

Creating concave hull for IFS fractals using DNA-based computing

AMM Sharif Ullah*

Department of Mechanical Engineering, Kitami Institute of Technology, 165 Koen-cho, Kitami, Hokkaido 090-8507, Japan

Abstract

Fractal geometry can be used to create CAD models of complex shapes observed in the living organisms (cell, tissue, lung, blood vessels, brain structure, and alike) and in the natural world (tree, leaf, flower, landscape, coastline, cloud), as well. If one considers making a physical model of a fractal-geometry-generated CAD model, it is important to perform some topological transformations (e.g., concave/convex hull generation) for making the CAD model meaningful to the manufacturing devices. As a contribution in this area, this study describes a simple but effective procedure that can be used to generate concave hulls for fractal shapes generated by a random walk called Iterated Function System (IFS). One of the constituents of the proposed procedure is an *in silico* DNA-Based Computing. To demonstrate how the proposed concave hull generating procedure works, a case study has been performed, and using the information of the concave hull generated, a physical model of the fractal has been produced with the aid of additive manufacturing (3D printer).

Introduction

Fractal geometry [1-3] can be used to create CAD models of complex shapes observed in the living organisms (cell, tissue, lung, blood vessels, brain structure, and alike) and in the natural world (tree, leaf, flower, landscape, coastline, cloud). If one considers making a physical model of a fractal-geometry-generated CAD model, it is important to perform some topological transformations for making the CAD model meaningful to the manufacturing devices. In doing so, creating concave/convex hulls [4-16] is a must. The concept of concave/convex hull is schematically illustrated in Figure 1. The shape used in Figure 1a is a point-cloud that models the shape of a fern-leaf created by a special random walk called Iterative Function System (IFS) [17,18]. The convex hull underlying the fern-leaf is shown in Figure 1b, which is the smallest perimeter fence enclosing the point-cloud. Some concave hulls (black and red boundaries) underlying the fern-leaf are shown in Figure 1c, which are the boundary fences encompassing the point-cloud as closely as possible. In case of concave hulls, internal and external boundary fences can be considered. For the case shown in Figure 1, the outer and internal concave hulls are shown by the black and red boundary fences, as seen from Figure 1c, respectively. Compared to the convex hull, concave hulls are more effective in preserving the shape information (compare the boundary fences shown in Figure 1b and 1c).

As mentioned before, if one considers making a physical model of a fractal shape from its CAD model (in the case of IFS fractals, the CAD model takes the form of a point-cloud), it is important to create a concave/convex hull first. The reason is that the concave/convex hull helps create other data (e.g., tool-paths [12,13] and STL data [19]) necessary for creating a physical model either by using subtractive manufacturing or by using additive manufacturing [12,13,16]. In certain cases, the remodeling of the fractal-geometry-generated CAD model is necessary for the sake of physical model building process [12,13,16,20]. Therefore, creating concave/convex hulls for the fractal-geometry-generated CAD model has been an active research topic. A series of systematic transformations is needed to make the information of the CAD model meaningful for the concave/convex hull generation

procedure. Nevertheless, most of the procedures developed so far for generating concave/convex hull are computationally heavy. This article deals with this issue by proposing a new concave hull generating procedure, where the primary shape information is IFS-generated point-cloud (*i.e.*, a fractal). The focus is on generating the outer concave hull, not the inner ones. One of the important constituents of the proposed procedure is a transformation that employs an *in silico* DNA-Based Computing (DBC) [21-24]. Thus, the remainder of this article is organized as follows. Section 2 describes the DBC employed in this article. Section 3 describes the proposed DBC driven outer concave

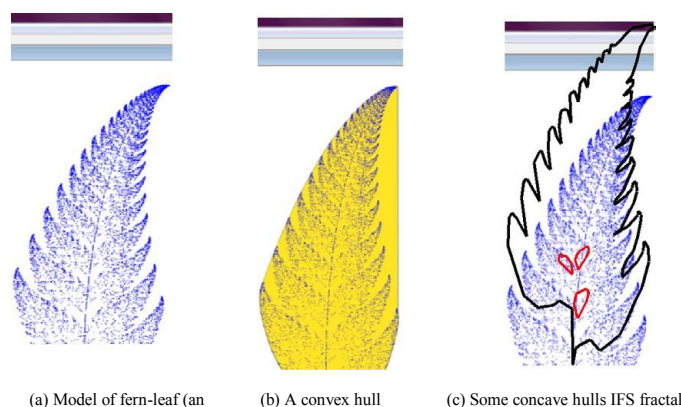


Figure 1. Concept of concave/convex hull.

Correspondence to: Sharif Ullah AMM, Department of Mechanical Engineering, Kitami Institute of Technology, 165 Koen-cho, Kitami, Hokkaido 090-8507, Japan, Tel/Fax: + 81-157-26-9207; **E-mail:** ullah@mail.kitami-it.ac.jp

Key words: fractals, concave/convex hull, DNA-based computing, additive manufacturing

Received: January 27, 2016; **Accepted:** March 03, 2016; **Published:** March 07, 2016

hull generation procedure. Section 4 describes a case study showing the effectiveness of the proposed procedure. Section 5 provides the concluding remarks of this study.

DNA-based computing (DBC)

A nature inspired computing methodology called DBC has been developed that takes the inspirations from the central dogma [25] of molecular biology. Central dogma of molecular biology simply means that once the sequential information of DNA/RNA has passed into protein (a sequence of amino acids) it cannot get out again [25]. A comprehensive description of the central dogma based DBC can be found in [21,22]. The remainder of this section briefly describes the DBC employed in this article for generating a concave hull.

Figure 2 schematically describes the form of DBC used in this article. In general, the DBC first maps a given binary array into DNA array. Finally, it maps the generated DNA array to a protein array (i.e., to a sequence of amino acids). The binary array must be a piece of information underlying the given problem (the point-cloud of the fractal shape created by a certain IFS). The protein array must help solve the given problem (in this case the concave hull creation problem).

As it is observed in Figure 2, DBC first maps the given binary array $\forall b_{ij} \in \{0,1\}$ to an DNA array, $\forall DNA_{ij} \in \{A, C, G, T\}$. In doing so, two consecutive elements ($b_{ij}b_{i+1j}$ or $b_{ij}b_{ij+1}$ = 00, 01, 10, or 11) are mapped into one of the elements taken from $\{A, C, G, T\}$.

This process underlies four different types of reading-frame: continuous/discrete row-/column-wise reading-frames. The case shown in Figure 2 corresponds to continuous column-wise reading-frame where binary array is read in the manner of $b_{ij}b_{i+1j}$, not $b_{ij}b_{ij+1}$, while creating each elements of DNA array, i.e., DNA_{ij} . Since $\forall DNA_{ij} \in \{A, C, G, T\}$, three consecutive elements of DNA array $DNA_{ij}DNA_{i+1j}DNA_{i+2j}$ or $DNA_{ij}DNA_{i+1j}DNA_{i+2j} = AAA, AAC, AAG, AAT, ACA, ACC, ACG, ACT, AGA, AGC, AGG, AGT, ATA, ATC, ATG, ATT, CAA, CAC, CAG, CAT, CCA, CCC, CCG, CCT, CGA, CGC, CGG, CGT, CTA, CTC, CTG, CTT, GAA, GAC, GAG, GAT, GCA, GCC, GCG, GCT, GGA, GGC, GGG, GGT, GTA, GTC, GTG, GTT, TAA, TAC, TAG, TAT, TCA, TCC, TCG, TCT, TGA, TGC, TGG, TGT, TTA, TTC, TTG, or TTT$. Each of these three-element combinations is called a codon and is mapped into a one-letter symbol of amino acid taken from the set of symbols $\{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, X, Y\}$ using the genetic code [21,22]. The genetic code is listed in Table 1. Note that X (Table 1) denotes one of the stop codons not an amino acid as such [21,22].

As a result, a protein array having $\forall Protein_{ij} \in \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, X, Y\}$ forms. The case shown in figure 2 corresponds to a continuous column-wise reading-frame, i.e., $DNA_{ij}DNA_{i+1j}DNA_{i+2j}$ not $DNA_{ij}DNA_{ij+1}DNA_{ij+2}$. As understood from the arbitrary case shown in Figure 2, a few-element piece of information (i.e., the binary or DNA array) transforms to a many-element piece of information (i.e., protein array) due to DBC. This characteristic of DBC has been found effective in solving pattern recognition problems of complex shapes [21-23]. In the case of creating an external concave hull, DBC can also be used. In this case, the protein must help distinguish the internal segment of a point-cloud from the external one. To describe the potentiality of DBC being a concave-hull-generator, consider the schematic diagram shown in Figure 3. The protein array shown in Figure 3 (right-hand-side) clearly distinguishes the outer and internal boundary fences. In this particular case, outer and inner boundary fences can be created following the closed loops

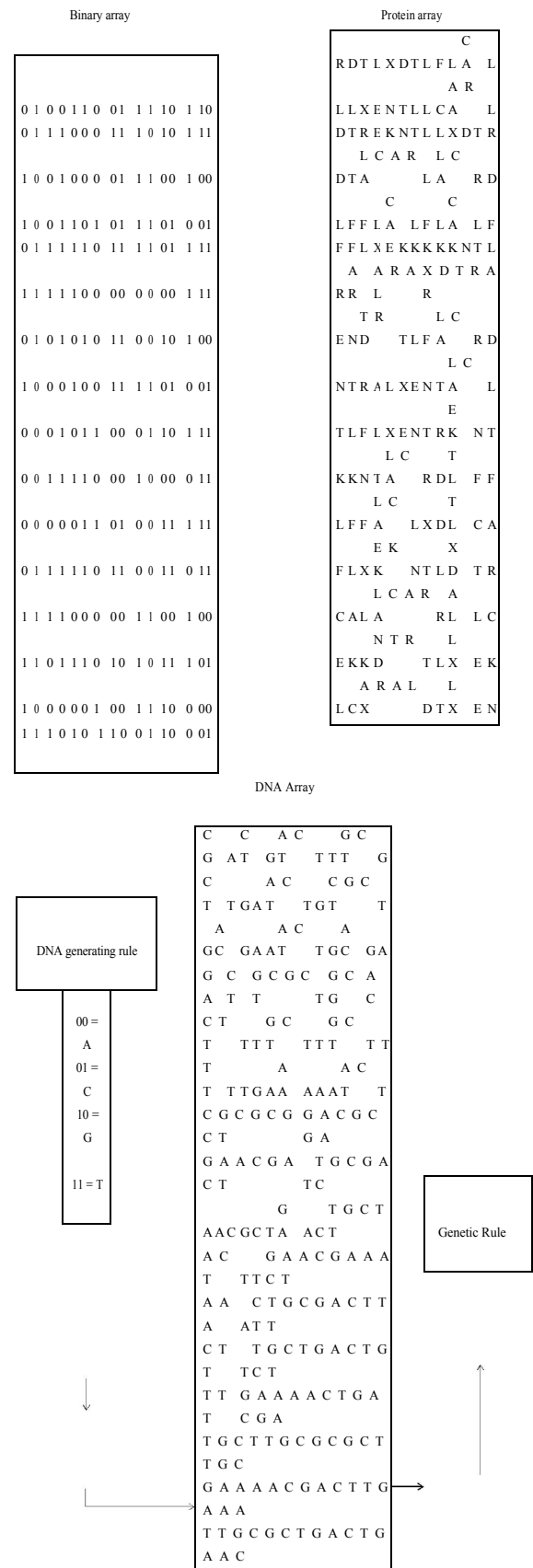
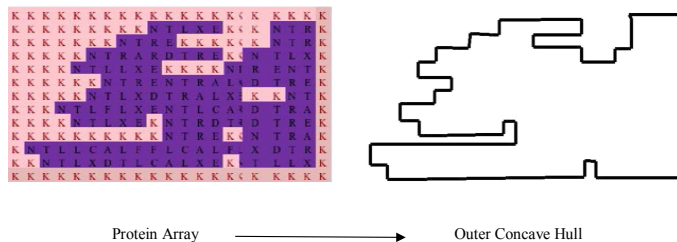


Figure 2. The DBC.

Table 1. The genetic code.

No	Amino Acid (single-letter symbol)	Codon in term of DNA base-pairs
1	Isoleucine (I)	ATT, ATC, ATA
2	Leucine (L)	CTT, CTC, CTA, CTG, TTA, TTG
3	Valine (V)	GTT, GTC, GTA, GTG
4	Phenylalanine (F)	TTT, TTC
5	Methionine (M)	ATG
6	Cysteine (C)	TGT, TGC
7	Alanine (A)	GCT, GCC, GCA, GCG
8	Glycine (G)	GGT, GGC, GGA, GGG
9	Proline (P)	CCT, CCC, CCA, CCG
10	Threonine (T)	ACT, ACC, ACA, ACG
11	Serine (S)	TCT, TCC, TCA, TCG, AGT, AGC
12	Tyrosine (Y)	TAT, TAC
13	Tryptophan (W)	TGG
14	Glutamine (Q)	CAA, CAG
15	Asparagine (N)	AAT, AAC
16	Histidine (H)	CAT, CAC
17	Glutamic acid (E)	GAA, GAG
18	Aspartic acid (D)	GAT, GAC
19	Lysine (K)	AAA, AAG
20	Arginine (R)	CGT, CGC, CGA, CGG, AGA, AGG
-	Stop (X)	TAA, TAG, TGA


Figure 3. The concept of outer concave hull from the view point of protein array

guided by an element denoted as K of the protein array. This may help simplify the concave hull generation process.

Proposed concave hull generating procedure

This section describes the proposed concave hull generation procedure. The procedure underlies four steps, as follows:

Step 1—creating an IFS fractal

To create an IFS fractal in the form of a point-cloud consisting of $N + 1$ points, $\{(x_0, y_0), (x_i, y_i) \mid i = 1, \dots, N\}$, an algorithm called IFS Algorithm can be used [12,13,17,18, 22], as follows.

IFS Algorithm:

	Seed (x_0, y_0)
Input	Number of Iterations N
	Mapping Parameters $(a_j, b_j, c_j, d_j, e_j, f_j), j = 1, \dots, n$
	Probabilities $(p_j \mid j = 1, \dots, n)$
Calculation	
	$w_1 = [0, p_1], \dots, w_j = [cp_{j-1}, cp_j], \dots, w_n = [cp_{n-1}, cp_n]$
	For $i = 1, \dots, N$
	Generate a Random Number: $r_i \in [0, 1]$
	If $r_i = w_1$ Then $x_i = a_1 x_{i-1} + b_1 y_{i-1} + e_1, y_i = c_1 x_{i-1} + d_1 y_{i-1} + f_1$
Iteration	...
	If $r_i = w_j$ Then $x_i = a_j x_{i-1} + b_j y_{i-1} + e_j, y_i = c_j x_{i-1} + d_j y_{i-1} + f_j$

...
If $r_i = w_n$ Then $x_i = a_n x_{i-1} + b_n y_{i-1} + e_n, y_i = c_n x_{i-1} + d_n y_{i-1} + f_n$

An IFS Algorithm has three segments, namely, Input, Calculation, and Iteration. In the Input segment four inputs items are set by the user, namely, Seed (x_0, y_0) , Number of Iterations (N), Mapping Parameters $((a_j, b_j, c_j, d_j, e_j, f_j), j = 1, \dots, n)$, and Probabilities $((p_j \mid j = 1, \dots, n))$. The Calculation segment calculates the relative weights $w_1 = [0, p_1], \dots, w_j = [cp_{j-1}, cp_j], \dots, w_n = [cp_{n-1}, cp_n]$ where $cp_j = p_1 + \dots + p_j, j = 1, \dots, n$, of the affine mappings needed to create points, $\{(x_i, y_i) \mid i = 1, \dots, N\}$. The Iteration segment creates the points $\{(x_i, y_i) \mid i = 1, \dots, N\}$ in a recursive manner $x_i = a_j x_{i-1} + b_j y_{i-1} + e_j, y_i = c_j x_{i-1} + d_j y_{i-1} + f_j, j = 1, \dots, n, i = 1, \dots, N$, preserving, at the same time, the relative weights of the affine mappings. It is worth mentioning that $p_1 + \dots + p_n = 1$ and all affine mappings are contracting mappings. See [17,18] for the conditions needed for an affine mapping to be included in an IFS. In addition, in most cases, Seed (x_0, y_0) is equal to $(0, 0)$.

Step 2—Creating a binary array

In this step, a binary array $\mathbf{BA} = (\dots, b_{kl}, \dots), \forall b_{kl} \in \{0, 1\}$, is created from $(x_i, y_i), i = 0, \dots, N$, i.e., from the point-cloud that models a fractal. To do this, a 2-dimensional grid is considered. To define the grid, the intervals $x_{\min}, \dots, x_{\max}$ and $y_{\min}, \dots, y_{\max}$ are needed where $x_{\min} \leq \min\{x_i \mid i = 0, \dots, N\}, x_{\max} \geq \max\{x_i \mid i = 0, \dots, N\}, y_{\min} \leq \min\{y_i \mid i = 0, \dots, N\}, y_{\max} \geq \max\{y_i \mid i = 0, \dots, N\}, x_{k+1} = x_k + x, k = 1, 2, \dots, Nk, x_{k=0} = x_{\min}, x_{k=Nk} = x_{\max}, y_{l+1} = y_l + y, y_{l=0} = y_{\min},$ and $y_{l=Nl} = y_{\max}$.

$$b_{kl} = \begin{cases} 1, & (x_i \geq x_k) \wedge (x_i \leq x_{k+1}) \wedge (y_i \geq y_l) \wedge (y_i \leq y_{l+1}), \exists i \in \{0, \dots, N\} \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

It is worth mentioning that $x_{\min}, x_{\max}, y_{\min},$ and y_{\max} are user-defined values in accordance with the above conditions.

Step 3—Creating a protein array

In this step, the binary $\mathbf{BA} = (\dots, b_{kl}, \dots)$ is transformed into a protein array using the DBC described in Section 2. The protein array is denoted as $\mathbf{Protein} = (\dots, \text{Protein}_{ij}, \dots)$.

Needless to say, $\forall \text{Protein}_{ij} \in \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, X, Y\}$.

Step 4—Recreating binary array

In this step, the protein array $\mathbf{Protein}$ is transformed into another binary array $\mathbf{PB} = (\dots, PB_{ij}, \dots)$ using a user-defined rule. The goal is to preserve the information of outer and inner segments of the fractal shape for generating the outer concave hull. For example, consider the arbitrary case shown in Figure 4. In this case, one can set a rule as follows: If $\text{Protein}_{ij} = K$ and $\text{Protein}_{i+1,j} = K$, then $PB_{ij} = 1$; Otherwise $PB_{ij} = 0$. If this rule is applied to the protein array shown in Figure 4 (left-hand-side), then a binary array $\mathbf{PB} = (\dots, PB_{ij}, \dots)$ also shown in Figure 4 (right-hand-side) is produced. As observed from Figure 4, this binary array clearly distinguishes the outer boundary (given by the digit 1) from the inner boundary (given by the digit 0).

Step 5—Determination of Boundary Fence

This is the last step where the boundary fence array $\mathbf{BF} = (\dots, BF_{ij}, \dots)$ is determined by transforming the $\mathbf{PB} = (\dots, PB_{ij}, \dots)$. The goal is to find out the coordinates of the outer boundary fence. The procedure of getting $\mathbf{PB} = (\dots, PB_{ij}, \dots)$ is described as follows.

Let $CM = \{-1, 0, 1\} \times \{-1, 0, 1\} - \{(0, 0)\}$ be a set and (p, q) be a member of it, i.e., $(p, q) \in CM$. Let $BF_{ij}(p, q)$ be a binary digit, i.e. $BF_{ij}(p, q) \in \{0, 1\}$, as

defined by the equation (2).

$$BF_{ij}(p, q) = \begin{cases} PB_{ij} \wedge (PB_{ij} \geq 0) \wedge (PB_{i+pq+q} = 0) \\ 0, otherwise \end{cases} \quad (2)$$

All possible values of $BF_{ij(p,q)}$ can be added to determine the elements of PB. This yields equation (3), as follows

The coordinates of the elements corresponding to $BF_{ij} > 0$ represent the outer boundary fence or the outer concave hull. Figure 5 shows the $BF = (..., BF_{ij}, ...)$ that has been determined from the $PB = (..., PB_{ij}, ...)$ shown in Figure 4 using the procedure described above. One can observe from figure 5 that $BF_{ij} > 0$ clearly defines the required concave hull.

Case study

This section describes a case study where an IFS fractal modeling a snow crystal. The mapping parameters and the probabilities are shown in Table 2. As listed in Table 2, seven affine mappings are used to create the model of the snow crystal in terms of a point-cloud. The results obtained applying the Steps 1-5, as described in Section 3, are shown in Figure 6.

In particular, Figure 6a shows the point-cloud (the result of Step 1), Figure 6b shows the binary array (the result of Step 2), Figure 6c shows the protein array (the result of Step 3), Figure 6d shows the binary array called PB (the result of Step 4), Figure 6e shows the boundary fence array (the result of Step 5). As observed in Figures 6a-e, the outer and inner segments become distinguishable due to the application of the Steps 1-5 in a successive manner. A physical model has also been manufactured by an additive manufacturing equipment (a 3D printer), as shown in Figure 6f. In doing so, the outer concave hull shown in

Table 2. Settings of IFS.

Mapping	Mapping Sets (i)						
Parameters	1	2	3	4	5	6	7
a_i	0.5	0.33333	0.33333	0.33333	0.33333	0.33333	0.33333
b_i	-0.2887	0	0	0	0	0	0
c_i	0.28868	0	0	0	0	0	0
d_i	0.5	0.33333	0.33333	0.33333	0.33333	0.33333	0.33333
e_i	0	0.57735	0	-0.5774	-0.5774	0	0.57735
f_i	0	0.33333	0.66667	0.33333	-0.3333	-0.6667	-0.3333

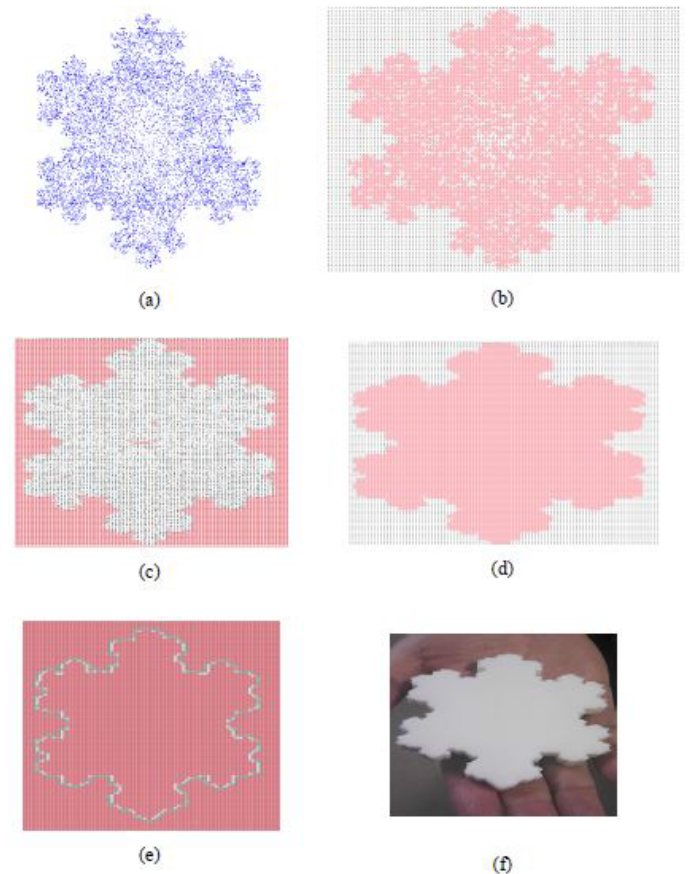


Figure 6. A case study of concave hull generation and physical model building.

Figure 6e has been used to generate the STL data. The STL data generating process can be found in [16].

This case study clearly demonstrates that the presented concave hull generation procedure is useful means to handle complex shapes for the sake of manufacturing their physical models.

Concluding remarks

Fractal geometry has extensively been used to quantify the complexity and normality/abnormality of the shapes observed in the living organisms (e.g., cell, tissue, lung, blood vassals, brain structures). These shapes are primarily represented by point-clouds having internal and external boundary fences (concave hulls). The extraction of these boundary fences is not an easy task. This study sheds some lights on this issue by proposing a simple but effective concave hull generating procedure where an in silico DNA-Based Computing plays an important role. It is demonstrated that the proposed concave hull

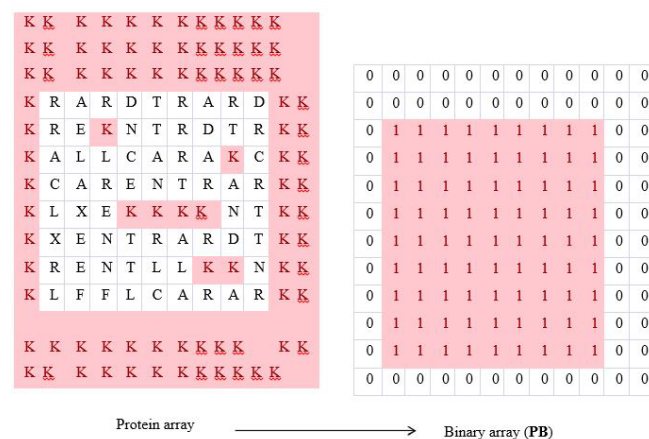


Figure 4. Significance of PB.

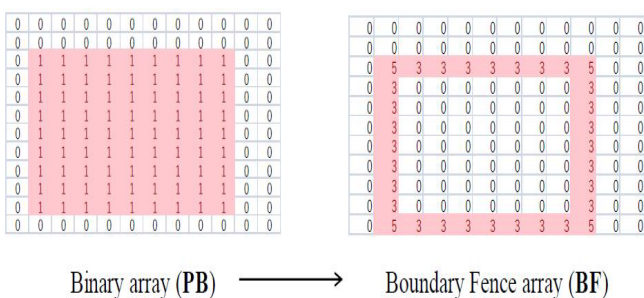


Figure 5. Significance of boundary fence array.

generating procedure is able to create the outer boundary fence of a fractal (*i.e.*, point-cloud created by an IFS) in a lucid manner. Further study can be carried out to generate the inner concave hulls of IFS fractals. Nevertheless, to get benefited from the capability of additive manufacturing in producing complex shapes having biomedical significance, the studies similar to this one must be continue in the years to come.

References

1. Global Diabetes Plan 2011-2021. International Diabetes Federation, Belgium.
2. American Diabetes Association; National Heart, Lung and Blood Institute; Juvenile Diabetes Foundation International; National Institute of Diabetes and Kidney Disease; American Heart Association. Diabetes mellitus: a major risk factor for cardiovascular disease, 1999, *Circulation*, 100: 1132-1133.
3. Kudat H, Akkaya V, Sozen AB, Salman S, Demirel S, et al. (2006) Heart rate variability in diabetes patients. *J Int Med Res* 34: 291-296. [[Crossref](#)]
4. Tarvainen MP, Cornforth DJ, Kuoppa P, Lipponen JA, Jelinek HF (2013) Complexity of heart rate variability in type 2 diabetes - effect of hyperglycemia. *Conf Proc IEEE Eng Med Biol Soc* 2013: 5558-5561. [[Crossref](#)]
5. Mirza M, Lakshmi ANK (2012) A comparative study of Heart Rate Variability in diabetic subjects and normal subjects. *International Journal of Biomedical and Advance Research* 3: 640-644.
6. Task Force of the European Society of Cardiology and the North American Society of Pacing and Electrophysiology, 1996. Heart rate variability: standards of measurement, physiological interpretation, and clinical use. *Circulation* 93: 1043-1065.
7. Ivanov PC, Amaral LA, Goldberger AL, Havlin S, Rosenblum MG, et al. (1999) Multifractality in human heartbeat dynamics. *Nature* 399: 461-465. [[Crossref](#)]
8. Smith RL, Wathen ER, Abaci PC, Bergen NHV, Law IH, et al. (2009) Analyzing Heart Rate Variability in Infants Using Non-Linear Poincare Techniques. *Computer in Cardiology* 36: 673-876.
9. Stanley HE, Amaral LA, Goldberger AL, Havlin S, Ivanov PCh, et al. (1999) Statistical physics and physiology: monofractal and multifractal approaches. *Physica A* 270: 309-324. [[Crossref](#)]
10. Peng CK, Havlin S, Stanley HE, Goldberger AL (1995) Quantification of scaling exponents and crossover phenomena in nonstationary heartbeat time series. *Chaos* 5: 82-87. [[Crossref](#)]
11. Baumert M, Javorka M, Seeck A, Faber R, Sanders P, et al. (2012) Multiscale entropy and detrended fluctuation analysis of QT interval and heart rate variability during normal pregnancy. *Comput Biol Med* 42: 347-352. [[Crossref](#)]
12. Rhaman Md, Karim AHM, Hasan M, Sultana J (2013) Successive RR Interval Analysis of PVC with Sinus Rhythm Using Fractal Dimension, Poincare Plot and Sample Entropy Method. *IJ Image, Graphics and Signal Processing* 2: 17-24.
13. Hurst HE, Black RP, Sinaika YM (1965) Long-term Storage in Reservoirs: An experimental Stud, Constable, London.
14. Gospodinov M, Gospodinova E (2005) The graphical methods for estimating Hurst parameter of self-similar network traffic. International Conference on Computer Systems and Technologies pp. IIIB.19-1-IIIB.19-6.
15. Gospodinova E, Gospodinov M, Georgieva-Tsaneva G, Cheshmedjiev K (2015) Spectral analysis of heart rate variability. International Conference AUTOMATICS AND INFORMATICS, Bulgaria, Sofia, pp 95-98.
16. Dey N, Das A, Chaudhuri SS (2012) Wavelet Based Normal and Abnormal Heart Sound Identification Using Spectrogram Analysis. *International Journal of Computer Science & Engineering Technology* 3.
17. Dey N, Samanta S, Yang SH, Chaudhuri SS, Das A (2013) Optimisation of Scaling Factors in Electrocardiogram Signal Watermarking using Cuckoo Search. *International Journal of Bio-Inspired Computation* 5: 315-326.
18. Mukherjee A, Dey G, Dey M, Dey N (2014) Web-based Intelligent EEG signal Authentication and Tamper Detection System for Secure Telemonitoring. Published by Brain-Computer Interfaces: Current Trends and Applications by Springer-Verlag, Germany, 2014.
19. Nandi S, Roy S, Dansana J, Karaa W, Ray R, et al. (2014) Cellular Automata based Encrypted ECG-hash Code Generation: An Application in Inter-human Biometric Authentication System. *International Journal of Computer Network and Information Security* 11: 1-12.
20. Gospodinova E, Gospodinov M, Domuschiev I, Nilianjan Dey, Ashour AS, et al. (2015) Analysis of Heart Rate Variability by Applying Nonlinear Methods with Different Approaches for Graphical Representation of Results. *International Journal of Advanced Computer Science and Applications* 6: 38-45.
21. Acharjee S, Dey N, Samanta S, Das D, Roy R, et al. ECG Signal compression using Ant Weight Lifting Algorithm for Tele-monitoring. *Journal of Medical Imaging and Health Informatics* [In - press]