

# Curve Detection with Dynamic Programming (Triple-points based cost calculation)

## CS 355 Project

*Xuehai Zhang*  
*May 17, 2001*

### 1. Abstract

This paper presents a refined algorithm for curve detection with dynamic programming. In this advanced algorithm, the structure of the cost function is based on a triple of consecutive variables instead of pairs of consecutive indices. We give the details of the algorithm and the approach to calculate the cost by merging the prior penalty and the log-posterior. Furthermore, the paper performs the efficiency comparison between this new algorithm and the one presented in the text Chap. 4 and points out both approaches' advantages and disadvantages; finally, we give some consideration about the further work aiming to gain more benefit.

### 2. Introduction

Our research work is a type of extension of the deformable curve model described in the text's chap. 4. First, we talk a little bit about the deformable curve model. The model is defined in terms of a sequence of points  $Z = (z_1, \dots, z_n)$  on the reference grid  $G$ . The instantiation is described directly in terms of a sequence  $(\theta_1, \dots, \theta_n)$  of locations in the image grid  $L$ , and the constraints are explicitly defined in terms of a set  $\Theta \subset L^n$  and a prior  $P(\theta)$  penalizing non-smooth deformations of the model curve. The data model is again based on a conditional independence assumption. In this model, when we perform computational work, we focus on discrete aspect instead of the continuum. This is the key difference from what we do in the deformable contour model. The cost function in this model is based upon a sum of costs on certain size of consecutive points. This calculation approach will buy itself a global optimization in the end. But we may have many choices to choose the size of the consecutive points. It is clear that different sizes of the consecutive points we choose, the different advantages and disadvantage about the result of curve detection we may gain. In the text, the author describes and provides the model and the algorithm of the curve detection by using pairs of consecutive points to calculate the cost. In our work, we let the cost function based on triples of consecutive points based on the following consideration: there are no particular assumptions on the shape of the curve, in addition, smoothness occupies as the largest part of the constraint.

From the text, we conclude the structure of the cost function implies an efficient minimization by using dynamic programming, and the algorithm has provided the specified implementation about it. In our research work, we just borrowed the same dynamic programming idea, but provide a different cost function. We would like to talk something about dynamic programming approach. In a dynamic programming framework, we consider the problem as one of passing sequentially through several stages with various state possible at each stage, the goal is to find the best (in our algorithm, the best means the minimum cost) path from the first stage to the last, the “destination” stage. For each transition from one state to another in a successive stage, there is an associated “distance” or “cost”(in our algorithm, it is called “cost”). You can lean more about it by referring to the algorithm texts. It is not hard to see the work the curve detection model performs is according to the nature of dynamic programming.

### 3. The Prior and The Posterior

First we briefly review the prior and the posterior used in the cost function based on the pairs of consecutive points. The curve is parameterized directly though the locations of the n points  $\theta_1, \dots, \theta_n$ . It is important to include a prior penalizing irregular non-smooth

curve. It is something like  $P(\theta_1, \dots, \theta_n) \propto \exp\left[-\left(\sum_{i=1}^{n-1} \varphi_i(\theta_i, \theta_{i+1})\right)\right]$ . This can have a variety

of forms. In the text, the author provides one means that comparing the angles and lengths between a pair of consecutive points in the instantiation and the corresponding pair in the model sequence.

$$\Phi_i(\theta_i, \theta_{i+1}) = A | \text{ang}(\theta_{i+1} - \theta_i, z_{i+1} - z_i) | + B | \log(| \theta_{i+1} - \theta_i | / | z_{i+1} - z_i |) |$$

Putting this together with the data model, the log-posterior cost function is denote as this:

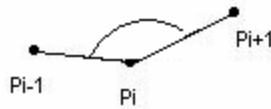
$$J(\theta) = -\log P(\theta_1, \dots, \theta_n | \hat{I}) = \sum_{i=1}^{n-1} \Phi_i(\theta_i, \theta_{i+1}) + C$$

where  $\Phi_i(\theta_i, \theta_{i+1}) = \varphi_i(\theta_i, \theta_{i+1}) - \psi_i(\hat{I}, \theta_i, \theta_{i+1}), i = 1, \dots, n-1$

Now, we consider how to formalize the cost function based on the triples of consecutive points. Similarly, we can write down the prior penalty. It might be like

$P(\theta_1, \dots, \theta_n) \propto \exp\left[-\left(\sum_{i=1}^{n-2} \varphi_i(\theta_i, \theta_{i+1}, \theta_{i+2})\right)\right]$ . Then we decide what is the proper type of

function  $\varphi_i(\theta_i, \theta_{i+1}, \theta_{i+2})$  here. We illustrate a triple of consecutive points in the following figure:



Here we also have many means to denote the function  $\varphi_i(\theta_i, \theta_{i+1}, \theta_{i+2})$ , for example, in Chap. 7 of the text, the author gives one approach of soft constraints which has the form

$$\sum_{j=1}^3 ((\alpha_{i_j} - a_{i_j}) \bmod 2\pi)^2$$

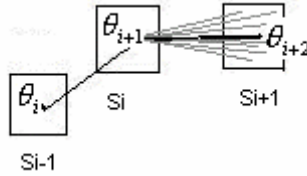
where  $\alpha_{i_j}$  is the angle at vertex  $\theta_{i_j}$  and  $a_{i_j}$  is the angle at vertex  $z_{i_j}$ . In our model, we use an even simpler one but basically in same purpose. We let  $\varphi_i(\theta_i, \theta_{i+1}, \theta_{i+2})$  be the modulus of the difference between the angle of the segment  $\theta_i, \theta_{i+1}$  and the segment  $\theta_{i+1}, \theta_{i+2}$ , that is  $\varphi_i(\theta_i, \theta_{i+1}, \theta_{i+2}) = |\text{ang}(\theta_i, \theta_{i+1}) - \text{ang}(\theta_{i+1}, \theta_{i+2})|$ . We assume that the smaller the difference is, the less cost the function is. In other words, if the segment  $\theta_i, \theta_{i+1}$  and the segment  $\theta_{i+1}, \theta_{i+2}$  are nearly in the same line, the minimum cost the function will gain. We also expect the cost function is in the format like this:

$$J(\theta) = -\log P(\theta_1, \dots, \theta_n | \hat{I}) = \sum_{i=2}^{n-2} \Phi_i(\theta_i, \theta_{i+1}, \theta_{i+2}) + f^*(\theta_i, \theta_{i+1})$$

where  $\Phi_i(\theta_i, \theta_{i+1}, \theta_{i+2}) = \varphi_i(\theta_i, \theta_{i+1}, \theta_{i+2}) - \psi_i(\hat{I}, \theta_i, \theta_{i+1}) - \psi_i(\hat{I}, \theta_{i+1}, \theta_{i+2})$

#### 4. Dynamic Programming Algorithm

We still borrow the state space idea from the chap. 4 of the text. Each point  $\theta_i$  along the curve is assigned a state space  $S \subset L$  of possible values, which are described in the following figure.



The key point to this algorithm is: for each  $\theta_i \in S_i$ , and each  $\theta_{i+1} \in S_{i+1}$ , we should find the best  $\theta_{i+2} \in S_{i+2}$ , which makes the cost be the minimum. Here, we view  $\theta_i, \theta_{i+1}$  as a pair of points, which means for every points pair whose head is in  $S_i$  and whose end is in  $S_{i+1}$ , we takes every  $\theta_{i+2}$  in  $S_{i+2}$  into consideration and calculate the cost, then we select the one who support the minimum cost in  $S_{i+2}$  be the optimized one.

##### 4.1 the refined algorithm

1. For  $I = 1, \dots, n$  define arrays in  $H_i$  indexed by points pair  $(x, y)$  where  $x \in S_{i-1}$  and

$y \in S_i$  and with 5 columns: the first two giving the coordinates of  $x$  are merely for convenience. The last three columns get updated as the algorithm proceeds. A tricky point needed to mention here is although in theory  $H_i$  is indexed by points pair  $(x, y)$  which is a 2-dimensional stuff, in reality, we only use a one-dimensional variable  $k$  as the index of  $H$ . So, there exists a translation between the 2-dimensional index  $(x, y)$  and one-dimensional index  $k$ . If we denote *maximum* as the size of the state space and we construct the one-dimensional index  $k$  by row sequence first, we will get the following translation formulas:

$$k = s * \text{maximum} + t \quad \text{or} \quad s = k / \text{maximum} \\ t = k \bmod \text{maximum}$$

2. Set  $H_n(k, 5) = 0$  for all pairs of  $(l, s)$  where  $l \in S_{n-2}$  and  $s \in S_{n-1}$ . ( we may get  $k$  using the translation formula in 1) Set  $i = n - 1$ .

3. While  $i \geq 1$ , do  
for every segment  $(l, s)$  where  $l \in S_{i-1}$  and  $s \in S_i$ , find the point  $t \in S_{i+1}$ , which minimizes  $\Phi_i(l, s, t) + H_{i+1}(s * \text{maximum} + t, 5)$ . Store the coordinates of  $t$  in column3 and column4 of the  $k = (s * \text{maximum} + t)$  entry of  $H_i$ , column5 is used to store the value  $f^*(l, s)$

4. Loop over the array  $H_2$  and find the entry with lowest value of column 5, let's assume the entry is  $k = (l * \text{maximum} + s)$ , here  $l \in S_1$  and  $s \in S_2$ . Set  $i = 2$

5. While  $i \leq n - 1$ , do  
Set  $t$  to be the point given in column 3 and 4 of  $H_i$  at the row indexed by  $(l, s)$ ,  
let  $l = s, s = t$ ,  
set  $i = i + 1$ ,

6. We will get the final optimized curve

#### **4.2 Code modification specification**

We implement the refined curve detection algorithm with dynamic programming based on the original source code, `curve_dyna.C`. Although we keep nearly the whole structure of the code, we made a lot of modification as to calculate the cost and redraw the curve at each stage.

A lot of codes are modified in function `void dynamic(Pixels &cf, int bsize, image &im, int dis, detection &det)`. We made it support the 3<sup>rd</sup> point by importing another loop.

We also divide the cost calculation function *void Psi(Pixel p0, Pixel p1, Pixel m0, Pixel m1, double lp, double lpb, double ap, double apb, dgrid& vals)* into two part and we use function *double CalPriPenalty(Pixel p0, Pixel p1, Pixel p2, Pixel m0, Pixel m1, Pixel m2)* as a separate, and new one to calculate the prior penalty. For we only need to do one time of prior penalty adding action while performing two times of *PSI* tasks.

Furthermore, we modified a lot at the function *void update\_graphics(viewport &vp, Pixelss& S, igrd &I, dgrids &H, int k, int maxnum)* to redraw curve at each stage.

## 5. Comparison and Analysis

As we can estimate, it is not necessary to try all possible  $n$ -tuples of points from  $S_i, i = 1, \dots, n$ . Rather the computation reduces to trying all possible pairs  $(i, i+1)$  or triples  $(i, i+1, i+2)$  of consecutive indices. Even with this reduction it is impractical to assume that  $S_i, i = 1, \dots, n$  is the entire image lattice. Here we only make a rough comparison of pairs- and triples- cases.

- Through the refined algorithm we describe above, we can conclude the triples-case works better than pairs-case when there is no particular prior shape to the curve and the only constraint is that of smoothness. The reason is from the way we calculate the prior penalty. Given 3 points a, b, c, we focus on the angle difference between two segments (a, b), (b, c). If the angle is little less than  $\pi$ , in other words, the three points are almost located on one line, we would like to say they are more likely to be selected in this stage to be in the curve. In the other hand, the pairs-case does not take the angle difference into consideration; it might lack efficiency while handling smooth curve.
- Things might change if we use these two approaches to handle the detection of those very irregular curves. What we mean is, those curves have lots of, continuous sharp angles along them. At this circumstance, the triples-case could not buy us any benefit comparing to pairs-case.
- The running cost (time & memory) is very high while implementing the triples-case because basically it is  $O(n^3)$  while the pairs-case is  $O(n^2)$ .

From the above analysis, we see each approach has advantages and disadvantages if constrained to certain situation, so we could not tell whether or not one is definitely better than the other in general case.

## 6. Conclusion and Future Work

In this paper, we describe in details the refined curve detection approach based on dynamic programming and triple-points cost calculation. We also give the modified algorithm model. From our analysis, we deem the triples-points based cost calculation model is a good alternative to pairs-points based cost calculation model but we are reluctant to say it is hardly a real improvement. Both of these two algorithms have so many constraints and could not be used efficiently in all cases.

As to the future work, we would like to import the parallel computing technologies in to these algorithms, and try to figure out whether they could lend us the benefit to reduce certain memory or running time cost.

## 7. Reference

*Yali Amit*, 2D Object Detection and Recognition: Models, Algorithms and Networks, January 9, 2001

## Appendix

- **Source Code – curve\_dyna.C**
- **Running Result on fig-4.2.par**