Intelligent Interactive Image Outlining using Spline Snakes

Myron Flickner Harpreet Sawhney Duaine Pryor Jeff Lotspiech

IBM Almaden Research Center,
650 Harry Road, San Jose, CA 95120-6099,
flick@almaden.ibm.com

Abstract

Outlining 2D shapes and image regions is often required in image manipulation and analysis. Most polygon and curve drawing tools in graphics do not use the knowledge of image intensities and its derivatives in guiding the outlining process, and hence are quite tedious and error prone. This paper presents a fast interactive drawing tool based on a parametric spline formulation of snakes that exploits the underlying image intensities. Specifically, interactively specified contours are naturally attracted to strong image edges in the vicinity because spline snakes are designed to migrate to significant ridges in the gradient magnitude image. Furthermore, allowable stretching and bending can be controlled by minimizing explicitly the bending and stretching energies of the curve. Parameterization of spline snakes in terms of a limited number of control points yields a fast minimization solution thus enabling interactive rubber banding of the curve on the screen. This feedback allows users to quickly guide the snake on the desired path.

1 Introduction

Snakes were introduced by Kass in 1988 [1] as an energy minimizing spline guided by external and internal forces. Formally, snakes are curves that minimize an energy functional with a term for the internal energy of the curve and a term giving external forces to the curve. The internal energy consists of curve bending energy and curve stretching energy. In image applications, external forces are linked to the gradient of the image causing the snake to be drawn to image edges. In [1] snakes are numerical solutions of the Euler equations for the functional minimization problem. In this paper we present an alternative formulation of snakes that is similar to [2, 3]. In our formulation spline control points are the unknown parameter. This formulation converts the snake problem into a parameter estimation problem. We show a fast method for evaluation of the energy functional that when coupled with various search techniques allows the snake to rubber band as a mouse or pointing device is moved.

2 Spline Formulation of Snakes

In this section we present a spline formulation of snakes. It is well know that a B-spline curve is defined by it control points as

$$f(t) = \sum_{i=0}^{N} x_i N_i(t) = \boldsymbol{x}^T \boldsymbol{N}$$

where $\boldsymbol{N}$ is the vector of the B-spline basis functions with element $N_i(t)$ and x_i is the i-th B-spline control point. A simple use of a B-spline curve in an energy-minimization formulation for deformable snakes is

$$E_{total} = w_1 E_{stretch} + w_2 E_{bend} - E_{image} \quad (1)$$

$$E_{image} = \int_0^1 |\nabla I(\boldsymbol{x}^T\boldsymbol{N}, \boldsymbol{y}^T\boldsymbol{N})| dt \quad (2)$$

$$E_{stretch} = \int_0^1 (|\boldsymbol{x}^T\boldsymbol{N}'|^2 + |\boldsymbol{y}^T\boldsymbol{N}'|^2) dt \quad (3)$$

$$E_{bend} = \int_0^1 (|\boldsymbol{x}^T\boldsymbol{N}''|^2 + |\boldsymbol{y}^T\boldsymbol{N}''|^2) dt \quad (4)$$

where $\boldsymbol{x}, \boldsymbol{y}$ are the unknown B-spline control points that define the snake. Now the problem is to find the control points that minimize E_{total}.

2.1 Energy evaluation

Now let us consider evaluation of (1). It is easy to show that

$$\int_0^1 |\boldsymbol{x}^T\boldsymbol{N}'|^2 = \int_0^1 \boldsymbol{x}^T\boldsymbol{N}'\boldsymbol{N}'^T\boldsymbol{x} dt = \boldsymbol{x}^T H \boldsymbol{x}$$

where H is a matrix with elements $\int_0^1 N_i'(t)N_j'(t)dt$. We can define a similar matrix G with elements

$\int_0^1 N_i''(t)N_j''(t)dt$ allowing us to evaluate the bending and stretching terms as the two quadratic forms

$$x^T F x + y^T F y$$

where the matrix $F = w_1 H + w_2 G$. Since B-splines have compact support F is banded and can be pre-computed each time the knot sequence changes since it depends on the B-spline basis not on the control points. Furthermore, if uniform B-splines are used F is symmetric and Toeplitz and the matrix can be created by pre-computing $2(n+1)$ integrals where n is the degree of the spline used. For example, for cubic B-spline one needs to pre-compute 8 numbers to create the F matrix. If a small number of control points are used, the stretching and bending terms are not needed due to the fact that splines are naturally smooth. In our implementation we used uniform cubic B-splines and did not include the bending and stretching terms.

The major difficulty minimizing E_{total} is the E_{image} term. Conceptually, this term is a line/path integral where the path is given by the curve and the surface is given by the gradient magnitude. The objective is to integrate the gradient magnitude over the path of the curve.

It is straightforward to estimate the gradient magnitude using one of several standard digital gradient operators. We use the recursive filter formulations proposed by Deriche in [4]. This formulation allows for quick computation of a smoothed gradient. The smoothing helps since it tends to widen edges and reduce noise.

Currently, the evaluation of the E_{image} integral in closed form remains an open problem in the math literature. Most numerical methods, such as gauss quadrature, do not apply since there isn't a simple relationship between the parameter t and the underlying gradient magnitude. With no closed form solution, we resort to a brute force digital approximation. Clearly, the value of the integral can be estimated by summing the digital gradient image along the B-spline path. Using this technique, the evaluation speed of E_{image} is dominated by the the speed at which we can render a B-spline. We use a very fast B-spline rendering algorithm that generates a poly-line that can be rendered/summed with the gradient image.

3 Interactive Spline Snakes

In the past, snakes have been used to refine an existing marked area. The user approximately marks an area and then a snake is used to refine the crude approximation. The problem with this approach is that the snake refinement often results in an unexpected path. We seek to avoid this problem by providing the user with constant feedback of the current snake path. This allows the user to better control the snake and avoid unexpected results. The difficultly in this approach is that the non-linear optimization problem has to be solved at interactive speeds. Thus, this is practical only if there is a very fast, ≈ 40 milliseconds, optimization process. Our spline snake formulation meets these requirements.

When the image is loaded, the image gradient and if uniform knots are used the G, H, and F matrices are pre-computed. When the button is pressed, the coordinate is recorded as the first three control points of the curve. As the mouse is moved with the button held down, the coordinate obtained replaces the last three control points of the curve. We then find an optimal path between the press point and current point optimizing all interior control points. We do not optimize the starting and ending triple control points since we want to force the curve to pass thru these points. Finally, when the button is released, we prepare for the next button press.

4 Fast Spline Rendering

Given B-spline control points a simple matrix multiplication will generate Bezier control points (see [5] page 243). This allows use of fast Bezier curve rendering algorithms.

Bezier subdivision refines a straight line approximation to a curve by successively dividing the curve in half. This section describes the use of a convergence result to generate a tight bound upper bound on the depth to which the Bezier subdivision scheme must be carried in order to achieve a given accuracy of approximation. This bound can be used to eliminate the costly termination test. This significantly improves the performance of the subdivision schemes such as those first used by Bezier and deCasteljeau at Renault. It also allows better estimates of the error introduced by finite precision arithmetic.

The subdivision of Bezier cubic curves is well known in the computer graphics literature[6, 7]. Here, the method will be quickly described, but no explanation or proof of correctness will be given. The curve itself is described by a control polygon consisting of four points, p_i. From this polygon, two more polygons, the left and right child polygons, l_i and r_i, are constructed with the property that half of the curve lies within each new polygon and that each new polygon is the control polygon for the part of the curve which it contains. Thus the subdivision may proceed recursively. The subdivision of p_i into r_i and l_i is defined below with the values at the end of such an iteration given by

$$\begin{aligned}
l_1 &= p_1 \\
l_2 &= \frac{p_1 + p_2}{2} \\
l_3 &= \frac{p_1}{4} + \frac{p_2}{2} + \frac{p_3}{4} \\
l_4 &= \frac{p_1}{8} + \frac{3 * p_2}{8} + \frac{3 * p_3}{8} + \frac{p_4}{8} \\
r_1 &= l_4 \\
r_3 &= \frac{p_4}{4} + \frac{p_3}{2} + \frac{p_2}{4} \\
r_2 &= \frac{p_3 + p_4}{2} \\
r_4 &= p_4
\end{aligned}$$

The algorithm then continues recursively with each of the two output polygons being subdivided. For any given resolution, T, the subdivision process ends when, curve within a polygon is at most a distance T from the baseline of that polygon. (The baseline of the polygon is the line connecting p_1 to p_4) If d is the function which gives the distance from a point to a line, and $l(p,q)$ represents the line which connects p to q, we have

$$\begin{aligned}
d(p_2, l(p_1, p_4)) &\leq T \\
d(p_3, l(p_1, p_4)) &\leq T
\end{aligned}$$

To evaluate these expressions, simply use the formula for the distance form a point to the line, giving

$$\frac{|(p_2 - p_1) \bullet \widehat{(p_4 - p_1)}|}{\sqrt{(p_4 - p_1) \bullet (p_4 - p_1)}} \leq T$$

$$\frac{|(p_3 - p_4) \bullet \widehat{(p_4 - p_1)}|}{\sqrt{(p_4 - p_1) \bullet (p_4 - p_1)}} \leq T$$

where $\bullet$ is the 2-D dot product and the hat of a vector is a vector of the same length as that vector but perpendicular to it. There are two such vectors, but the expression has the same value for both because of the absolute value.

Bezier subdivision generates a binary tree wherein each node corresponds to a subdivision step and each edge descending from a node corresponds to a control polygon which is the result of the subdivision at that node. In most cases, most of the work done at each node consists of testing the ending condition. In a naive implementation, this can be as high as 90 percent.

In [8] a pre-computable bound on the maximum depth to which the Bezier subdivision tree can grow is derived. Let l be the depth to which the recursion is taken and let $(x(t), y(t))$ be the coordinate functions of the curve. Let

$$\begin{aligned}
\alpha = \max(&|x''(0)|, |x''(1)|, \\
&|y''(0)|, |y''(1)|)
\end{aligned}$$

If,

$$l \geq log_4 \left(\frac{7\sqrt{2}\alpha}{\Delta_E} \right) \qquad (5)$$

then the error in the curve will be less than Δ_E. So, given the tolerance, one can easily (in about 30 operations) compute the depth to which to take the recursion. Note that this formula is easy to compute (requiring 13 additions, 6 shifts, 3 compares and a variable number of shifts which is less than half the word size.) and needs to be computed only once, using quantities available before the subdivision starts.

The approach described here is to estimate the depth of the shortest uniform tree which will cause the ending condition to be true at each leaf. The algorithm then becomes,

compute the required depth l using (5),

execute Bezier Subdivision to a depth l without checking the ending condition,

draw the resulting lines.

In addition to efficiency benefits, this approach allows better estimates of the precision needed by the algorithm. The bound on the depth of the tree which we will compute is roughly one half of the usual worst case bound used in numerical calculations.

5 Search heuristics

Now that we have a fast method to evaluate the energy function we need heuristics to limit the search. To start with we need an initial guess. During a drawing operation a reasonable guess would be the last position of the curve along with the current point. However, this method tends to give a very non-uniform parameterization of the curve in that the segment between the last knot and the current point is the longest. We found a better approach is to approximate an arc length parameterization. This can be approximated by making the current point a triple control point ending the curve, rendering the resulting curve and placing the control points on the curve at equal distance intervals. Putting the control points on the curve also helps avoid self intersecting curves.

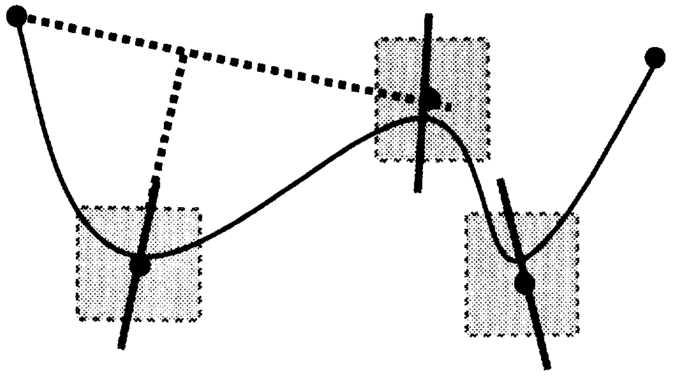

Figure 1: The window search (shaded rectangle) and tangent normal search (solid line) for the optimal curve.

5.1 Window search

A simple search strategy allows each control point to move in a rectangular window centered about an initial guess as shown in Figure 1. Since a full search of this space is very expensive computationally, each control point is optimized sequentially. Once all the control points are searched, we use the resulting local minimum. Treating each control point independently doesn't guarantee a global minimum in the window search space and resulting local minimum is a function of the order in which the control points are searched. However, it is simple to implement and fast enough interactive performance for a small number of control points.

5.2 Tangent normal search

We can improve on the window search by observing that typically outlines are drawn parallel to the edge. This suggest that we can search along a line segment perpendicular to the tangent of the curve as shown with the solid lines in Figure 1. This line segment is easy to obtain by observing that the tangent at the control point, c_i can be approximated using the neighboring control points. The line segment of length s is given by the points $(x1, y1), (x2, y2)$ which can be computed as

$$\begin{aligned} dx &= x_{i-1} - x_{i+1} \\ dy &= y_{i-1} - y_{i+1} \\ scale &= \sqrt{s^2/(dx^2 + dy^2)} \\ x1 &= x_i + scale * dy \\ y1 &= y_i - scale * dx \\ x2 &= x_i - scale * dy \\ y2 &= y_i + scale * dx \end{aligned}$$

Given the line segment $(x1, y1), (x2, y2)$ we can save function evaluation by using an intelligent search such as the golden section search [9]. Using golden search we can determine the optimal value of α between $[0, 1]$ of

$$\begin{aligned} x &= x1 - \alpha * dx \\ y &= y1 - \alpha * dy. \end{aligned}$$

The golden search reduced function evaluation by about 50% thus enabling twice as many control points to be optimized.

5.3 Conjugate Gradient

No discussion of minimizing a nonlinear function would be complete without a discussion of the conjugate gradient (CG) method. CG has the advantage of optimizing all unknowns in a single step. This improves the chance that the true minimum is found. For excellent discussions of CG see [10, 9]. Conjugate gradient requires knowledge of the derivatives of the function being optimized. We estimated $\frac{\partial E_{image}}{\partial x}$ and $\frac{\partial E_{image}}{\partial y}$ by summing $I_{xx}(x^T N, y^T N)N$ and $I_{yy}(x^T N, y^T N)N$ along the curve. I_{xx} and I_{yy} were estimated by applying a [1,0,-1] kernel to the gradient magnitude image. This approximation assumes the Hessian of the gradient magnitude is diagonal (certainly not true, and probably not a good assumption). We limit the scope of the line searches so that we will never move a control point more than a set limit, s, typically $3 < s < 20$. This keeps the snake from getting pulled to a minimum far away from the current solution. To allow for more effective bracketing of the minimum, the line segment is offset so that 2/3 of the segment is in the conjugate direction and 1/3 opposite the conjugate direction. We then apply a golden section search on the resulting conjugate line segments.

6 Results

This section describes our practical results using the various search methods. All benchmarks were done on an IBM PowerPC machine (62.6 SpecInt92, 72.2 SpecFloat92). Although most of the time is spent in the control point search, there is overhead in erasing and redrawing the curve. Within the control point search, the bulk of the time is spent rasterizing the spline rendered poly-line using Bresenham's algorithm. The tools appears very interactive if there are less than 125 function/derivative evaluations per motion event.

Using either tangent normal search or conjugate gradient we can rubber band 10 control points with a line segment length of 10 pixels. Typically, we need about 9 function evaluation for each control point to find the minimum. The tangent normal search results in a more intuitive tool since the curve doesn't dramatically change paths. Conjugate gradient can result in surprisingly large path changes, particularly when large search segments are used.

The window search allows for a 5 x 5 window to be searched with 5 control points. This results in 125 function evaluation per motion event.

Both the window method and the tangent method are simple to implement without floating point arithmetic.

References

[1] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *International Journal of Computer Vision*, pp. 321–331, 1988.

[2] S. Menet, P. Saint-Marc, and G. Medioni, "B-snakes: Implemenation and application to stereo," in *Image Understading Workshop*, pp. 720–726, Darpa, September 1990.

[3] S. Sandor and R. Leahy, "Automatic recognition of brain regions from magnetic resonance images," in *Proceedings of the 26th Asilomar Conference on Signals, Systems, and Computers*, pp. 172–177, 1992.

[4] R. Deriche, "Fast algorithms for low-level vision," *IEEE PAMI*, vol. 12, pp. 78–87, January 1990.

[5] R. Bartels, J. Beatty, and B. Barsky, *An Introduction to Splines for use in Computer Graphics and Geometric Modeling.* Los Altos, CA: Morgan Kaufmann Publishers, Inc, 1987.

[6] I. D. Faux and M. J. Pratt, *Computational Geometry for Design and Manufacture.* Wiley, 1979.

[7] T. Pavlidis, *Algorithms for Graphics and Image Processing.* Rockville,MA: Computer Science Press Inc., 1982.

[8] D. W. Pryor Jr., "A method and apparatus for fast generation of parametric curves." US Patent Number 07/705041.

[9] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, second ed., 1992.

[10] J. R. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," Tech. Rep. CMU-CS-94-125, Carnegie Mellon University, March 1994.

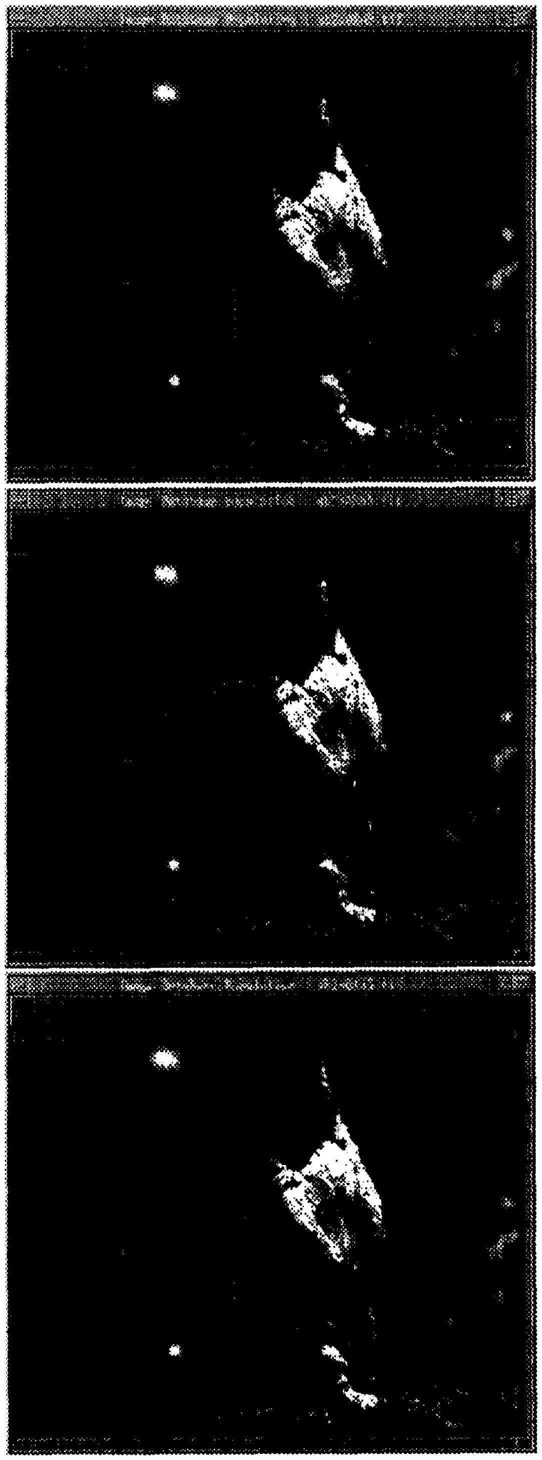

Figure 2: An interactive spline snake with 20 control points. The edge is traced from bottom to top. The top picture show an intermediate state. Then the mouse is moved away from the edge resulting in the middle picture. Finally the mouse is returned to the edge and edge following continues.