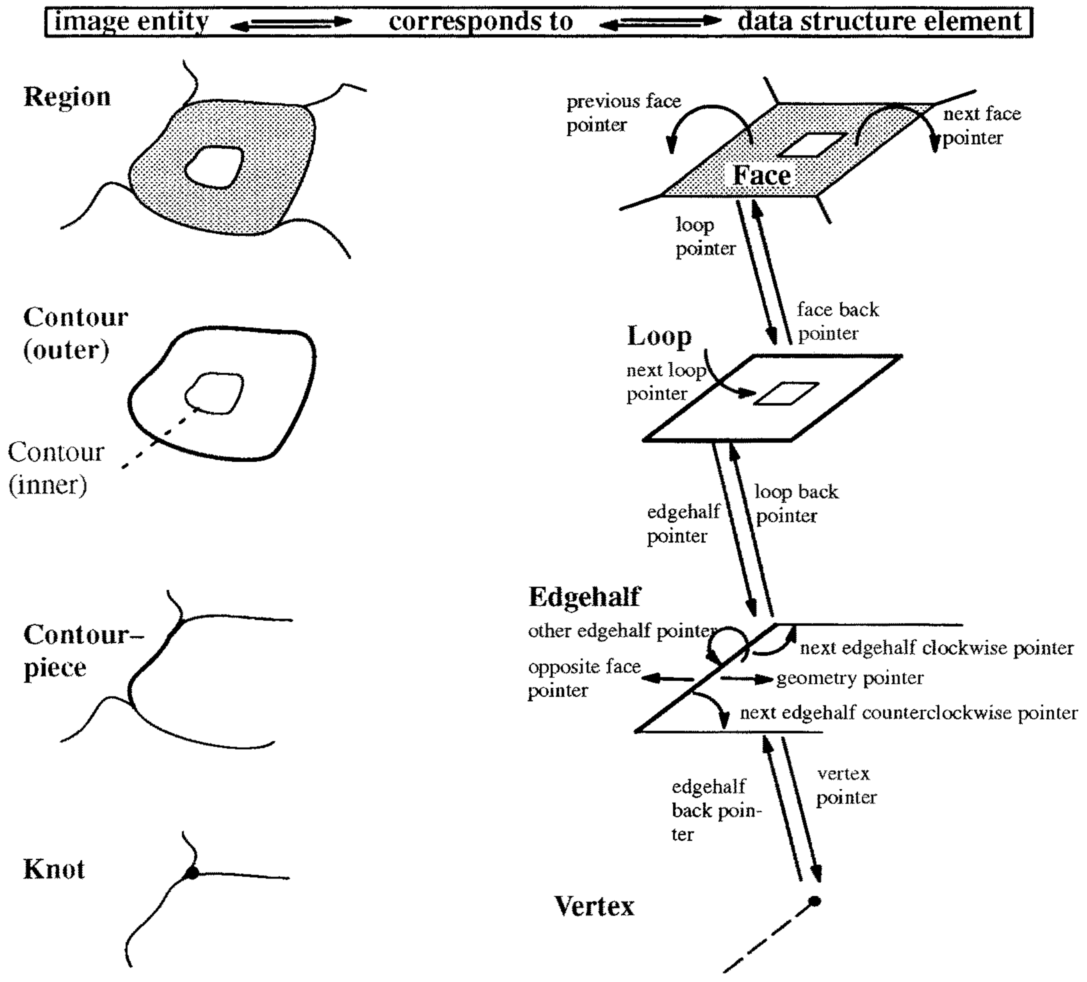


Fig. 14. The vectorization data structure

Choosing the minimization of the sum of the squared distances from the data points to their corresponding points on the approximating curve as the fitting criterion, the following error function has to be minimized:

$$S = \sum_{i=0}^{m} \|D_i - Q(u_i)\|^2 \tag{16}$$

where the D_i are the data points and u_i is the parameter value associated with point D_i. The chord-length parametrization is used in most cases. According to the properties of the Bézier curves (Farin 1990), the first and last control points of the approximating curve are the first and last data points:

$$P_0 = D_0 \tag{17}$$

$$P_3 = D_m \tag{18}$$

If the vectors $\boldsymbol{t}_0$ and $\boldsymbol{t}_1$ are estimated from the data, the remaining unknowns of Eq. 16 are the constants σ_0 and σ_1. Thus, minimizing the error function S corresponds to solving the following linear system:

$$\begin{cases} \dfrac{\partial S}{\partial \sigma_0} = 0 \\ \dfrac{\partial S}{\partial \sigma_1} = 0 \end{cases} \tag{19}$$

Afterwards, the control points P_1 and P_2 are computed using σ_0 and σ_1 and the unit tangent vectors $\boldsymbol{t}_0$ and $\boldsymbol{t}_1$. An approximating curve is then completely defined.

Using the described least squares data approximation algorithm, a recursive procedure is executed to compute a piecewise approximating curve, given the set of data points and the associated parametrization:

1. Estimate the unit tangent vectors at the first and last data points.
2. Compute the approximating curve using the least-squares algorithm.
3. Compute the maximum approximation error.
4. If this error exceeds the given tolerance, subdivide the data set and proceed recursively.

The data-splitting step can be accomplished in one of two ways: subdivide the data at the point of maximum error (Schneider 1990) or at its midpoint (Pudet 1992). To avoid performing too many subdivisions, and thus generating a large number of curve pieces, a parametrization refinement can be carried out (Schneider 1990). Clearly, the number of required subdivisions – and the final number of curve segments – is determined by the shape of the discrete curve represented by the 3D data points and the error tolerance. Note that by appropriately computing tangent vectors it can be checked if the data splitting point is a corner or not: in the former case, only positional continuity is associated with the adjoining curve segments, while in the latter first-order continuity is defined.

Before computing the approximation the sequence of data points should be filtered to reduce the rasterization effects and make tangent computation more reliable. Corner detection can also be performed to split a priori the data sequence; again, at the detected corner points the approximating curve will only have positional continuity.

Given the 3D control points for each curve segment, a recursive subdivision algorithm is used to generate a sequence of points which approximates the given curve within a certain tolerance (Lane and Riesenfeld 1980). The (x, y) coordinates of each one of these points approximate the skeleton of the corresponding edge, while the third coordinate approximates the line thickness. Reconstruction of the original thick lines is done by rendering filled polygons. Each polygon is built up taking two successive points of the skeleton approximation; the corresponding thickness values are marked perpendicular to the tangent at the points spanning the polygon. Special attention must be paid to the rendering of the lines endings.

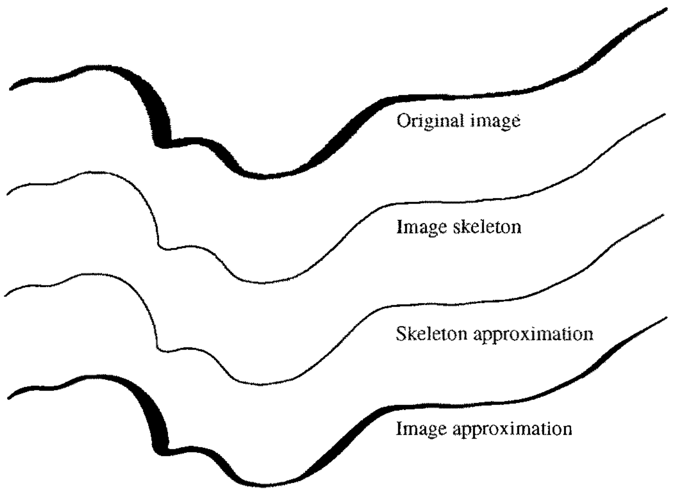
7.3 Results

In order to give an idea of the performance of the data structure buildup and of the approximation steps, two images of different sizes and contents are presented in Figs. 15 and 16.

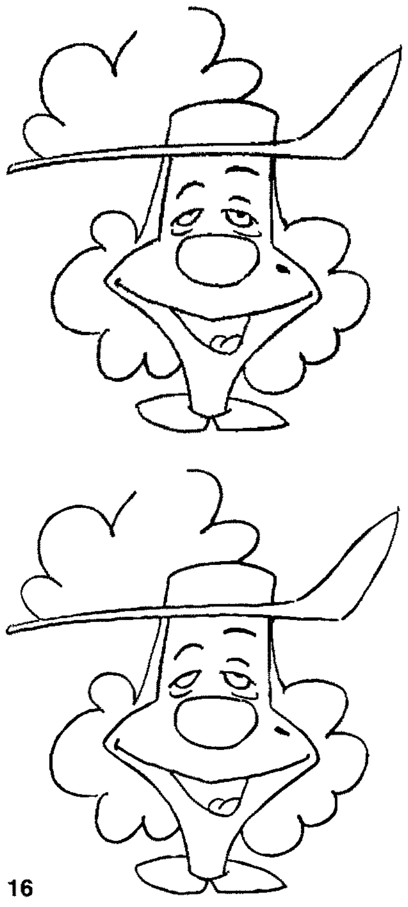
The original scanned image in Fig. 15 has 249×825 pixels and contains 7,766 black ones. The preprocessing steps needed as a basis for the data-structure build up – cleaning, skeletonizing, region segmentation and contour following, as well as classification of the skeleton pixels to allow vertex extraction – take on average 12.6 s. In contrast, the data structure buildup takes on average just 1.6 s; note that this is an extremely simple image with just one region, one loop, two edgehalves and two vertices. The approximation step takes 2.3 s; the 3D data points – there are 858 skeleton pixels – are approximated by 49 Bézier curves of degree three, within a 0.5 error bound. The image skeleton, its approximation, and the reconstructed image, which is quite visually pleasing, are also depicted in Fig. 15.

Figure 16 shows the second example; the original scanned image has 532×516 pixels, of which 16,384 are black. The preprocessing steps take on average 20.2 s and the data structure buildup 2.7 s; 12 faces, 24 loops, 178 edgehalves and 90 vertices (46 endpoints and 44 branch points) are defined in the image. The approximation step takes 10.2 s; the 4,472 data points are approximated by 369 Bézier curves of degree three, within a 0.5 error bound. Note that some reconstruction deficiencies are recognizable in the eyes and the hat.

Significant storage gains are achieved by the presented vectorization approach. For the first example, the 7,766 black pixels defining the first image are represented after the approximation by the 196 three-dimensional control points defining the Bézier curves; in this case, since the image defines just one topological edge, there is no need



15



16

Fig. 15. Approximation example

Fig. 16. Image reconstruction: original (*top*) and approximation (*bottom*) (original image by courtesy of France Animation)

to store twice the junction points between adjacent curves; thus, only 127 of the control points actually have to be stored.

Although the reconstructions show only minor errors, artifacts are unavoidable in cases where both high curvature and rapid changes in thickness occur. The use of separate filters for the shape and thickness information seems to provide more visually pleasing results.

8 Further work

Further work will be carried out in three distinct areas: assisted painting, vectorizing and in-betweening. For assisted painting new region-shape comparison methods are to be investigated and integrated into the current framework, aiming at higher accuracy and faster matching times. Another idea is to use the available vectorial representation as a basis for the region comparison.

An additional topic is the development of more precise line approximation and reconstruction methods. It would also be important to provide the user with a tool that presents full editing capabilities, not only allowing editing of the image topology, but also editing of the line shape and thickness information.

Considering the restrictions of fully automatic intermediate frame generation, a tool for interactively supported key-frame animation is currently under development, based on the vectorial representations of the key-frames.

9 Conclusion

An approach towards computer-aided cartooning has been presented that aims at supporting the drawing and painting stages of traditional cartoon production. The sequence of operations allowing the (semi-)automatic coloring of a sequence of digitized images has been thoroughly described.

Since it was decided to accept hand-drawn images as input, appropriate image processing algorithms were selected and integrated into a pipeline to accomplish the required image preprocessing. This preprocessing stage is fully automatic, thus allowing off-line processing with a satisfactory execution time. By including some feature extraction steps at the end of this stage, the execution time of later steps requiring some user intervention is considerably reduced.

The (semi-)automatic coloring stage substantially accelerates the production of traditional cartoons: instead of the user having to spend significant time to paint each image, this task can be performed in seconds. This assisted painting is based on the extraction and coding of shape information for each image region and on topologically guided shape matching between sets of regions defined in the painted reference image and in the image to be painted. After obtaining a measure of the dissimilarity between pairs of regions, an assignment is performed which minimizes the sum of the dissimilarity values associated with the assigned regions. Thus, the color information can be propagated from each region in the reference image to its corresponding region in the image to be painted. Clearly, corrections are needed in some cases, because either wrong regions were assigned – due to their similarity – or no regions in the reference image exist that match the remaining regions in the image to be painted. However, even if corrections have to be done by the user, our approach to automatic painting is still considerably faster than fully interactive coloring.

When aiming at supporting in-betweening, an image-vectorization stage is needed to build up an appropriate image representation, Two steps are accomplished in an automatic way: extraction of the appropriate topological relationships between the various image elements and approximation of the line shape and thickness information. Once the shape and topological information is stored, appropriate in-betweening and image-editing tools can be developed, which maintain the consistency of the image representation.

The examples presented illustrate the performance and applicability of the developed approach to support important stages of traditional cartoon production without using special, dedicated hardware.

Acknowledgements. The authors would like to thank Prof. J.L. Encarnação for his constant support. Thanks are also due to the students who helped implement some of the software modules and to Louis Laborelli and Patrick Courounet of the Institut National de l'Audiovisuel (France) for helpful comments and discussions. The work reported was carried out in the context of Animation 2000, which is a joint research and development project undertaken by France Animation (France), Getris Images (France), Institut National de l'Audiovisuel (France), and the Computer Graphics Center (Germany). Animation 2000 is supported by the Media Investment Club in association with Cartoon (Media Programme of the European Union) and Anvar (France). The first author's stay at the Computer Graphics Center in Darmstadt is being supported by JNICT/CIENCIA scholarship BD/2205/92-IA.

References

Baecker RM (1969) Picture-driven animation. Proceedings of the Spring Joint Computer Conference. AFIPS Press, Montavale, NJ, pp 273–288

Burtnyk N, Wein M (1971) Computer generated key frame animation. J Soc Motion Pict Television Eng 80:149–153

Burtnyk N, Wein M (1976) Interactive skeleton techniques for enhancing motion dynamics in key frame animation. Commun ACM 19:564–569

Catmull E (1978) The problems of computer-assisted animation. ACM Comput Graph 12:348–353

Durand CX (1991) The "TOON" project: requirements for a computerized 2D animation system. Comput Graphics 15:285–293

Farin G (1990) Curves and surfaces for computer aided geometric design: a practical guide, 2nd edn. Academic Press, San Diego London

Gass SI (1975) Linear programming: methods and applications, 4th edn. McGraw-Hill, Kogakusha, Tokyo

Gonzalez RC, Wintz P (1987) Digital image processing, 2nd edn. Addison-Wesley, Reading, Mass

Haralick RM, Shapiro LG (1992) Computer and robot vision, vol 1. Addison-Wesley, Reading, Mass

Haralick RM, Sternberg SR, Zhuang X (1987) Image analysis using mathematical morphology. IEEE Trans Pattern Anal Mach Intell 9:532–550

Heckbert PS (1990) A seed fill algorithm. In: Glassner AS (ed) Graphics gems. Academic Press, San Diego London, pp 275–277

Kochanek DHU, Bartels RH (1984) Interpolating splines with local tension, continuity and bias control. ACM Comput Graph 18:33–41

Lane JM, Riesenfeld RF (1980) A theoretical development for the computer generation and display of piecewise polynomial surfaces. IEEE Trans Pattern Anal Mach Intell 2:35–46

Lasseter J (1987) Principles of traditional animation applied to 3D computer animation. ACM Comput Graph 21:35–44

Levoy M (1977) A color animation system based on the multiplane technique. ACM Comput Graph 11:65–71
Leymarie F, Levine MD (1992) Fast raster scan distance propagation on the discrete rectangular lattice. CVGIP Image Understand 55:84–94
Liu H-C, Srinath MD (1992) A string descriptor for matching partial shapes. In: Shapiro L, Rosenfeld A (eds) Computer vision and image processing. Academic Press, San Diego, pp 575–592
Niblack CW, Gibbons PB, Capson DW (1992) Generating skeletons and center lines from the distance transform. CVGIP: Graphic Models Image Process 54:420–437
Pudet T (1992) Dessin à main levée et courbes de Bézier: comparaison des algorithmes de subdivision, modélisation des épaisseurs variables. PhD Thesis, Université de Paris Sud (XI), Centre d'Orsay, France. Available as Research Report 23, DEC Paris Research Laboratory, Paris.
Reeves WT (1981) Inbetweening for computer animation utilizing moving point constraints. ACM Comput Graph 15:263–269
Schneider PJ (1990) An algorithm for automatically fitting digitized curves. In: Glassner AS (ed) Graphics gems. Academic Press, San Diego London, pp 612–625
Sederberg TW, Greenwood E (1992) A physically based approach to 2-D shape blending, ACM Comput Graph 26:25–34
Sederberg TW, Gao P, Wang G, Mu H (1993) 2-D shape blending: an intrinsic solution to the vertex path problem. Proc SIGGRAPH '93 (Annual Conference Series). ACM SIGGRAPH, New York, pp 15–18
Stern G (1979) SoftCel – an application of raster scan graphics to conventional cel animation, ACM Comput Graph 13:284–288
Thomas F, Johnston O (1981) Disney animation: the illusion of life. Abbeville Press, New York
Wagner RA, Fischer MJ (1974) The string-to-string correction problem. JACM 21:168–173
Weiler K (1985) Edge-based data structures for solid modeling in curved-surface environments. IEEE Comput Graphics Appl 5:21–40
Wertheimer M (1921) Untersuchungen zur Lehre von der Gestalt. Psychologische Forschung I

JOAQUIM MADEIRA graduated in Electrical Engineering in 1986 and received an MSc degree in Computer Science in 1991, both from the University of Coimbra, Portugal. He joined the Computer Graphics Group of the Mathematics Department of the University Coimbra in September 1986. Since October 1992 J. Madeira has been working towards his Ph.D. at the Computer Graphics Center in Darmstadt, Germany. His current main research interest is computer-supported cartooning. He is a member of the ACM, EUROGRAPHICS and the IEEE.

ANDRÉ STORK is a researcher at the Department for Industrial Applications of the Fraunhofer Institute for Computer Graphics in Darmstadt, Germany. He graduated at the Fachhochschule in Darmstadt in Computer Science in 1992. From 1992 to July 1994 he was with the Visual Computing Group of the Computer Graphics Center (ZGDV, Darmstadt), where he was involved in a computer-supported cartooning research project. His current research interests are advanced interaction and visualization techniques in 3D-CAD.

MARKUS GROß received a Dipl.-Ing. degree in Electrical Engineering in 1986 and a Ph.D. on Computer Graphics and Image Analysis in 1989, both from the University of Saarbrücken Germany. In 1990, he joined the Computer Graphics Center in Darmstadt, where he established and directed the Visual Computing Group. Since July 1994, he has been as Assistant Professor at the Computer Science Department of the ETH-Zürich. His research interests include scientific visualization, rendering techniques, wavelets and neural networks for visual pattern recognition and computer vision. Dr. Groß has published numerous refereed technical papers on computer graphics and image analysis, and he is the author of the book *Visual Computing*, Springer, 1994. He is a member of the editorial advisory board of *Computers & Graphics*, Pergamon Press. In 1987, he received the VDE-Saar award, and in 1991 the best papers award of the German Institute for Electrical Engineers (VDE). He is a member of the IEEE and VDE.