Fractal based progressive representation of 2D contours

Wiesław Kotarski[1], **Krzysztof Gdawiec**[2], **Grzegorz T. Machnik**[1]

[1] Institute of Computer Science, University of Silesia
kotarski@ux2.math.us.edu.pl, gmachnik@wp.pl
[2] Institute of Mathematics, University of Silesia
kgdawiec@ux2.math.us.edu.pl

Abstract. *In the paper we present a method, different from those presented in literature, for progressive representation of two dimensional contours. The method is based on fractal representation of a set of linear and quadratic curves that approximate a given contour. If one knows IFS for fractal rendering of every part of the contour, then using the set of all IFSs (the so-called PIFS) it is possible to generate that contour fractally. When starting iterations from a single points belonging to the segments of the contour in every iteration further points lying on the contour are generated. In every iteration number of points placed on the contour is doubling. So, the contour is presented progressively in higher and higher resolution showing gradually larger number of details.*

1. Introduction.

Curves are used in many applications in CAD/CAM and computer graphics. Using them, e.g. high quality fonts can be designed or 2D shapes can be modelled. Curves for such applications should have flexible representations allowing their easy editing, smoothing and scan conversion. For such applications subdivision curves, multiresolution curves and progressive ones are used. Recall, in short what we mean by the mentioned curves.

Subdivision curves [2],[3],[12],[13] are produced starting with a set of coarse points P_k from which a larger set of points P_{k+1}, using a subdivision matrix S, is obtained. This process is repeated a finite number of times to get the finer set of points. Successive application of subdivision leads to a hierarchy of curves, which converge to a smooth limit curve. Subdivision process can be described by the following formula:

$$SP_k = P_{k+1}. \qquad (1)$$

There are known many subdivision schemes e.g. Chaikin's [2], de Casteljau, Dyn-Gregory [3].

Multiresolution (MR) [4], [11] is a kind of representation which allows the user to change a high resolution into a lower one, in such a way that the original data can be reconstructed correctly. Higher resolution means that details are well recognizable whereas they are lost in lower resolution. There are known at least two main approaches to multiresolution. The first one is based on classical wavelets [12]. Another approach to construct MR is using reverse subdivision [1],[10]. This is because any subdivision scheme converts a low resolution approximation to a higher one, whereas reverse subdivision works in opposite – it converts a high resolution approximation to a lower one.

In the paper we present a method, that applies results given in [5],[6]. The method is different from those presented in literature, for progressive representation of two dimensional contours. The presented method is based on fractal representation [8],[9] of a set of linear and quadratic curves that approximate a given contour. If one knows IFS for fractal rendering of every part of the contour, then using the set of all IFSs (the so-called PIFS) it is possible to generate that contour fractally. When starting iterations from a single points belonging to the segments of the contour on every iteration, further points lying on the contour are generated. In every iteration number of points placed on the contour is doubling. So, the contour gradually is presented in higher and higher resolution with larger number of details. If additionally, one joins those points by linear segments then even on early iterations a good representation of the given contour is obtained. To perform the process of progressive representation it is needed the same small amount of information on every iteration step. That information is stored in the coefficients of IFSs easily computed using points defining every segment. The technique introduced can have various applications such as view-dependent rendering, flexible editing and progressive transmission.

2. Curves as fractals. Fractals are self-similar objects that can be obtained iteratively using the following basic transformations: translation, rotation, scaling, shears. Those transformations can be described as follows:

$$\begin{cases} x' = ax + by + e, \\ y' = cx + dy + f, \end{cases} \tag{2}$$

where $[x, y]$ and $[x', y']$ are a point and its image, respectively in $\mathbb{R}^2$, a, d are coefficients responsible for scaling, e, f for translation, and b, c for shears and rotation.

Using homogenous coordinates the above transformation can be presented in the matrix form:

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \cdot F, \tag{3}$$

where

$$F = \begin{bmatrix} a & c & 0 \\ b & d & 0 \\ e & f & 1 \end{bmatrix}. \tag{4}$$

Matrix F defines the so-called affine transformation. IFS (Iterated Function System) is a collection of affine transformations. Only IFSs consisting of contractive affine transformations (such that they decrease distance) can produce attractors i.e. fractals using deterministic or probabilistic algorithms. Usually, thinking about fractals we have in mind very complex objects and we do not consider smooth curves to be fractals. But, in reality, they are fractals. For example, following [5] it is known that quadratic Bézier curve that is defined by three control points $P_0 = [x_0, y_0]$, $P_1 = [x_1, y_1]$, $P_2 = [x_2, y_2]$ can be generated fractally using IFS of the following form:

$$IFS = \{P^{-1} \cdot L \cdot P, P^{-1} \cdot R \cdot P\}, \tag{5}$$

where

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1/2 & 0 \\ 1/4 & 1/2 & 1/4 \end{bmatrix}, R = \begin{bmatrix} 1/4 & 1/2 & 1/4 \\ 0 & 1/2 & 1/2 \\ 0 & 0 & 1 \end{bmatrix},$$
$$P = \begin{bmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{bmatrix}, \tag{6}$$

P^{-1} denotes the inverse matrix to P. P^{-1} exists, when points P_0, P_1 ,P_2 are not co-linear.

Fig.1 demonstrates fractal generation of a quadratic curve approximation using deterministic algorithm that starts from a single point or a triangle, respectively. If additionally, starting point belongs to the curve (as e.g. P_0 or P_2) then all further generated points lie on the curve and they are distributed almost uniformly along that curve. Observe, that in every iteration, when starting from a single point, the number of generated points increases twice on each iteration. So, the curve is presented in higher and higher resolution that increases gradually. Moreover, it is easily seen that when starting from a triangle or any other initial shape one obtains the same resulting quadratic curve. But, the curve is better recognizable already on early iterations when starting from a single point.

A linear segment with ends $P_0 = [x_0, y_0]$, $P_1 = [x_1, y_1]$ can also be generated fractally using IFS of the following form:

$$IFS = \{f_1, f_2\}, \tag{7}$$

where

$$f_1 = \begin{bmatrix} 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \\ x_0/2 & y_0/2 & 1 \end{bmatrix},$$
$$f_2 = \begin{bmatrix} 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \\ (x_0 + x_1)/2 & (y_0 + y_1)/2 & 1 \end{bmatrix}. \tag{8}$$

The form of IFS producing a linear segment is obvious. Every linear segment is similar to its halfs: the right one and the left one. So scaling with $1/2$ coefficients and suitable translations are needed to produce it. When starting iterations, in

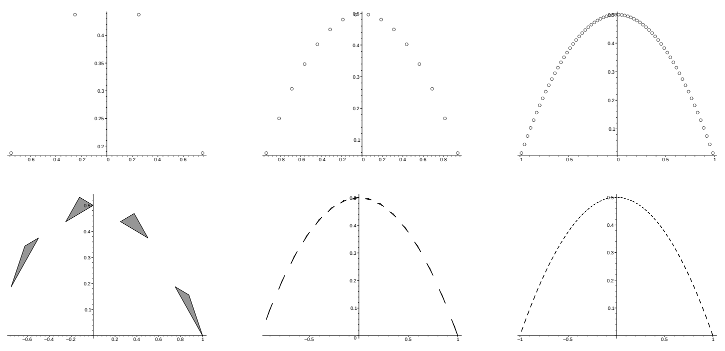

Fig. 1. Approximation of a quadratic Bézier curve generated fractally, iterations: 2, 4, 6. Top – start from a single point, bottom – start from a triangle.

deterministic algorithm, from a single point belonging to the segment (as e.g. P_0 or P_1) one obtains uniformly distributed points lying on the segment. The resulting linear segment can also be generated starting from a triangle or any other shape. But start from a single point assures that the line is well recognizable on early iterations.

Fig.2 demonstrates fractal generation of a linear segment approximation using deterministic algorithm that starts from a single point or a triangle, respectively.

So then, we conclude that quadratic curves and linear segments can be generated fractally. They can be treated as fractal primitives which can be used to built up any contour.

3. Simple contour fractally. Basing on information from previous section one can obtain fractally any contour approximation that can be modelled using segments being, e.g. quadratic Bézier or linear curves. Further, we assume that segments are joined together with C^0 continuity. However, it is possible to join them with C^1 continuity obtaining everywhere smooth contours. Every linear or a quadratic segment can be represented with the help of its IFS. The set of all such IFSs forms a PIFS giving fractal representation of the contour. To obtain a given contour in a fractal way one can use deterministic algorithm simultaneously to every IFS obtaining fractally the contour. When starting from a single point from iteration to iteration one obtains more and more points lying on the contour.

In Fig.3 an approximation of a heart generated fractally is presented. The heart is modelled using four quadratic Bézier curves. From iteration to iteration more and more points are placed on the contour. The deterministic algorithm has started from two single points that are lying on suitable parts of the contour. It is easily seen that points are almost uniformly distributed on the contour.

Of course, it is possible to perform different number of iterations for every segment. So, it is possible to generate a chosen part of the contour with smaller or larger resolution, as it is presented in Fig.4.

Here, it should be pointed out that one can change resolution easily only on segments from which the contour originally is built up. But of course, every segment can be divided into at least two parts of the same kind. Namely, linear segments can be split into linear segments and quadratic curves into parts also being quadratic curves (using, e.g. de Casteljau algorithm). So then, a user flexible can change resolution of an arbitrary chosen part of the contour. Also one can change resolution of a contour from low to high and vice versa, by simple choosing the number of iterations that should be performed by deterministic algorithm starting from a single points.

4. Complex contours fractally. Assume now, that some contour is given. For example, it can be modelled manually or extracted from an image using image processing techniques. The process of contour extraction will not be described here. So, we start our further considerations from

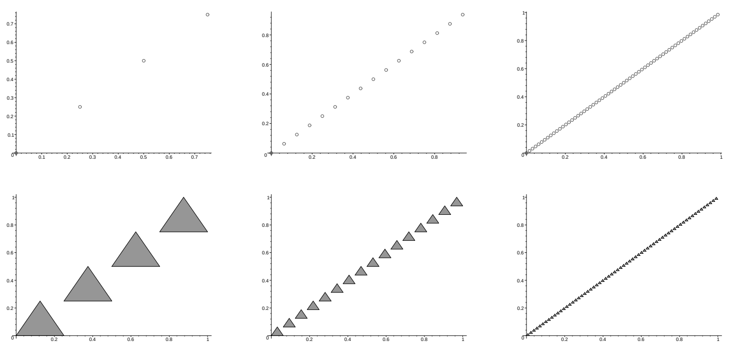

Fig. 2. Approximation of a linear segment generated fractally, iterations: 2, 4, 6. Top – start from a single point, bottom – start from a triangle.

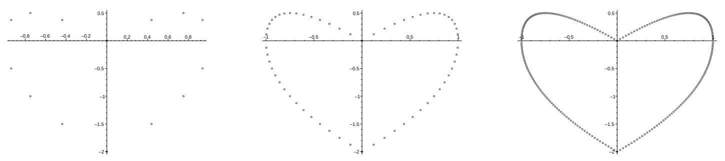

Fig. 3. Approximation of a heart generated fractally, iterations: 2, 4, 6.

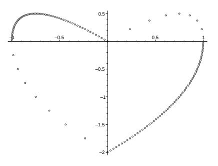

Fig. 4. Segments of the heart contour generated fractally with different resolutions.

the given contour. The contour can be split into a finite number of linear segments or quadratic curves that exactly or approximately represent it. Contour division can be done using the well-known segmentation algorithms, e.g. IPAN'99 [14] which is the one of the most effective segmentation algorithms. It finds points on the contour with the highest curvature. Further, for every segment of the given contour, we know its IFS. All those IFSs form PIFS. Similarly, as in the case of fractal rendering of the heart from previous section, one can generate fractally any contour basing on its PIFS.

In Fig.5 we present an airplane contour generated fractally with the help of 35 IFSs. Obtained points are joined by lines. So, the airplane contour is well recognizable after performing few iterations. It is easily seen that the airplane contour is generated progressively showing gradually more and more details.

5. Chaikin's subdivision and its reverse.

Consider Chaikin's subdivision applied to $n > 3$ points creating a polygonal line, as in Fig.6.

In a sequel, every side of the polygonal line is divided in the ratio $1/4 : 1/2 : 1/4$. Two new points are put and the corner is cut. In such a way from iteration to iteration the polygonal line having a greater number of sides is created and is getting more smooth – producing the limit curve p^{∞}. Subdivision can be described with the help of a bi-diagonal matrix having in its rows elements $1/4, 3/4$, or equivalently by the following recurrence:

$$\begin{cases} p_{2j}^{k+1} = \frac{3}{4}p_j^k + \frac{1}{4}p_j^{k+1}, \\ p_{2j+1}^{k+1} = \frac{1}{4}p_j^k + \frac{3}{4}p_j^{k+1}, \end{cases} \tag{9}$$

where k denotes the number of subdivision iteration and j enumerates points. From the above recurrence one can compute:

$$p_j^k = \frac{3}{2}p_{2j-1}^{k+1} - \frac{1}{2}p_{2j}^{k+1}. \tag{10}$$

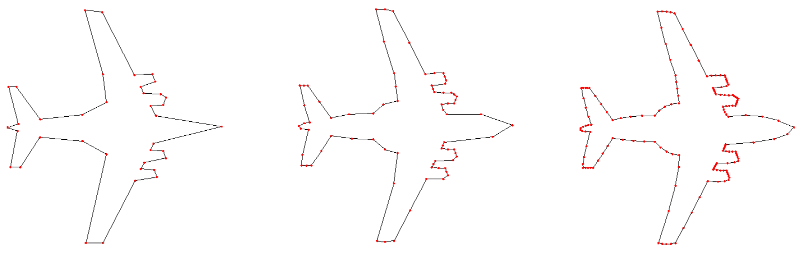

Fig. 5. Contour of an airplane generated fractally (iterations: $0, 1, 2$) with 35, 70 and 140 points, respectively joined by lines.

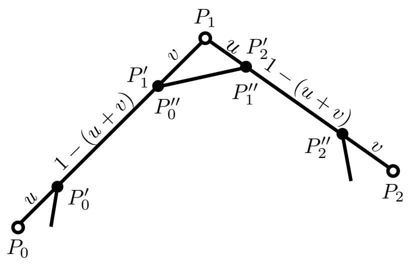

Fig. 6. Chaikin's cutting corner algorithm.

The point p_j^k can also be computed using the other pair of points:

$$p_j^k = \frac{3}{2}p_{2j-2}^{k+1} - \frac{1}{2}p_{2j-3}^{k+1}. \tag{11}$$

After computing the mean value of two above representations of p_j^k we obtain:

$$p_j^k = -\frac{1}{4}p_{2j-3}^{k+1} + \frac{3}{4}p_{2j-2}^{k+1} + \frac{3}{4}p_{2j-1}^{k+1} - \frac{1}{4}p_{2j}^{k+1}. \tag{12}$$

From the last equation it is possible to find an approximate dependency between points of lower level and higher level of subdivision. It means that it is possible to reverse Chaikin's subdivision. In Fig.7 the performance of Chaikin's subdivision is presented, whereas in Fig.8 the performance of reverse Chaikin's subdivision is showed.

In Fig.9 we present reduction of points of the airplane contour having originally 1108 points using reverse Chaikin's subdivision algorithm. The experiment has been performed with the help of the program from [7].

Comparing Fig.5 with Fig.9 it is easily seen that in both methods the number of points changes from iteration to iteration twice. But in both methods the points are distributed along the airplane contour differently. In fractal method, at early iterations, the contour is well recognizable – much better than in reverse Chaikin's subdivision with the same number of contour points. That observation has been approved by many experiments carried out by the authors. It is easy to explain that fact. Really, at the very beginning of iteration process in fractal method points placed on a contour comes from IPAN99 algorithm that extracts the most significant points of the contour – namely points with the highest curvature. Reverse Chaikin's subdivision algorithm treats all points of the contour equally. So, during points reduction process those the most significant points often can be eliminated.

6. Conclusions. In the paper we showed how to generate any contour progressively using fractal algorithm. Similar results can be obtained using Chaikin's subdivision and its reverse. From the experiments we conclude that the fractal method can be a good alternative for reverse Chaikin's subdivision. The fractal method seems to be less complicated than reverse Chaikin's subdivision. In both methods the number of points on the contour is changing from iteration to iteration twice. The results of the paper can be extended from curves to surfaces. Really, it is known following [8] how to generate fractally patches obtained via

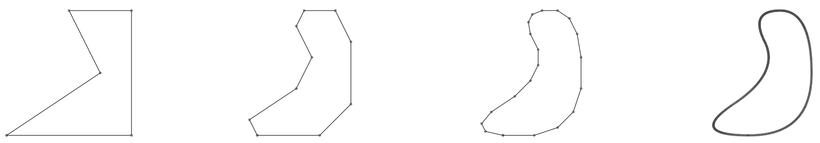

Fig. 7. Chaikin's subdivision algorithm.

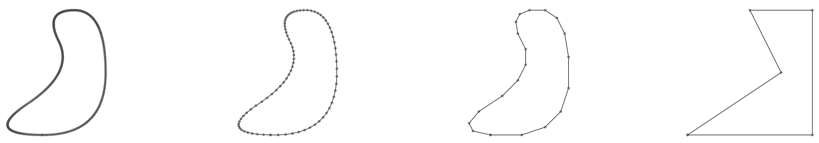

Fig. 8. Chaikin's reverse subdivision algorithm.

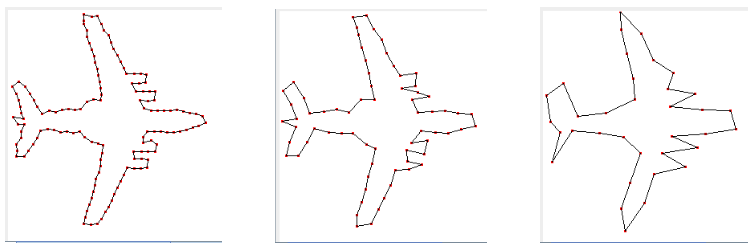

Fig. 9. Contour of the airplane obtained via reverse Chaikin's subdivision (iterations: 3, 4, 5) with 140, 70 and 35 points, respectively joint by lines.

subdivision. Also some results on reverse subdivision methods for patches are reported in literature [1],[10]. The main difficulty in extension of the fractal method from 2D to 3D is the lack of efficient segmentation algorithms for 3D shapes similar to as IPAN99 algorithm which is applicable only for 2D contours.

References.

[1] Bartels R.H., Samavati F.F.: *Reversing subdivision rules: local linear conditions and observations on inner products*. Journal of Computational and Applied Mathematic, Vol. 119, 29-67, 2000.

[2] Chaikin G.: *An algorithm for High Speed Curve Generation*. Computer Graphics Image Processing, 3, 346-349, 1974.

[3] Dyn N., Levin D., Gregory J.: *A 4-point Interpolatory Subdivision Scheme for Curve Design*. Computer Aided Geometric Design, 4, 257-268, 1987.

[4] Finkelstein A., Salesin D.H.,: *Multiresolution Curves*. Proceedings of Siggraph, 261-268, 1994.

[5] Goldman R.: *The Fractal Nature of Bézier Curves*. Proceedings of the Geometric Modeling and Processing, April 13-15, Beijing, China, 2004, pp.3-11.

[6] Gdawiec K.: *Fractal Interpolation in Modeling of 2D Contours*. International Journal of Pure and Applied Mathematics, Vol. 50, No. 3, 421-430, 2009.

[7] Holeksa S.: *Multiresolution representation of contours using reverse subdivision*. M.Sc thesis, Institute of Computer Science, University of Silesia, Sosnowiec, 2008 (in Polish).

[8] Kotarski W.: *Fractal modelling of shapes*. EXIT, Warszawa 2008, (in Polish).

[9] Kotarski W., Lisowska A.: *On Bézier-Fractal Modeling of 2D Shapes*. International Journal of Pure and Applied Mathematics, Vol. 24, No. 1, 123-134, 2005.

[10] Samavati F.F., Bartels R.H.: *Multiresolution Curve and Surface Representation by Reversing Subdivision Rules by Least-Squares Data Fitting*. Computer Graphics Forum, Vol. 18, No. 2, 97-120, 1999.

[11] Stollnitz E.J., DeRose T.D., Salesin D.H.: *Wavelets for Computer Graphics*. Morgan Kaufmann Publishers 1996.

[12] Warren J., Weimer H.: *Subdivision Methods for Geometric Design: A Constructive Approach*. Morgan Kaufmann Publishers 2002.

[13] Zorin D., Schroder P.: *Subdivision for Modeling and Animation*. ACM Computer Graphics, Course Notes 2000.

[14] http://visual.ipan.sztaki.hu/corner/cornerweb.html (IPAN99), available September 2009.