

For a fast implementation and to overcome the drawback of performing floating-point operations, the interpolation is computed by using integers only and by applying a recursive top-down process to the tree levels. Therefore, given the vertices of a 3-D block, the middle points of each segment, of each face, and of the block are computed sequentially.

To speed up the evaluation of the error function, a complete approximation for the block is not required. To decide on spatial splitting, only the first frame is examined, whereas, to decide on temporal splitting, only the LIs of the four pairs of corresponding vertices in the temporal direction are calculated. If there is no movement in a block, the first face can be considered representative of the whole block, so spatial splitting can be performed on the block. Analogously, if a block does not contain any high frequencies, the vertices of each face can be considered representative of the whole face, hence the temporal-splitting condition depends on these vertices only.

When the interpolator has to be calculated on the global tree structure (for the final reconstruction), some additional problems arise from the necessity of maintaining the interpolation continuity. It is worth noting that a tree structure may produce configurations in which some vertices do not match with those of the neighbouring blocks; this may cause discontinuities if such information is disregarded, and discontinuities may strongly affect the visual quality of the reconstructed image (Fig. 2).

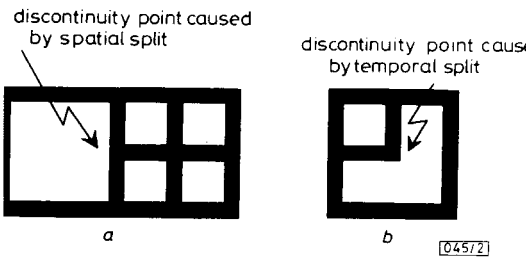


Fig. 2 Examples of possible discontinuities in grid interpolation

To face such situations, a different method for the global interpolation can be used. The algorithm implemented is of the recursive type, based on the tree structure of the grid. At each recursion, the points belonging to a lower level of the tree hierarchy are considered; the temporal interpolation is computed first (in order to solve discontinuities of the *b* type), then the spatial one (*a* type).

Starting from a block of maximum dimensions, the sequences of interpolated points observe the order in Table 1.

Table 1: Sequences of interpolated points

Approx.	For levels	Block size
<i>t</i> -level...( <i>t</i> <sub>0</sub> -1)	<i>s</i> -level...( <i>s</i> <sub>0</sub> )	(2 <sup>2s</sup> +1, 2 <sup>2s</sup> +1, 2 <sup>2s-1</sup> +1)
<i>s</i> -level...( <i>s</i> <sub>0</sub> -1)	<i>t</i> -level...( <i>t</i> <sub>0</sub> , <i>t</i> <sub>0</sub> -1)	(2 <sup>2s-1</sup> +1, 2 <sup>2s-1</sup> +1, 2 <sup>2s-1</sup> +1)
<i>t</i> -level...( <i>t</i> <sub>0</sub> -2)	<i>s</i> -level...( <i>s</i> <sub>0</sub> , <i>s</i> <sub>0</sub> -1)	(2 <sup>2s-1</sup> +1, 2 <sup>2s-1</sup> +1, 2 <sup>2s-2</sup> +1)
<i>s</i> -level...( <i>s</i> <sub>0</sub> -2)	<i>t</i> -level...( <i>t</i> <sub>0</sub> , <i>t</i> <sub>0</sub> -1, <i>t</i> <sub>0</sub> -2)	(2 <sup>2s-2</sup> +1, 2 <sup>2s-2</sup> +1, 2 <sup>2s-2</sup> +1)

As the maximum *t*-level is generally lower than the maximum *s*-level (to reduce storage requirements), the *t*-level reaches its minimum value after a few iterations. From this point on, *s*-level approximations only will be performed at each recursion.



Fig. 3 Three original frames

**Results:** The described algorithm has been developed for road-image transmission. Tests were carried out on an image sequence acquired aboard a moving vehicle and showing the road ahead

(256 × 256 pixels and 8 bit/pixel per frame). Three original frames are presented in Fig. 3. The average tree size (for 16 frames) was around 50000 nodes (25000 leaves), and about 30000 grid vertices were used for the interpolation. The tree was Lempel-Ziv coded (bit rate: < 0.4 bits per node), and the vertices were coded by a DPCM-Huffman encoder (bit rate: 2.3 bits per sample). The final bit rate was around 0.1 bit/pixel, and the visual quality shown in Fig. 4 was achieved.



Fig. 4 Same frames after reconstruction (bit rate: 0.16 bit/pixel)

**Acknowledgment:** This work was carried out within the framework of the EUREKA-Prometheus-Procom project.

© IEE 1993

9 November 1993

Electronics Letters Online No: 19940066

F.G.B. de Natale (DIBE, Università di Genova, Italy)

G.S. Desoli (Ph.D Engineering, Genova, Italy)

D.D. Giusto (Istituto di Elettrotecnica, Università di Cagliari, Italy)

#### References

- 1 FARRELLE, P.M.: 'Recursive block coding for image data compression' (Springer-Verlag, New York, 1990)
- 2 DE NATALE, F.G.B., DESOLI, G.S., and GIUSTO, D.D.: 'Adaptive least-squares bilinear interpolation: A new approach to image-data compression', *Electron. Lett.*, 1993, 29, (18), pp. 1638-1640

## Fast greedy algorithm for active contours

K.-M. Lam and H. Yan

Indexing terms: Segmentation, Image processing

The greedy algorithm is a fast iterative method for energy minimisation of snakes for image contour detection. In the Letter, an even faster algorithm is proposed. The new algorithm has the same performance as the conventional greedy algorithm, but reduces the computing time by 30% on average.

**Introduction:** Snakes [1] is an active contour model for representing image contours. The basic snake model is an energy-minimising spline which can be operated under the influence of internal contour forces, image forces and external constraint forces.

A snake is represented as a parametric curve  $v(s) = (x(s), y(s))$ , where the arc length *s* is a parameter. An energy functional of the snake is defined as

$$\begin{aligned}
 E_{snake}^* &= \int_0^1 E_{snake}(v(s)) ds \\
 &= \int_0^1 E_{internal}(v(s)) + E_{image}(v(s)) \\
 &\quad + E_{constraint}(v(s)) ds \quad (1)
 \end{aligned}$$

where  $E_{internal}$  represents the internal energy of the contour due to bending or discontinuities,  $E_{image}$  refers to the image forces, and  $E_{constraint}$  is the external constraint forces. The location of the snakes corresponds to the local minima of the energy functional.

The greedy algorithm [2] allows a contour with controlled first and second order continuity to converge in an area with high image energy. The energy-functional to be minimised by this algorithm is

$$E = \int (\alpha(s)E_{continuity} + \beta(s)E_{curvature} + \gamma(s)E_{image}) ds \quad (2)$$

In this algorithm, the energy function is computed at  $v_i$  and each of its eight neighbours. The points before and after it on the contour are used in computing the continuity constraints. The continuity force  $E_{continuity} = (\bar{d} - |v_i - v_{i-1}|)$  encourages even spacing of points, where  $\bar{d}$  is the average distance between points. The curvature force  $E_{curvature} = |v_{i-1} - 2v_i + v_{i+1}|^2$  gives a reasonable and quick estimate of curvature. For a point with magnitude ( $mag$ ), the image force,  $E_{image}$ , is defined as  $(min - mag)/(max - min)$ , where ( $max$ ) and ( $min$ ) are the maximum and minimum gradient in each neighbourhood. The location having the smallest value is chosen as the new position of  $v_i$ . At the end of each iteration, the curvature at each point on the new contour is determined. If the value is larger than a threshold,  $\beta_i$  is set to zero for the next iteration.

**Fast greedy algorithm:** The greedy algorithm is a fast iteration method for the minimal energy iteration process of the active contour model. The computations required in this algorithm can be classified into two main processes:

- (i) The energy function is computed at a point and at each of its eight neighbours to determine the new position for the point.
- (ii) At the end of each iteration, the curvature at each point on the new contour is determined.

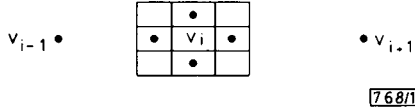


Fig. 1 Neighbours searched when pattern = 1 in fast greedy algorithm

The neighbours of the location having the smallest value of the energy function also have small values. Therefore the computation required in searching for a new position can be greatly reduced by searching the neighbours in alternate patterns shown in Figs. 1 and 2. By alternating the two search patterns, the whole space searched by the greedy algorithm is covered.

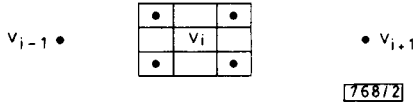


Fig. 2 Neighbours searched when pattern = -1 in fast greedy algorithm

The computation for curvature can also be reduced because the curvature energy  $E_{curvature}$  provides a good estimate of curvature at a point. If  $E_{curvature}$  is large at a point, the magnitude of curvature  $c_i = |\bar{u}_i / |\bar{u}_i| - \bar{u}_{i+1} / |\bar{u}_{i+1}||^2$  will also be large. Therefore, the curvature at a point is calculated only when its value  $E_{curvature}$  is larger than a threshold. This new greedy algorithm is described as follows:

```

j_min = j
Move point v_i to location j_min
if j_min not current location, ptmoved+ = 1
pattern = -pattern
/*process to determine where to allow corners in the
next iteration*/
for i = 0 to n - 1
  if E_curvature_i < threshold1 then
    c_i = |\bar{u}_i / |\bar{u}_i| - \bar{u}_{i+1} / |\bar{u}_{i+1}||^2
  for i = 0 to n - 1
    if (c_i > c_{i-1} and c_i > c_{i+1}
    and c_i > threshold2 and mag(v_i) > threshold3)
      then \beta_i = 0
until ptmoved < threshold3

```

Pseudo-code:

```

do
  /*loop to move points to new locations*/
  for i = 0 to n
    E_min = BIG
    if pattern = 1 then
      for j = 0 to 4
        k = jth point in Fig. 1
        E_j = \alpha_i E_{continuity,k} + \beta_i E_{curvature,k} + \gamma_i E_{image,k}
        if E_j < E_min then
          E_min = E_j
          j_min = j
      else
        for j = 0 to 4
          k = jth point in Fig. 2
          E_j = \alpha_i E_{continuity,k} + \beta_i E_{curvature,k} + \gamma_i E_{image,k}
          if E_j < E_min then
            E_min = E_j

```

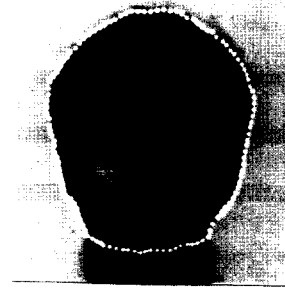


Fig. 3 Contour representation for greedy algorithm

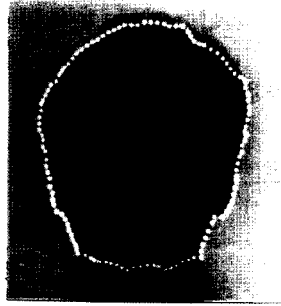
**Experimental results:** The experiments were conducted on a Sun Sparc 2 workstation. The numbers of iterations and the execution times for an image with the two algorithms are compared and tabulated in Table 1. Parameters ( $\alpha$ ,  $\beta$ ,  $\gamma$ ) are chosen to be (1.0

Table 1: Number of iterations and execution time for greedy and fast greedy algorithms

$\alpha, \beta, \gamma$		Greedy	Fast greedy
(1.0, 1.0, 1.0)	Iterations	17	20
	Time [s]	1.0730	0.7118
(1.0, 1.0, 1.2)	Iterations	12	18
	Time [s]	0.7618	0.6422
(1.0, 1.2, 1.0)	Iterations	16	18
	Time [s]	1.0105	0.6335
(1.2, 1.0, 1.0)	Iterations	17	18
	Time [s]	1.1105	0.6338

1.0, 1.0), (1.0, 1.0, 1.2), (1.0, 1.2, 1.0) and (1.2, 1.0, 1.0). The execution time required for the fast greedy algorithm is reduced by ~30% on average. The number of iterations required is larger with this fast algorithm as expected. Figs. 3 and 4 illustrate the contour representation for  $\alpha = 1.0$ ,  $\beta = 1.0$ ,  $\gamma = 0$  with the greedy and fast greedy algorithms, respectively. Their performance in contour representation is very close.

**Conclusion:** We have presented a fast algorithm for active contour modelling based on the greedy algorithm. In this method, a reduc



768/4

Fig. 4 Contour representation for fast greedy algorithm

tion in execution time of ~30% can be achieved with almost the same performance in contour representation as the greedy algorithm.

© IEE 1993

13 October 1993

Electronics Letters Online No: 19940040

K.-M. Lam and Hong Yan (Department of Electrical Engineering, Sydney University, NSW 2006, Australia)

#### References

- 1 KASS, M., WITKIN, A., and TERZOPOULOS, D.: 'Snakes: Active contour model'. Proc. First Int. Conf. on Computer Vision, 1987, (London), pp. 259-269
- 2 WILLIAMS, D.J., and SHAH, M.: 'A fast algorithm for active contours and curvature estimation', *CVGIP: Image Understanding*, 1992, **55**, (1), pp. 14-26

## Image compression using quadtree partitioned iterated function systems

G. Lu and T.L. Yew

*Indexing terms: Image processing, Data compression, Fractals*

Partitioned iterated function systems have been used to compress images, but in previous work, images have been partitioned into fixed size blocks. The authors present an image compression technique using variable block sizes based on quadtree partitioning. Experimental results show that it can achieve higher compression performance than the fixed-size partitioning technique.

**Introduction:** An iterated function system (IFS), consisting of a set of contractive affine transforms, corresponds to a unique fractal image [1]. Based on this principle, if we can find the IFS for an image, a high compression ratio can be achieved by only storing a set of parameters representing the IFS [2]. To find the IFS for an image, the image is divided into a number of non-overlapping parts which cover the entire image. Each part is a transformation of the entire image. The IFS is the collection of these transformations.

However, it is difficult to find IFSs for natural images. To solve this problem, Jacquin [3,4] proposed a block-coding scheme based on IFS. The idea is as follows. To encode an image  $f$ , the image is divided into range blocks  $R_1, R_2, \dots, R_n, \dots, R_N$ , so that  $f = R_1 \cup R_2 \cup \dots \cup R_N$ , and  $R_i \cap R_j = \emptyset$  when  $i$  is not equal to  $j$ . That is, the range blocks cover the whole image and they do not overlap. The image is also divided into larger blocks called domain blocks  $D_1, D_2, \dots, D_p, \dots, D_M$ , which can overlap. For each range block  $R_n$ , we search for a matching domain block  $D_j$  among all the domain

blocks so that the transformed  $D_j$  with a contractive transformation  $W_i$  is similar to  $R_n$ , that is  $R_i = W_i(D_j)$ . The combination of  $W_1, W_2, \dots, W_n, \dots, W_N$  is called partitioned IFS (PIFS)  $W$ . It has been proven that provided  $W$  is contractive, application of  $W$  to an arbitrary image repeatedly will result in a fixed image, and provided that  $W(f)$  is close to  $f$ , the fixed image will be close to the original image  $f$  [4]. The contractivity of  $W$  is guaranteed when domain blocks are larger than range blocks. This means that a close copy of the original image can be generated from  $W$ . By storing parameters representing  $W$ , considerable compression can be achieved. This method is similar to vector quantisation, but it has the advantage that no codebook is required during the decoding process.

In his work, Jacquin [3] used a fixed range block size of  $8 \times 8$  and domain block size of  $16 \times 16$ . For each range block, a matching domain block is found. The information required to be stored for each range block includes the position of the matching domain block and the required contractive transformation. Jacquin achieved a compression ratio of about 10 with SNR of 27.7dB.

The limitation with fixed size square range blocks is that it does not take advantage of the contents of images. In typical images, there are range blocks for which it is impossible to find matching domain blocks within a certain error tolerance when the fixed range block size is used. Similarly, there are regions where larger range blocks could be used and their matching domain blocks could still be found. This implies that higher compression performance may be achieved if variable size range blocks are used. Based on this observation, Fisher suggested a number of possible image partition techniques, including quadtree partitioning, HV-partitioning and triangular partitioning [5]. However, no detailed work has been reported using these partition techniques. One of the problems with these partitioning techniques is how to store information indicating block sizes and block locations in order to achieve high compression. In this Letter, we present our work using the quadtree partitioning technique.

**Quadtree partitioning:** In quadtree partitioning, the image is divided into four equal blocks. For each block, a matching domain block is searched. If a matching domain block is found for a range block, the range block is coded by storing information of the location of the matching domain block and the transformation used. If no matching domain block can be found for a range block within the specified error tolerance, that range block is further divided into four sub-blocks. This process repeats recursively starting from the whole image and continuing until the range blocks are small enough to be able to find matching domain blocks within some specified error tolerance. It is easier to find matching domain blocks for small range blocks because contiguous pixels in an image tend to be highly correlated. For typical images, block sizes of encoded range blocks can be  $64 \times 64$ ,  $32 \times 32$ ,  $16 \times 16$ ,  $8 \times 8$ , and  $4 \times 4$ . The block size of the matching domain block is one size larger than that of the range block. The domain block SINS can be  $128 \times 128$ ,  $64 \times 64$ ,  $16 \times 16$  and  $8 \times 8$ . In the quadtree partitioning method, because larger range blocks can be used, fewer range blocks, i.e. fewer transformations, are required to represent the image.



97071

Fig. 1 Original 'Lena' image of 8bit/pixel

However, because variable range block sizes are used in the quadtree partitioning method, the information about the block size used and the block location has to be stored together with the transformations. If this information is stored simply, three bits are required to indicate block size and 14bits are required to indicate the position of the block when the image to be coded is  $512 \times 512$ .