MVA '92 IAPR Workshop on Machine Vision Applications Dec. 7-9,1992, Tokyo

Primitive Extraction Using Incremental Curve Generation

Gerhard Roth

Laboratory for Intelligent Systems
National Research Council of Canada
Ottawa, Canada K1A OR6
roth@iit.nrc.ca

Abstract

This paper discusses the problem of extracting curves, such as lines, circles and ellipses, from 2D edge data. Our proposed extraction algorithm operates by taking random samples of minimal subsets, and then matching the curve through each minimal subset against the edge data. We perform the time-critical curve matching step using a new approach based on incremental curve generation algorithms. These algorithms have been used previously in the computer-graphics field, but never, to our knowledge, for this purpose. This simple idea has the potential to make real-time curve extraction possible using very inexpensive hardware. It has a number of advantages over the Hough transform, which is the most common method of extracting curves.

1 Introduction

Primitive extraction is the process of finding geometric primitives in sensor data. A geometric primitive is a curve or surface which has an associated equation, such as a line, circle, plane, etc. Extraction of geometric primitives is an important task in model-based vision. Recently a number of new algorithms have been proposed to perform this task [1, 2]. The most computationally expensive step in these approaches matches a potential geometric primitive against the sensor data. This matching process returns the number of points within a small template centered on the primitive. This process is repeated a number of times, and the primitive with the most points is returned.

In this paper we describe a new way of performing this matching when the primitives are 2D curves, such as lines, circles and ellipses, and the sensor data consists of points on a 2D array. Such data are produced by the standard processing of an intensity image using edge detection, followed by thresholding to produce a binary edge map [3]. Our idea is to match potential curves against the 2D edge data by using an incremental curve generation algorithm. Such algorithms are widely used for drawing 2D curves on the screens of graphic displays [4]. More precisely, the input to our matching routine is the equation of the curve, and an array containing the 2D edge points. The curve generation routines then produces the addresses of each point on the curve. This is done by moving along the curve from point to point in a very efficient manner. Instead of drawing the curve, we propose to simply count the number of edge points that are also curve points. This count represents how well the 2D curve matches the 2D edge data.

The time taken to accomplish the matching in this way depends only on the number of points on the 2D curve, which is independent of the total number of edge points. This means that the matching time will not increase with an increase in the number of edge points. The current VLSI implementations of these incremental curve generation algorithms which are used for drawing curves, can also be used for matching curves [4]. Such VLSI implementations could match in the order of thousands of curves per second. At this rate real time extraction of 2D curves would be possible using very inexpensive hardware.

2 Hough Transform

The most widely used approach to extracting curves is the Hough transform (HT) [5]. To make a reasonable comparison of our method to the HT some understanding of how it operates is necessary. The basic principle of operation is that each edge point votes for all parameter combinations that could have produced it [3]. This voting process requires that the parameter space be partitioned into cells (usually rectangular) by quantizing each of the dimensions of this space. Then each edge point adds a single vote to all the cells whose combination of parameters could have produced a geometric primitive through that point.

An individual cell in parameter space thus describes the edge points covered by all the geometric primitives whose parameter vectors are contained in the cell. These points taken together define a template of the same approximate shape as the curve, whose size is determined by the cell size. When this template is applied to the data it returns the same set of points that voted for this cell in the standard HT algorithm. This shows that the HT is simply a time-efficient, but space-inefficient way to do template matching [6]. Assume that there are N edge points. Then for each of these points the voting process must mark the appropriate cells in parameter space. Thus, the execution time of the HT is clearly proportional to N, the number of edge points.

The relationship of the HT to template matching makes the limitations of the approach clear. The first problem is that the template produced by a cell depends on the parameterization of the geometric primitive, on the cell shape and on the cell size. It was observed that different parameterizations change the template shape. Any rectangular cell produces some distortion in the template shape relative to the ideal situation, which is to reproduce exactly the same shape as the geometric primitive. When extracting lines this distortion was lessened by using the (ρ, θ) parameterization instead of the usual slope-intercept parameterization

[7]. The second problem is that the size of parameter space is exponential in the degrees of freedom of the primitive. This means in practice, the HT is used for curves with only two-degrees of freedom. The HT is commonly used to extract lines, and how to apply it to more complex curves such as circles and ellipses, is still an open question [8].

The exponential storage requirements and distortions in template shape are problems that are unavoidable when using the HT. The relation between the HT and template matching makes it clear that these limitations are intrinsic and cannot be overcome. The use of global memory by the HT also makes it difficult to run efficiently on parallel hardware [9]. We claim that primitive extraction based on incremental curve generation avoids these difficulties. It can be used directly for extracting lines, along with more complex curves such as circles and ellipses. It can also be easily parallelized using inexpensive VLSI drawing hardware.

3 Random Sampling for Extraction

Our extraction algorithm is based on the idea of random sampling. This idea was introduced in the computer vision field by the RANSAC method [10], and was also independently discovered in the robust statistics field [11, 12]. The principle of this algorithm is that often small set of points on a curve is a good representation of the entire curve. This is trivially true for perfectly accurate data, and is less true as the accuracy decreases. Thus, for this method to be applicable the data should be reasonably accurate. By reasonable, we mean that the average error in the 2D edge data should not be more than a few picture elements.

A minimal subset is the smallest set of points necessary to define a unique instance of a curve. For a line a minimal subset has two points, since one point underconstrains a line, and three points over-constrains it. Similarly, the size of a minimal subset for a circle is three points, and for an ellipse is five points. It is clear that a curve passes through the points in a minimal subset exactly, with no error of fit.

In a key paper on the use of the HT for line extraction the suggestion was made that an alternative approach was to exhaustively match all possible lines through two edge points [7]. This was quickly discarded as being impractical, because of the large number of potential matches. If there are N edge points, and a minimal subset has R points, then there are $\binom{N}{R}$ possible minimal subsets. However, what was overlooked was that for accurate data many of these minimal subsets define the same curve, and are thus redundant. Thus if enough minimal subsets are chosen at random, then in many circumstances far fewer than $\binom{N}{R}$ will be necessary to have all the minimal subset points on a single curve.

The pseudo-code for the random sampling algorithm based on minimal subsets is below. The input is the curve definition, the 2D edge points, the number of points in a minimal subset (R), and the total number of minimal subsets (K).

For K randomly chosen sets of R points

1. *Find the parameter vector of the curve through the minimal subset points.*
2. *Match this curve against the edge points.*
3. *Save the curve that matches the most edge points.*

After completion of the algorithm the curve containing the most edge points is returned. By simply removing these matched points, and repeating the process with the remaining edge points, all the curves can be found. As we have stated, the reason for the effectiveness of this process is that for many cases K, the number of minimal subsets necessary for successful extractions is far less than the maximum possible value.

The value of K that is used depends on Y, the minimum expected number of points on a single curve. Let ϵ be the probability that a single randomly drawn point out of the N edge points is one of the Y points on the desired curve. The value of ϵ is then equal to Y/N. The probability of all of R randomly drawn points of a single minimal subset being on the curve is therefore ϵ^R. Let s be the probability that at least one of the K minimal subsets has all its R points on this curve. Then s as a function of ϵ, R, and K is:

$$s = 1 - (1 - \epsilon^R)^K \tag{1}$$

This formula is a simple application of combinatorial analysis and the same result has been presented elsewhere [11, 10]. The value of K as a function ϵ, s, and R is:

$$K = \frac{\ln(1-s)}{\ln(1-\epsilon^R)} \tag{2}$$

If we wish to have a high confidence of successful extraction then s is set to a large value (usually .95). The above equation is then used to set the value of K accordingly. However, this is the worst case value for K; the expected value, which is more realistic, can be found by setting s equal to .5.

Essentially ϵ, and therefore K, is an implicit estimate of the maximum number of possible curves of this type in the edge data, which is $1/\epsilon$. For example, if ϵ takes on the values of .5, .2 and .1, then this assumes that there are at most two, five and ten curves present. Of course there may be fewer than this number of curves, since this estimate is an upper bound. What is surprising is that for many values of ϵ the value of K is not excessive. This is demonstrated in Table 1 which lists the required K to reach 95% and 50% confidence of success ($s = .95$ and $s = .5$) for the cases where ϵ is .5, .2 and .1. For example, if there are ten lines present (ϵ is .1) then on the average seventy minimal subsets are sufficient for successful extraction. That so few minimal subsets are required in many situations is surprising. The explanation is that the value of K exhibits a definite threshold effect, increasing rapidly to the maximum value only for small values of ϵ.

However, K is an exponential function of R, the size of a minimal subset. When K is two, three and five, the associated curves are lines, circles and ellipses. From Table 1 we see that for each increase in R, there is a significant increase in K. Thus it is not practical to use this method when there are ten ellipses, but it is practical for five ellipses, ten lines, or ten circles. For complex curves such as circles and ellipses there are often constraints that make it unnecessary to actually match each of these K curves against the edge points. For example, in circle extraction the radius of the circle to be found often has an upper bound. Then if the circle defined by a minimal subset does not meet this constraint, it is not a potential solution, and need not be matched against the edge data. This increases the number of curves defined by minimal subsets that can be evaluated in a given time, since

Fraction Points ϵ	*Minimal Subset* R	*Number Trials* K	
		$s = .95$	$s = .50$
.5	2	11	3
.5	3	23	6
.5	5	95	23
.2	2	74	18
.2	3	374	87
.2	5	9361	2167
.1	2	299	70
.1	3	2995	694
.1	5	299573	69315

Table 1: The number of trials K for 95% and 50% confidence for various sizes of minimal subsets and various fractions of the total points on a single curve.

the most expensive computational step in the algorithm is matching these curves. Therefore for complex curves extraction is still practical even for fairly large values of K.

4 Incremental Curve Generation

The main drawback of this method of primitive extraction is the execution time, which is often excessive. The reason is not that the required K is excessive, since the previous section shows that K is often reasonably sized. The problem is the amount of time taken to match each curve against the edge points. This matching process returns the number of points within a small template centered on the curve. The more points there are then the more likely that this curve is valid. For curve extraction from 2D edge data this distance is set in the range of one to four pixels.

The obvious way to match a curve is to compute the distance of each of the N edge points from the curve, and then to count the number of points that are close enough. However, N is often in the order of a few thousand edge points, which makes the matching time excessive. While this time can be decreased using parallel hardware, we would like a simpler way of achieving this goal. We do this by using incremental curve generation routines to compute the list of points on the curve. Such routines generate the curve points incrementally, by moving from one point on the curve to another, using only simple integer operations. They are widely used for drawing lines, circles, parabolas and ellipses on graphics screens [13, 14]. Two examples of their output are shown in Figure 1.

Our idea is to perform the template matching by using

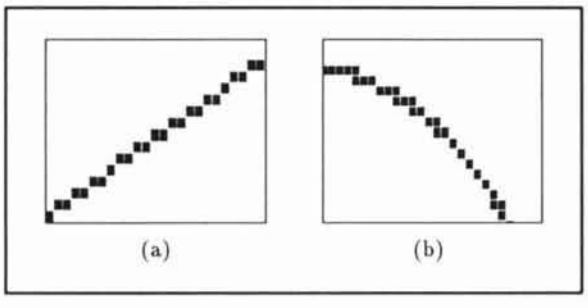

Figure 1: Results of incremental curve generation (a) Picture elements for a line (c) Picture elements for an arc of a circle.

such incremental curve generation routines. Instead of drawing the points on the curve we simply count the number of edge points that are also curve points. This produces the same result as counting the number of edge points within a given distance of the curve. However, the time taken for this new approach is proportional to the number of points on the generated curve. This makes the matching time independent of N, the number of edge points. By contrast, the direct form of curve matching has a running time proportional to N. Since, as we stated above, N is often in the order of thousands, while the number of curve points is in the order of hundreds, the difference is significant. Our experience is that with this type of curve matching the execution time of the random sampling extraction algorithm is decreased by an order of magnitude.

This type of curve matching can be used directly for the extraction of lines, circles, and ellipses since there exist efficient incremental drawing routines for these curves, [14]. In principle the same method can be used to match any curve defined by an implicit equation [15]. On ordinary serial processors in the order of a hundreds of curves a second can be matched. However, when implemented in inexpensive VLSI hardware [4], a rate of several thousand matches a second is feasible. If such VLSI hardware is available then it should be possible to extract curves such as lines, and circles, in real-time.

5 Experimental Results

In this section we will show some experimental results for line and circle extraction. The images were chosen to be typical curve extraction examples that have also been used in other papers [2]. The first example demonstrates line extraction. The original edge points are shown in parts (a) and (c) of Figure 2. Part (a) shows the edge points for a garbage can, while part (c) shows the edge points for a number of ceiling tiles. The extracted lines are shown in parts (b) and (d). In both cases the extracted curves are drawn in dark black, and are superimposed on the edge points. The execution time for part (a) was 3 seconds, and for part (c) was 4 seconds, and both examples ran on a Macintosh IIx.

The second example will demonstrate circle extraction. The original edge points are shown in parts (a) and (c) of Figure 3. Part (a) shows the edge points for a number of rings, while part (c) shows the edge points for a number of coins. The extracted circles are shown in parts (b) and (d). Again the extracted curves are drawn in dark black, and are superimposed on the edge points. The execution time for part (a) was 3 seconds, and for part (c) was 15 seconds, and both examples ran on a Macintosh IIx.

6 Discussions and Conclusion

We have described a method of curve extraction in 2D edge data that operates by creating curves through random samples of minimal subsets, and then matching these curves against the edge data. Doing the curve matching by incremental curve generation substantially decreases the execution time of this method. It makes the entire extraction process independent of N, the total number of edge points, and dependent only on the number of points on the curve.

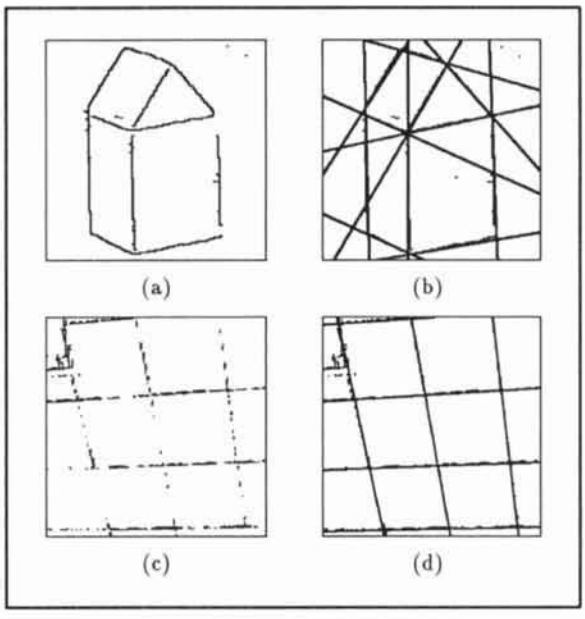

Figure 2: Extracting Lines (a) Edge Points for Garbage Can (b) Extracted Lines drawn in black (c) Edge Points for Ceiling (d) Extracted Lines drawn in black

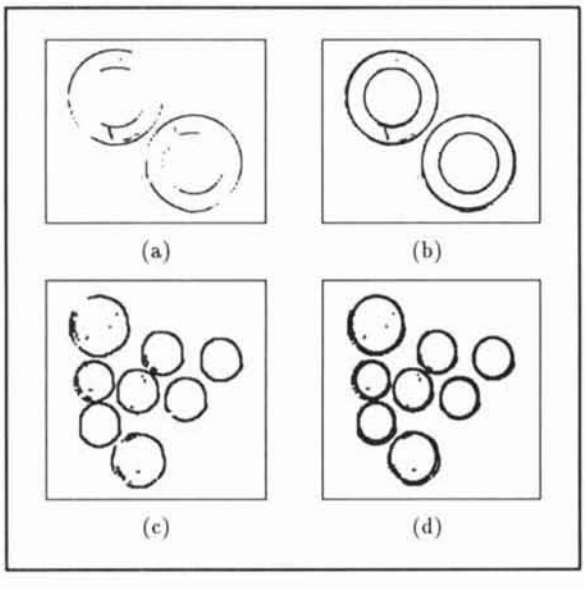

Figure 3: Extracting Circles (a) Edge Points for Rings (b) Extracted Circles draw in black (c) Edge Points for Coins (d) Extracted Circles drawn in black

This approach is able to extract a wider variety of curves than the Hough transform (HT), and has the potential to be implemented efficiently using very inexpensive VLSI hardware [4]. The HT finds all the curves, while this approach finds only one curve at a time. This means that this extraction procedure must be applied repeatedly to find all the curves. However, as long as the total number of curves is in the order of a dozen, the method is practical. We also require that the edge data be reasonably accurate. For many model-based vision tasks these two assumptions hold. Any curve that has a defining equation can potentially be matched by such incremental curve generation routines [15].

References

[1] G. Roth and M. D. Levine, "Random sampling for primitive extraction," in *International Workshop on Robust Computer Vision*, (Seattle, Washington), Oct. 1990.

[2] G. Roth and M. D. Levine, "A genetic algorithm for primitive extraction," in *Proceedings of the Fourth International Conference on Genetic Algorithms*, (San Diego), pp. 487–494, July 1991.

[3] D. Ballard and C. Brown, *Computer vision*. Prentice Hall, 1982.

[4] M. Asal, G. Short, T. Preston, R. Simpson, D. Roskell, and K. Guttag, "The texas instruments 34010 graphics system processor," *IEEE Computer Graphics and Applications*, vol. 6, pp. 24–39, Oct. 1986.

[5] J. Illingworth and J. Kittler, "A survey of the hough transform," *Computer Vision, Graphics and Image Processing*, vol. 44, pp. 87–116, 1988.

[6] G. C. Stockman and A. K. Agrawala, "Equivalence of hough transform to template matching," *Communications of the ACM*, vol. 20, pp. 820–822, 1977.

[7] R. O. Duda and P. E. Hart, "The use of the hough transform to detect lines and curves in pictures," *Communications of the ACM*, vol. 15, pp. 11–15, 1971.

[8] H. K. Yuen, J. Princen, J. Illingworth, and J. Kittler, "Compartive study of the hough transform methods for circle finding," *Image and Vision Computing*, vol. 8, pp. 71–77, Feb. 1990.

[9] A. Rosenfeld, J. O. Jr., and Y. Hung, "Hough transform algorithms for mesh-connected simd parallel processors," *Computer vision, graphics and image processing*, vol. 41, pp. 293–305, Mar. 1988.

[10] M. A. Fischler and R. C. Bolles, "Random sample consensus," *Communications of the ACM*, vol. 24, pp. 381–395, June 1981.

[11] P. J. Rousseeuw and A. M. Leroy, *Robust regression and outlier detection*. Wiley, 1987.

[12] P. Meer, D. Mintz, A. Rosenfeld, and D. Y. Kim, "Robust regression methods in computer vision: a review," *International Journal of Computer Vision*, vol. 6, no. 1, pp. 59–70, 1991.

[13] J. E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems Journal*, vol. 4, no. 1, pp. 25–30, 1965.

[14] G. Hegron, *Image Synthesis*. Cambridge, Mass.: MIT Press, 1988.

[15] R. E. Chandler, "A tracking algorithm for implictely defined curves," *IEEE Computer Graphcs and Applications*, pp. 83–89, Mar. 1988.