CS664 Lecture 20: Tracking

Scribed by Benjamin Cichy
bcichy@cs.cornell.edu

April 11, 2000

1 Tracking Theory and Snakes

Our discussion of tracking will center around the idea of Active Contour Models, commonly called Snakes. Snakes originate from the physics based vision movement which became popular around 1987. (Physics based vision strives to form analogies between vision algorithms and basic real-world physical phenomenon. Using these analogies we hope to gain a better intuition into how the complex parameters to our algorithms effect their behaviors)

The generic example of snakes involves selecting an object from an environment by first sketching a boundary around the object. Our snake then contracts itself until it finds the best boundary for the object. The way in which the snakes contract and find this best boundary will be the topic of this lecture.

Using the principles of physics based vision we'd like to draw some analagy between our contraction problem and the real-world. So how can we model the snakes behavior in our physical world? We can break the snakes actions into two different competing requirements. We would like to get the snake to fit as close as we can to our object, but we would also like the snake to form a smooth boundary around this object. We can view these two basic requirements as two basic forces acting upon our snake. The force which controls how "close" we get to our object can be viewed as the force of gravity on a hilly surface (we'll see later that the hills in the surface will be defined by local changes in intensities). The force acting on our snake to make it smooth can be viewed as a spring force, which would like to form straight smooth lines. The parameters to these forces are the steepness of our hills and the stiffness of our spring. We will call the force due to gravity our external force, and the force exerted by the spring to be our internal force.

These forces have the following properties in our image:

1. External Force - "Attraction to Features (edges)" Height of pixel is -$|\nabla I|$ where $|\nabla I|$ is a local measure of how fast the intensity is changing. We can see that our snake will be attracted to high $|\nabla I|$, i.e. edges.

2. Internal Force - "Stiffness" Snake is a 2d curve paramerterized by $V(s) = (X(s), Y(s))$ where $s \in [0,1]$.

We wish to minimize the (earily familiar) energy function

$$E(V(s)) = \int_0^1 E_{internal} + E_{external} dS$$

In this equation the internal force energy is more interesting:

$$E_{int} = \alpha(V_s(S))^2 + \beta(V_{ss}(S)^2 \text{ where } V_s(S) = \frac{\partial}{\partial S}V(S)$$

Extending the physics analogy, the physicists amoung us will refer to α as the membrane term and β as the thin plate term. We can use α and β to control the influence of the first and second derivatives in our energy function. To gain an intuition consider the following cases:

1. $[\alpha = \beta = 0]$ In this case our internal force is zero, so our snake will not be pulled by our spring force away from sharp edges.
2. $[\alpha > 0, \beta = 0]$ Here we bound the first derivative $V_s(S)$, and therefore V $\in C^1$ i.e. we can take the first derivative
3. $[\alpha > 0, \beta > 0]$ Now both V_s and V_{ss} are bounded, and our function is therfore continuously differentiable

2 Practice

We can't live in our continuous dream land forever, at some point we need to break our snakes up into sets of discrete managable control points. Once we do so we can think about our snakes as splines with a special set of control points.

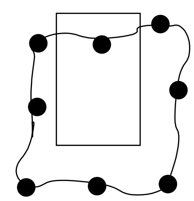

Figure 1: Spline Fit to Soda Can Object

If we define a spline

$$V = v_1, v_2, ..., v_n$$

of control points, we can then express our internal energy as

$$E_{int} = \alpha(v_i - v_{i-1})^2 + \beta(v_{i+1} - 2v_i + v_{i-1})^2$$

We wish to update our control points by minimizing the complete energy function. For simplicity we'll just consider the case where $\beta = 0$. Here we want to minimize

$$\textstyle\sum_{v \in V} E_{int}(v_i) + E_{ext}(v_i)$$

However we do not want to find the global minimum of the energy function, only the local minimum (the global minimum might end up being in a region that wasn't even encompassed by the original position of the snake). We'll solve this by limiting the distance the control points of our spline can move to plus or minus one pixel.

How can we compute the local minimum efficiently? By magical graph cuts? Nope, this time we need to resort to good ol' dynamic programming. We do so by taking advantage of the following fact:

$$E(v_1, v_2, .., v_n) = E_1(v_1, v_2) + E_2(v_2, v_3) + \ldots + E_{n-1}(v_{n-1}, v_n)$$
$$\text{where } E_i(v_i - 1, v_i) = E_{ext}(v_i) + E_{int}(v_{i-1}, v_i)$$

Note that this equation doesn't handle the boundary conditions, specifically v_{n-1} to v_1 forms a circle. For the purposes of our solution we will assume that the snake is "open" and that β=0 (only consider first derivative terms).

This fact gives us a straightforward dynamic programming solution. We can create a simple decomposition since each data point v_i only influences the total energy through E_{i-1} and E_i.

We'll use a method of dynamic programming refered to as the Viterbi Algorithm. As usual in dynamic programming solutions we will form our solutions from a table of solved subproblems. For every possibility of the i^{th} vertex the table will hold the optimal configuration of all previous verticies.

Introduce S_i, for i=2..n, where $S_i(v_i)$ = lowest total energy for $v_1, ..., v_{i-1}$ given a particular position of v_i. For example:

$$\begin{gathered}
S_2(v_2) = \text{min over } v_2 \text{ of } E_1(v_1, v_2) \\
S_3(v_3) = \text{min over } v_3 \text{ of } (S_2(v_2) + E_2(v_2, v_3) \\
S_4(v_4) = \text{min over } v_4 \text{ of } (S_2(v_3) + E_3(v_3, v_4) \\
\vdots \\
S_n(v_n) = \text{min over } v_n \text{ of } (S_{n-1}(v_{n-1}) + E_{n-1}(v_{n-1}, v_n) \\
\text{therefore total min E(all V)} = \text{min over } v_n(S_n(v_n))
\end{gathered}$$

The algorithm is thus:

1. start at v_1 and work to v_n filling in table
2. compute $\min_{v_n} S_n(v_n) ==$ min table entry at v_n
3. read backwards from end of table outputting the optimal positions of verticies