SnakeToonz : A Semi-Automatic Approach to Creating Cel Animation from Video

Aseem Agarwala[1]

Starlab NV
http://www.agarwala.org

Abstract

SnakeToonz is an interactive system that allows children and others untrained in cel animation to create two-dimensional cartoons from video streams and images. The ability to create cartoons has traditionally been limited to professional animation houses and trained artists. SnakeToonz aims to give anyone with a video camera and a computer the ability to create compelling cel animation. This is done by combining constraints of the cartooning medium with simple user input and analysis of that input.

A cartoon is created in a dialogue with the system. After recording video material the user sketches contours directly onto the first frame of video. These sketches initialize a set of spline-based active contours which are relaxed to best fit the image and other aesthetic constraints. Small gaps are closed, and the user can choose colors for the cartoon. The system then uses motion estimation techniques to track these contours through the image sequence. The user remains in the process to edit the cartoon as it progresses.

CR Categories:
I.3.8 [Computer Graphics]: Applications— [I.4.9]: Image Processing and Computer Vision—Applications

1 Introduction

Cel animation is a difficult and time-consuming medium to work in. Kids love drawing cartoon-like sketches, but have never been able to draw an entire animation. SnakeToonz allows children and other untrained users to create animated cartoons using a combination of the child's own video material and sketching.

We use *active contours*, a computer vision technique informally known as *snakes* [Kass et al. 1987], as the basic primitives in our cartoon animations. A snake is a two-dimensional contour which is relaxed to a minimal energy configuration; the energy function is typically based on the greyscale values of an image so that the curve conforms to a certain object or feature in the image. Snakes are an active area of research in computer vision.

Two-dimensional cartoons are currently made in two ways. Television and feature-film cartoons are hand-drawn by animation houses with large teams of highly-trained artists at a very high cost [White 1988]. Web animation, usually streamed in the Flash format, is a quickly growing alternative. These cartoons are created by graphic designers using complicated design software. It has been estimated that it takes between 10 and 30 hours to create a single minute of Flash animation, notwithstanding the training and experience involved [Paul 2000].

Creating a compelling cartoon normally requires the ability to draw well, the ability to create and plan attractive motion trajectories, and lots of time. The medium is far out of the reach of the average person because of the high effort required. A recent and exciting focus in computer graphics research has been the attempt to give children and untrained users the ability to express their creativity in new mediums and at quality levels previously unattainable [Igarashi et al. 1999; Anderson et al. 2000]. These systems combine simple human input with computational analysis to create media that is better than either the computer or untrained user could have achieved alone. SnakeToonz brings this approach to the medium of 2D cartoons. By combining human input with video analysis and motion estimation, SnakeToonz creates cartoons more quickly and easily than traditional methods. The downside of such a semi-automatic technique, of course, is that the flexibility and quality levels of a fully manual approach are partially sacrificed.

Our approach is inspired by common tracing techniques. Children often create much better drawings than they could create alone by laying semi-transparent paper over an image and tracing it. Even trained animators occasionally use rotoscoping [Maltin 1980], where cartoons are traced from film projected onto their desk, to handle especially complicated sequences; Disney's Snow White used this technique extensively [Thomas 1976]. However, cartoon rotoscoping still requires the animator to hand-draw each frame of the animation, a process too laborious for children. SnakeToonz combines this process of creating cartoons with video analysis. The result is a method of creating a cartoon that requires little expertise, but still allows plenty of creative freedom for the user.

Video motion estimation is far from a perfect art, however, and works best when tuned to a specific application. Thus, our goal scenario is to allow children to create their own cartoons using their own toys as actors and actresses. The child can act out scenes with their toys as puppets in front of a video camera, and then use SnakeToonz to draw cartoons from this video material. Creating cartoons becomes a fun learning activity for children.

SnakeToonz creates animations that are entirely vectorial. The result is animation that can be streamed on the internet over low-bandwidth in a vector format such as Flash. It can also be rendered at any resolution without faceting, as parametric curves are used as the basic primitives.

The process of creating a cartoon is modeled as a dialogue between child and computer. The child first creates a cartoon of the first frame of video by drawing curves directly on the image. The system responds by modifying the drawn curves to best fit the edges in the image as well as other aesthetic constraints. The system also snaps together small gaps between drawn curves. The child can advance to the next frames as the system attempts to automatically propagate the cartoon using video motion estimation. He or she can

[1]Currently at Department of Computer Science and Engineering, University of Washington, Seattle WA. Email: aseem@agarwala.org

then edit the system's suggestion, if necessary, and is free to add or delete curves as occlusions occur or new perspectives of objects appear.

This paper is organized as follows. After discussing previous work, the process of creating a single frame of cartoon is presented. We then show how the single-frame cartoon is propagated into a multiple-frame animation. Finally, we present results and conclusions.

2 Previous Work

We have found little published work dealing specifically with the application of active contours to animation; a notable exception is the work of Hoch and Litwinowicz [1996], where snakes were used to track the facial features of an actor for driving animated characters. The main difference from our system is that the active contours were not used directly as primitives for animation.

We draw inspiration from several systems that allow children to play in otherwise complicated and advanced mediums. Teddy [Igarashi et al. 1999] allows children to create 3D models from sketching, and the work of Anderson et al. [2000] gives children tangible access to modeling in 3D using blocks and clay.

We are also heavily indebted to a large body of research from the computer vision community in active contours and tracking. Snakes were first developed by Kass et al. [1987]. An in-depth treatment of recent active contour research can be found in a book by Blake and Isard [1998]. Spline-based snakes were first explored by Menet et al. [1990]. We use a combination of these techniques, as none were specifically suited to our requirements.

Finally, we are also inspired by the large and recent effort in non-photorealistic techniques for computer graphics. Much of this effort has focused on the rendering of 3D models, including cartoon-like rendering [Kowalski et al. 1999]. However, these techniques are less accessible to children and untrained users, as 3D modeling and animation is a complicated task. On the other hand are systems that work with images and video. Some of these systems conduct automatic transformations [Hertzmann 1998; Litwinowicz 1997]; we are more interested in combining computation with the spark of human creativity, such as in interactive systems for creating illustrations from images [Salisbury et al. 1994; Salisbury et al. 1997; Ostromoukhov 1999]. These efforts are the most similar to ours in inspiration, though they deal with very different styles, and only with single images.

3 Drawing One Cartoon Frame

The process of creating an animation starts with drawing a cartoon from the first frame of video. In our simple model, a cartoon image consists of solid regions of color demarcated by curves. We assume that the salient curves in a cartoon representation of a photographic image will lie on strong edge features in the image. This assumption may not always hold, but in our applications involving videos of toys, it has proven to be the case.

To create a cartoon the child draws contours directly onto the photographic image. In a sense, the user is identifying features in the image that he or she wishes to include in the animation. The child has complete control over these decisions, which is useful in creating cartoons of toys; the child does not need to include the hand holding the toy, so that the cartoon characters will seem to move on their own.

The problem is that the child will not generally draw a curve that perfectly follows curves in the image; nor will he or she exactly adjoin endpoints so as to clearly define closed regions for coloring. The former is especially a problem for tracking the contours over time; contours that do not lie directly on high-frequency regions of image data are difficult to track. To this end, we must modify the child's sketching to best fit the edges in the data. We must also exactly align adjacent endpoints of contours so as to close gaps and form closed regions, a technique we call *snapping*. We discuss these two techniques in the following two sections. Of course, interactive controls must exist to allow the user to manually correct any changes the system might make. These tools are discussed in Section 3.4. Finally, once the contours of a cartoon image are well defined, the user can enter a coloring mode and set colors for the closed regions. This is discussed in Section 3.3.

3.1 Snakes

Traditional snakes are represented as many points connected by line segments. Such contours, however, are not visually attractive as they exhibit faceting and do not scale well over different resolutions. We use connected piecewise cubic Bézier splines as the representation of the snake contours. In addition to their visual advantages, spline-based snakes have another benefit. Their representation is more compact, which means the optimization space has lower dimensionality. Another common approach to user-guided contour extraction uses a live-wire boundary, such as Intelligent Scissors [Mortensen and Barrett 1995]. However, this technique attempts to be pixel accurate, and thus suffers similiar limitations in the visual appearance of the resultant contours.

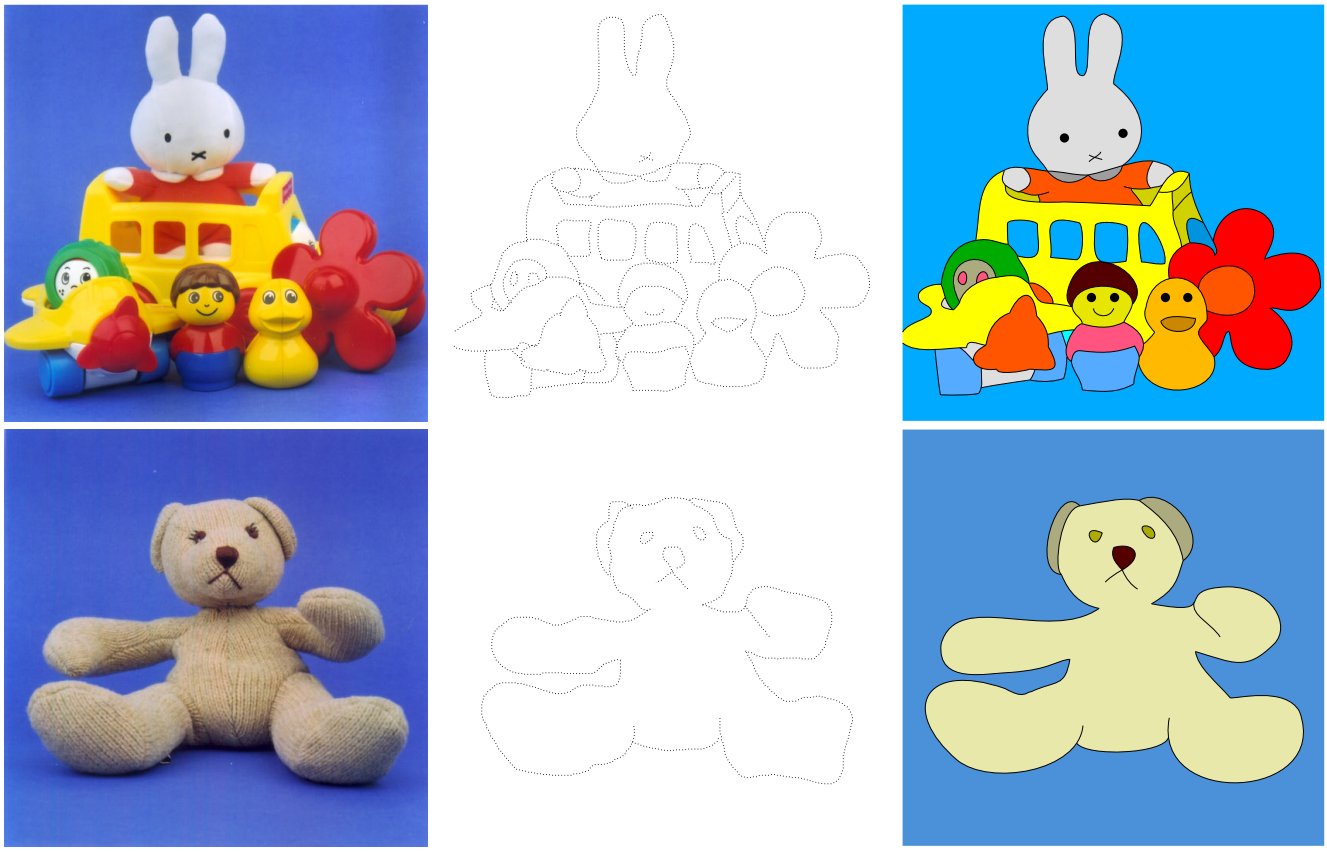
To start the process, the user traces a contour directly onto the image. We assume that the user has placed the first and last points of the contour correctly, and do not allow the system to change their location (except for snapping, as discussed later). This is because fixed endpoints improve the results of snake relaxation. The user can move these points interactively if he or she is not happy with their initial locations. We then fit piecewise Bézier splines to the points in the user's sketched stroke using the technique of Schneider [1990a]. This method uses iterative least squares minimization to fit a Bézier spline to the set of points. If this spline cannot fit the data to within a specified tolerance the spline is split in half and the method proceeds recursively. For an untrained user this approach is more intuitive than directly specifying the control points of a spline.

Once the contour is initialized we relax the snake to best fit the edges in the images and to maximize other aesthetic objectives. Many techniques for optimizing snakes exist [Kass et al. 1987; Amini et al. 1990]; we chose a simple, greedy technique similar to Williams and Mubarak [1992]. We iterate over each control point in the snake, except for the first and last which are frozen. We calculate an objective function for each location in a 3x3 neighborhood around the current location, and move the control point to the location that minimizes this function.

The objective function is calculated over those portions of the contour that are affected by the location of the control point in question. For the interior points of each Bézier segment, only the current Bézier segment is affected. For the end points of each Bézier segment, two segments are affected. The objective function consists of three main terms which are weighted and linearly combined:

1. **Edge Overlap:** This term is the negative of the line integral of the image gradient along the Bézier segment. It is calculated discretely using adaptive sampling of the Bézier curve [de Figueiredo 1995], and is normalized by the length of the curve. This term pushes the snake towards edges in the image.

2. **Curvature:** The curvature term, which discourages convoluted splines, is calculated by first taking the average of the three vectors formed by subtracting adjacent control points in the cubic Bézier segment. The term is the sum of the squared differences between these three vectors and their average, normalized by the length from the first control point to the last.



(a) **(b)** **(c)**

Figure 1: Creating cartoon images of toys. (a) The original photographs. (b) The user's tracing. (c) The finished, colored cartoons.

This term is smallest when the three vectors are equivalent, which is the Bezier representation for a straight line.

3. **Corner Sharpness:** This term, which is weighted lightly, discourages corners between connected Bézier segments unless the image data strongly encourages it. It is simply the angle between the tangent lines of the Bézier segments where they connect.

The image gradient data used in the edge overlap term is calculated in a pre-processing step. We first clean the video by taking each frame as an image and de-interlacing it; every other line of image data is discarded. Next, we apply the SUSAN [Smith and Brady 1997] structure-preserving noise reduction filter to each image. This filter blurs the image while still preserving strong edges; though not a necessary step, it tends to improve results from grainy video. SUSAN operates on greyscale images, so we apply the filter independently in the three color planes. Finally, we apply a simple Sobel edge detection filter to the image to calculate the gradient.

In addition to the three main terms we add several penalty terms in certain rare cases that we wish to discourage. For one, we do not want points of extremely high curvature to exist along a Bézier segment, as these can correspond to a cusp or other unsightly features. So we add a strong penalty when this occurs. Calculating points of maximum curvature along a cubic Bézier segment in closed form is not practical. So we again adaptively sample [de Figueiredo 1995] the curve, and find the maximum angle between the resultant line segments. Another penalty term is necessary to avoid confusion in the definition of closed regions for coloring. We do not want Bézier segments whose endpoints are snapped together (see Section 3.2) to depart from that endpoint in nearly the same direction. So we penalize this fairly rare condition.

The relaxation of the snake using this objective function continues until either a local minimum is reached or the user clicks the mouse to signal that he or she is satisfied with the contour position. A cooperative user eliminates the need for termination criterion.

When the relaxation of the snake is finished, we make one more attempt to minimize its representation. It is common that a section of the user-drawn curve that could not be represented by a single Bézier segment before relaxation can now be; this corresponds to unnecessary wiggles in the user's hand movements that were resolved during relaxation. So, we attempt to join together adjacent Bézier segments if this can be done within a certain tolerance. This is done by generating a set of points from the two segments using adaptive forward differencing [Lien et al. 1987] to avoid overweighting any part of the curves. We attempt to fit a single Bézier segment to this point set using Schneider's [1990a] least squares technique. If this can be done within a certain tolerance, we replace the two segments with one.

The final result is a cartoon contour that closely matches the user's desires and the data in the image while maintaining aesthetic properties such as smoothness. This process can be used to create single cartoon images as well as frames in an animation. A child could, for example, create cartoon images of the toys in his or her collection. Two examples are shown in Figure 1. The images showing user input are a log of the user's tracing directly over the original image. The second example shows a more extreme case of poor user input.

3.2 Snapping

When the user draws a curve that is supposed to be attached to another, it is unlikely that the user will be able to exactly position the curve so that it is attached. This "gap closing" problem is common in cartooning applications [Gangnet et al. 1994; Fekete et al. 1995], and is important for two reasons. One, gap closing improves the look of the final cartoon. Two, it closes off regions so that they may be colored without leaks.

We solve the problem by simply detecting endpoints placed very near each other during interactive drawing and snapping them together. We also keep data structures recording these snaps, so that we can keep snapped points together during interactive manipulation and tracking through multiple frames. There are four types of snaps that are handled.

1. **Edge Snapping:** If the user starts or ends a curve very near the edge of the image, it is likely that the user wishes the curve to attach to the edge. This will define two separate regions bounded by the curve and the edge of the image.

2. **Close Snapping:** The user may be drawing a closed curve that ends where it starts. If the last point of the curve is very close to the first point, we snap them together.

3. **Endpoint Snapping:** The user may wish a curve to start or end at one of the endpoints of another curve. We thus check this possibility for both the first and last points of a newly-drawn curve. Note that we can snap to the endpoints of another curve as well as the internal corner points that form the intersection between connected Bézier segments.

4. **Curve Snapping:** The final case is when the user wishes the first or last point of a newly-drawn curve to attach to an arbitrary point along another curve. We must find the closest point on each of the other curves to the endpoint in question, and then determine whether the closest of these points is within a certain tolerance. The solution to this problem involves finding the roots of a fifth order polynomial, and we use the formulation as presented by Schneider [1990b].

3.3 Coloring

Once the cartoon contours are defined, the user can select a color for each closed face of the drawing. To do so, we must be able to define each closed region. What makes the coloring problem more difficult is that we cannot simply flood-fill each region; if we did, the resultant cartoons would no longer be entirely vector-based, making it impossible to transmit in a vector-format like Flash. Instead, we must be able to generate vectorial paths that describe each closed region. We must also solve the point-location problem; given a point in the image space, it must be possible to determine which closed region contains this point. This is necessary to allow the user to interactively select regions for coloring.

The version of this problem where all primitives are line segments is a common topic in computational geometry [de Berg et al. 1999], and is commonly referred to as a planar subdivision. The problem with Bézier splines was dealt with in depth in the work of Gangnet [1989]. However, their focus on exact arithmetic and efficiency is not necessary in our case. It is also not necessary to find intersections between splines, as the user is expected to end individual contours with snaps at their intersections. We thus build a doubly-connected edge list [de Berg et al. 1999], or *DCEL*, to aid us in defining closed regions. This data structure has the advantage that each counter-clockwise cycle represents one closed region, or face, while each clockwise cycle represents a hole that is entirely contained by an enclosing closed region. We also add the boundaries of the image into the data structure, so that the background can be colored by the user.

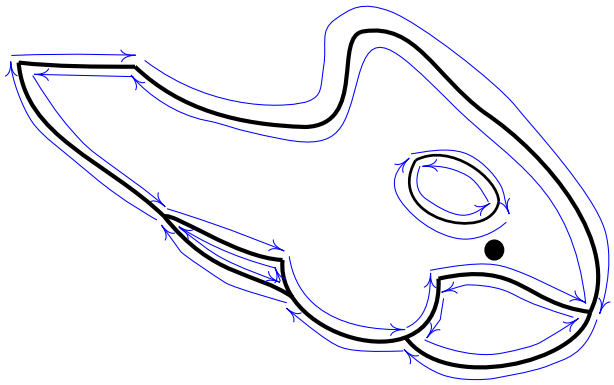


Figure 2: A doubly-connected edge list example.

3.3.1 Building the DCEL

A visual example of a DCEL is shown in Figure 2. The memory representation consists of three lists: vertices, directed edges, and faces. Each edge knows its origin vertex, its previous edge, and its next edge. Each edge is also aware if its twin, which is the edge traveling along the same contour but in the opposite direction. The key invariant to maintain when inserting and deleting edges is that during a clockwise traversal around a vertex, each incoming edge is followed by its next edge which is leaving the vertex. Thus, it is necessary to know the tangent vector of each spline at the vertices to maintain proper ordering of the edges at each vertex in the DCEL. We must also take care to split any contours into multiple segments around curve snaps, as these curve snaps must be vertices in the DCEL.

To build a DCEL for a frame of the cartoon, we first add all the vertices and edges while maintaining the properties of the DCEL. Next, we iterate over the edges and find all closed loops; that is, we find the closed loops that occur by traversing the next edge field until the original edge is found again. A face only needs to store a pointer to one edge in this closed loop. Each edge stores a pointer to the face on its left. After finding all the faces, we trim away any edges that are not part of a face. We determine whether each face is a contour-clockwise or clockwise loop; clockwise faces are holes, and the others are closed regions which can be colored by the user. Each counter-clockwise face must store pointers to any holes it contains, so as to fully define the geometry of a colorable face.

3.3.2 Using the DCEL

The result is a data structure that contains vectorial paths for each colorable region. We solve the point-location problem by casting a ray leftwards from the point in question, and use the closest edge hit that corresponds to an enclosing face. The edge hit does not correspond to an enclosing face if the face it points to is a hole, or if the cross product of the cast ray and the tangent vector to the edge at the intersection point indicates that this edge is part of another hole of the same face. Intersecting an axis-aligned ray with a Bézier spline can be done using de Casteljau subdivision [Salomon 1999]. The face pointed to by the found edge is thus the desired region for coloring. Note that this technique is also used to identify the face which encloses a specific hole.

Finally, it is necessary to interactively render these filled regions from their vectorial paths. The rendering of all splines in the sys-

tem is done using simple forward differencing [Salomon 1999]. To render a region of color, we iterate over the scanlines in the bounding box of the region expanded by one pixel in each direction. The locations of the intersections of the scanline with the splines composing the region are calculated using de Casteljau subdivision and then sorted in increasing order. Since we know the starting edge of the scanline is outside the region, a simple even-odd rule informs us of which intervals of the scanline need to be colored.

3.4 Interactive Tools

The user must have complete control over the geometry of the cartoon image. Therefore, several interactive tools are necessary. To begin with, a *move* tool allows the user to push and pull at the contours. If the user clicks on an endpoint of a Bézier segment he or she is allowed to move that endpoint. A more complicated case arises if the user wishes to edit the internal geometry of the curve. Again, we do not wish to require the user to understand the concept of control points for a Bézier spline. Instead, the user must be able to click directly on a curve, and push and pull on it. The endpoints should remain in position.

An approach to this problem can be found in the work of Fowler and Bartels [1993], which provides a general formulation for finding the minimum displacement to spline control points under a set of constraints. The first constraint, in our case, is that the user-selected point on the curve passes through its new location. The second and third constraints are that the endpoints do not move. The solution is as follows. If t is the parametric value along the Bézier segment the user is dragging, and the vector $\mathbf{\Delta p}$ is the change in position from dragging, then

$$\begin{aligned} \hat{f} &= 9(t^2(1-t)^4 + t^4(1-t)^2) \\ \mathbf{\Delta d} &= \frac{\mathbf{\Delta p}}{\hat{f}} \left[\begin{array}{cc} 3t(1-t)^2 & 3t^2(1-t) \end{array} \right]^T \end{aligned}$$

yields the 2x2 matrix $\mathbf{\Delta d}$ whose columns are added to the two internal control points of the Bézier segment.

Four other tools complete the interface. One tool allows users to add contours to the cartoon, and another erases contours. The user can also split two Bézier segments into two, or join two segments into one. Splitting segments is helpful after tracking over multiple frames, where perspective changes of the toy can reveal more complicated geometry than can be described by the current number of segments. Joining segments is helpful for the reverse situation.

4 Drawing a Cartoon Animation

Once the child has created a cartoon of the first frame of the video, he or she can work with the system to create a multiple frame animation. The user can click to the next frame, and the system will attempt to automatically track the drawn cartoon to the next frame. The user can accept this automatically-propagated cartoon frame, or edit it using the interactive tools. When significant events occur such as an occlusion, a new object entering the scene, a significant perspective change, or other such situations, the user must intervene to guide the process. Contours can be added or deleted at any frame, and the DCEL for every frame is built independently. Color choices are propagated automatically where possible (as discussed in Section 4.3).

To track the contours, we first copy the current frame's cartoon forward to the next frame along with the snapping information. We then use a point tracking technique to track the ends of the contours. Internal points are moved using a transformation space defined by the endpoints. The snakes are then relaxed to the edges in the new frame. We track by the endpoints since for our application the ends of the contours tend to lie at corners in the image that are easy to track. Also, since the user-specified endpoints are not moved during snake relaxation, it is necessary to track them as accurately as possible.

We first describe the point tracking technique used and the particulars of moving the internal points. We then discuss the relaxation of the snakes with an additional term to preserve temporal coherency.

4.1 Point Tracking

Tracking snakes by tracking their endpoints was first suggested by Hoch and Litwinowicz [1996]. However, their use of block matching through variance-normalized cross correlation met with little success in our experiments. Block matching is a process of exhaustively searching for a window of image data in frame $i+1$ that best matches the window in frame i centered at the point being tracked. The similarity of image windows is measured using sum-of-squared-differences. The main problem, besides slow performance, is that this technique does not account for any rotation of the window; it assumes that the small amount of motion that can occur between two frames of video can be modeled entirely by translation. While this is true for most point tracking applications, it is not true in our case; toys being moved by hand can have surprising amounts of inter-frame rotation.

We thus use the Shi-Tomasi feature tracker [1994] which tracks points with an affine motion model. In fact, the authors use their tracker limited to a translation model, as this leads to better stability. The full affine model was only used for measuring dissimilarity between the current and initial frame. However, their applications focused on stationary scenes and moving cameras. For our situation, the full affine model has worked well with little evidence of instability, while the tracker limited to the translational model did not perform well. The tracker operates on greyscale images.

The Shi-Tomasi tracker models the displacement δ of a point between two frames with an affine motion model

$$\delta = \mathbf{D}\mathbf{x} + \mathbf{d}. \tag{1}$$

where $\mathbf{D}$ is a 2x2 deformation matrix and $\mathbf{d}$ is a translation. Then, if a point $\mathbf{x}$ in the first image $\mathcal{I}$ moves to a point $A\mathbf{x}+\mathbf{d}$ in the second image $\mathcal{J}$ where $A = \mathbf{I}+\mathbf{D}$ and $\mathbf{I}$ is the 2x2 identity matrix, we can hope that:

$$\mathcal{J}(A\mathbf{x}+\mathbf{d}) = \mathcal{I}(\mathbf{x}) \tag{2}$$

The Shi-Tomasi tracker attempts to find the 6 motion parameters in $\mathbf{D}$ and $\mathbf{d}$ that minimize a least squares measure of dissimilarity between a window around the point being tracked in image $\mathcal{I}$ and the window of image data in image $\mathcal{J}$ define by the motion parameters. We thus minimize

$$\varepsilon = \int\int_W [\mathcal{J}(A\mathbf{x}+\mathbf{d}) - \mathcal{I}(\mathbf{x})]^{\mathbf{2}} \mathbf{dx} \tag{3}$$

where W is the window around the point (we use a 13x13 pixel window). To track in a pure translational model $\mathbf{D}$ is set to zero. Equation 3 can be linearized into a 6x6 linear system. The derivation and resultant linear system is too complex to discuss here, but may found in the references [Shi and Tomasi 1993]. Since a Taylor series is used to linearize equation 3 the solution is not exact. So, the tracker proceeds in a Newton-Raphson-style iterative minimization to refine the estimate until convergence.

We make two simple modifications to the Shi-Tomasi tracker. For one, the stability of the tracker is improved by a good initial estimate of the motion parameters. So, we record the velocity and acceleration of each point during tracking and use it to predict the

next position. In addition, we track through a two-level image pyramid. That is, we calculate an initial estimate of the motion parameters in a sub-sampled version of the image data. This estimate is then refined in the full resolution image. Image pyramids are common in tracking applications [Bergen et al. 1992] and are used by a freely available implementation of the translation-only Shi-Tomasi tracker [Birchfield 1998]. The sub-sampled images and the image gradient data needed by the tracker are pre-processed and stored.

Finally, the calculated translation $\mathbf{d}$ is added to the location of the point being tracked in the current frame. When tracking the endpoints of cartoon contours, the calculated $\mathbf{D}$ matrix is discarded since we only need translation results; $\mathbf{D}$ only exists to add degrees of freedom to the minimization problem. However, we also have the potential to add other types of features to the cartoon. In the current system a user can add small, filled circles which can represent eyes, as evident in Figure 1. These are tracked using the Shi-Tomasi tracker. Other similar types of features could be added, and the appearance of these features could be transformed by the results of the $\mathbf{D}$ matrix.

4.2 Handling Internal Control Points

The two extreme endpoints of the snake contour are tracked using the Shi-Tomasi tracker. However, when possible we also use this tracker to track the internal endpoints. The internal endpoints are those control points at the junction between two connected Bézier splines; these tend to lie at easily trackable locations. However, this is not always the case. We must decide when it is worth tracking these points, or when these points should simply follow the movements of their neighbors.

Fortunately the Shi-Tomasi tracker gives us an excellent measure of when a point will track well. If the minimum eigenvalue of the 2x2 Z matrix (see [Shi and Tomasi 1993]) is below a certain threshold, we can safely guess that the point will not track well. In this case, the point is skipped. We also abandon tracking of the point if the iterative refinement process becomes unstable, or if the difference between the image windows in images $\mathcal{I}$ and $\mathcal{J}$ are too large.

We propagate those control points that have not been tracked by a transformation matrix defined by the previous and the next control points that were tracked successfully. The transformation of the line connecting these two control points is calculated, composed of the translation of the center of the line and a rotation and scale about that center.

4.3 Snake Relaxation After Tracking

Once the cartoon contours have been automatically tracked to the next frame, we once again allow the snakes to relax against the new image data. Hopefully, if tracking was successful, these contours will not have to move much. This is, of course, not always the case. However, we want to make sure that the shape of the contours does not change significantly between frames, as this will harm the appearance of temporal coherency. So we add a shape deformation penalty to the objective function discussed in Section 3.1.

The shape deformation penalty term calculates a measure of the change in shape of the contour from its shape in the previous frame. The snake relaxation will thus discourage significant changes in shape. The measure of change in shape must be invariant to translation, rotation, and scale; that is, the measure should not complain about uniform transformations that do not affect our notion of shape. A common shape signature for such a situation is the *turning function* [Arkin et al. 1991; Wolfson 1990]. The turning function at a point along the curve is simply the angle that the tangent to the curve makes to a certain reference angle, such as the x-axis. The behavior of this function along the length of the curve is an excellent characterization of the curve's shape, and is invariant to uniform changes in translation, scale, or rotation. The shape deformation penalty is thus the sum of

$$\eta = \int_0^1 \left(\Theta_2(t) - \Theta_1(t)\right)^2 dt \tag{4}$$

for each Bézier segment in the contour, where $\Theta_2(t)$ is the turning function in the current frame and $\Theta_1(t)$ is the one for the previous frame. This is calculated discretely using Romberg integration [Press et al. 1992].

Once the contours have been tracked and the snakes have relaxed to a mostly stable state, the user can edit the resultant cartoon. This can involve both moving the endpoints tracked by the Shi-Tomasi tracker and pushing and pulling on the contour to the state desired by the user. The user can also delete and add new contours.

A new DCEL for coloring information is built for each new frame. To propagate coloring choices automatically from the previous frame, we maintain pointers between corresponding contours across adjacent frames. To propagate a color for a face, we simply find an edge of the face with a valid pointer to an edge in the previous frame, and use the color of the face that that edge bordered.

5 Results

Several frames of several cartoon animations created using SnakeToonz are shown in Figure 3 (see color plate). In some of the examples we have inserted background images behind the cartoon, which is easy since the animations are entirely vectorial. It is also possible to use real photographs or live action video as a background for a merging of cartoon and real imagery. Individual frames of a cartoon are saved by the system as encapsulated postscript. Five animations are shown on the accompanying video: a bear animation of 125 frames, a penguin puppet animation of 116 frames, a bunny animation of 100 frames, a whale animation of 98 frames, and a school bus animation of 89 frames.

We have also found it simple to print flipbooks, which provides a simple physical and portable embodiment of the child's animation. This creates an interesting cycle from physical play to a cartoon animation and back to a tangible trace of the original play.

We have found that it takes an adult an average of about 4 seconds per frame to create a cartoon, at 25 frames per second using the PAL video standard, which is an order of magnitude faster than current methods of cartoon animation. This 4 seconds includes a second of loading pre-processed data and calculating the motion estimation, and a second for snake relaxation. The user spends an average of 2 seconds of interactive editing per frame. Most frames require no editing, while a few require more involved effort. The longest effort is required when a new view of an object starts to appear; planning the cartoon changes in response to this can be fun and educational though time-consuming. The most common user edit, which takes little time, is correcting drift in tracked points which tend to appear every 20 to 30 frames.

We feel that the quality of the animations is high, but they are clearly much simpler than one would find in professional animation for television or film. However, the savings in time, training, and effort justify the losses in flexibility and quality for the casual, non-professional user.

6 Conclusion

6.1 Limitations

Our system works best on clean video with well-defined edges and few occurrences of occlusion. The approach of SnakeToonz will

never work on arbitrary sequences of video; a complicated crowd scene is one example of something SnakeToonz could not handle. Cel animation as a medium is not well-suited to representing fine detail, and this must be kept in mind when choosing video material.

Another limitation to note is that cartoons created with SnakeToonz are restricted to the realities of what can be captured in video. Traditional animation depends heavily on exaggeration, caricature, and other unrealistic motions and appearances that are difficult to record on tape.

6.2 Future Work

We began SnakeToonz with the the assumption that we could simply plug-in the current state-of-the-art in semi-automatic contour tracking; it soon became clear that this is not the case. Our results are fairly simple; they do not include significant 3D rotations, occlusions, or very complicated figures. Also, it is not clear to us that our system is ready for use by children; we feel that too much hand-editing of the tracking is required to keep children engaged. To address these limitations we need a significantly improved method for semi-automatic tracking.

However, contour tracking techniques seem to cluster around two poles. The computer vision community focuses on nearly automatic techniques without user input. On the other hand, professional rotoscoping software packages such as Commotion require significant and repetitive user effort.

Clearly there is a need for new, semi-automatic contour tracking techniques. We see many applications of such a tool beyond SnakeToonz. Most interactive systems for the non-photorealistic manipulation of images involve a combination of hand-drawn annotations with an analysis of image features. The corresponding task for video has not been significantly explored due to the added complexity of the time-axis; without tracking, these annotations need to be made for each frame. Thus, our main plan for future work is the exploration of user-guided contour tracking in video.

6.3 Epilogue

The goal of SnakeToonz is not to surpass professional animation, or to trivialize the medium; the creativity of a professional artist will always be unreachable by a semi-automatic system. Instead, the contribution of SnakeToonz is to allow those without experience in cel animation to express themselves in the medium at quality levels much better than they could have accomplished alone.

We believe that SnakeToonz embodies a new and useful approach to human-computer artistic dialogues that will help those of us who are brimming with creative vision but lack the skill and experience to map from this vision into results. The key is to abstract the constraints and objectives of a medium, and then to project an inexperienced user's input into this space. This approach promises to be useful in a variety of artistic mediums beyond cartooning.

6.4 Acknowledgements

This research was conducted at Starlab, a private research lab located in Brussels, Belgium. But alas, it exists no longer. Thanks to the many wonderful people who worked there for their help, encouragement, and toys. Helpful comments on drafts of this paper were received from Benedict Brown, Bob Sumner, Joe Marks, and the anonymous reviewers. Thanks to Elke Van de Velde for photography assistance, and to David Salesin and the University of Washington graphics group for the resources used to put together the final version of this work.

References

AMINI, A. A., WEYMOUTH, T. E., AND JAIN, R. C. 1990. Using dynamic programming for solving variational problems in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-12*, 9 (Sept.), 855–867.

ANDERSON, D., FRANKEL, J. L., MARKS, J., AGARWALA, A., BEARDSLEY, P., HODGINS, J. K., LEIGH, D., RYALL, K., SULLIVAN, E., AND YEDIDIA, J. S. 2000. Tangible interaction + graphical interpretation: A new approach to 3D modeling. In *Proceedings of SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH, New York, K. Akeley, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 393–402.

ARKIN, E. M., CHEW, L. P., HUTTENLOCHER, D. P., KEDEM, K., AND MITCHELL, J. S. B. 1991. An efficiently computable metric for comparing polygonal shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-13*, 3 (Mar.), 209–216.

BERGEN, J. R., ANANDAN, P., HANNA, K. J., AND HINGORANI, R. 1992. Hierarchical model–based motion estimation. In *European Conference on Computer Vision*, Springer-Verlag, vol. 588, 237–252.

BIRCHFIELD, S., 1998. KLT: An implementation of the Kanade-Lucas-Tomasi feature tracker, http://vision.stanford.edu/~birch/klt.

BLAKE, A., AND ISARD, M. 1998. *Active Contours*. Springer-Verlag.

DE BERG, M., VAN KREVELD, M., OVERMARS, M., AND SCHWARZKOPF, O. 1999. *Computational Geometry Algorithms and Applications*. Springer-Verlag.

DE FIGUEIREDO, L. H. 1995. Adaptive sampling of parametric curves. In *Graphics Gems V*. Academic Press, Boston, 173–178.

FEKETE, J.-D., BIZOUARN, É., COURNARIE, É., GALAS, T., AND TAILLEFER, F. 1995. TicTacToon: A paperless system for professional 2D animation. In *Proceedings of SIGGRAPH 1995*, ACM Press / ACM SIGGRAPH, New York, R. Cook, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 79–90.

FOWLER, B., AND BARTELS, R. 1993. Constraint-based curve manipulation. *IEEE Computer Graphics and Applications 13*, 5 (Sept.), 43–49.

GANGNET, M., HERVE, J.-C., PUDET, T., AND THONG, J.-M. V. 1989. Incremental computation of planar maps. *Computer Graphics (Proceedings of ACM SIGGRAPH 89) 23*, 3 (July), 345–354.

GANGNET, M., THONG, J. V., AND FEKETE, J. 1994. Automated gap closing for freehand drawing. In *Technical Sketch, SIGGRAPH 94*, ACM.

HERTZMANN, A. 1998. Painterly rendering with curved brush strokes of multiple sizes. In *Proceedings of SIGGRAPH 1998*, ACM Press / ACM SIGGRAPH, New York, M. Cohen, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 453–460.

HOCH, M., AND LITWINOWICZ, P. C. 1996. A semi-automatic system for edge tracking with snakes. *The Visual Computer 12*, 2, 75–83.

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: A sketching interface for 3D freeform design. In *Proceedings of SIGGRAPH 1999*, ACM Press / ACM SIGGRAPH, New York, A. Rockwood, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 409–416.

KASS, M., WITKIN, A., AND TERZOPOULOS, D. 1987. Snakes: Active contour models. *International Journal of Computer Vision 1*, 4, 321–331.

KOWALSKI, M. A., MARKOSIAN, L., NORTHRUP, J. D., BOURDEV, L., BARZEL, R., HOLDEN, L. S., AND HUGHES, J. 1999. Art-based rendering of fur, grass, and trees. In *Proceedings of SIGGRAPH 1999*, ACM Press / ACM SIGGRAPH, New York, A. Rockwood, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 433–438.

LIEN, S.-L., SHANTZ, M., AND PRATT, V. 1987. Adaptive forward differencing for rendering curves and surfaces. *Computer Graphics (Proceedings of ACM SIGGRAPH 87) 21*, 4 (July), 111–118.

LITWINOWICZ, P. 1997. Processing images and video for an impressionist effect. In *Proceedings of SIGGRAPH 1997*, ACM Press / ACM SIGGRAPH, New York, T. Whitted, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 407–414.

MALTIN, L. 1980. *Of Mice and Magic: A History of American Animated Cartoons*. McGraw-Hill Book Company.

MENET, S., SAINT-MARC, P., AND MEDIONI, G. 1990. B-snakes: implementation and application to stereo. In *Proceedings DARPA Image Understanding Workshop*, DARPA. Pittsburgh.

MORTENSEN, E. N., AND BARRETT, W. A. 1995. Intelligent scissors for image composition. In *Proceedings of SIGGRAPH 1995*, ACM Press / ACM SIGGRAPH, New York, R. Cook, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 191–198.

OSTROMOUKHOV, V. 1999. Digital facial engraving. In *Proceedings of SIGGRAPH 1999*, ACM Press / ACM SIGGRAPH, New York, A. Rockwood, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 417–424.

PAUL, F. 2000. Flashing the web. *Technology Review* (Mar.).

PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., AND FLANNERY, B. P. 1992. *Numerical Recipes in C, 2nd. edition*. Cambridge University Press.

SALISBURY, M. P., ANDERSON, S. E., BARZEL, R., AND SALESIN, D. H. 1994. Interactive pen–and–ink illustration. In *Proceedings of SIGGRAPH 1994*, ACM Press / ACM SIGGRAPH, New York, A. Glassner, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 101–108.

SALISBURY, M. P., WONG, M. T., HUGHES, J. F., AND SALESIN, D. H. 1997. Orientable textures for image-based pen-and-ink illustration. In *Proceedings of SIGGRAPH 1997*, ACM Press / ACM SIGGRAPH, New York, T. Whitted, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 401–406.

SALOMON, D. 1999. *Computer Graphics and Geometric Modeling*. Springer-Verlag.

SCHNEIDER, P. J. 1990. An algorithm for automatically fitting digitized curves. In *Graphics Gems*. Academic Press, Boston, 612–626, 797–807.

SCHNEIDER, P. J. 1990. Solving the nearest-point-on-curve problem. In *Graphics Gems*. Academic Press, Boston, 607–611, 787–796.

SHI, J., AND TOMASI, C. 1993. Good features to track. Technical Report TR93-1399, Cornell University, Computer Science Department, Nov.

SHI, J., AND TOMASI, C. 1994. Good features to track. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society Press, IEEE, 593–600.

SMITH, S., AND BRADY, J. 1997. Susan: A new approach to low-level image-processing. *Internation Journal of Computer Vision 23*, 1 (May), 45–78.

THOMAS, B. 1976. *Walt Disney: An American Original*. Fireside Books (Simon and Schuster).

WHITE, T. 1988. *The Animator's Workbook*. Watson-Guptill Publications.

WILLIAMS, D. J., AND MUBARAK, S. 1992. A fast algorithm for active contours and curvature estimation. *Computer Vision, Graphics, and Image Processing. Image Understanding 55*, 1 (Jan.), 14–26.

WOLFSON, H. J. 1990. On curve matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence 12*, 5 (May), 483–489.

Figure 3: Several frames from four examples of animation created with SnakeToonz along with their corresponding video frames.