Generation of Glyphs for Conveying Complex Information, with Application to Protein Representations

Greg D. Pintilie[1], Brigitte Tuekam[1], and Christopher W.V. Hogue[1,2]

[1] Blueprint, Samuel Lunenfeld Research Institute, Toronto, Canada
{gpintilie, btuekam, chogue}@blueprint.org
[2] Department of Biochemistry, University of Toronto, Toronto, Canada

Abstract. We present a method to generate glyphs which convey complex information in graphical form. A glyph has a linear geometry which is specified using geometric operations, each represented by characters nested in a string. This format allows several glyph strings to be concatenated, resulting in more complex geometries. We explore automatic generation of a large number of glyphs using a genetic algorithm. To measure the visual distinctness between two glyph geometries, we use the iterative closest point algorithm. We apply these methods to create two different types of representations for biological proteins, transforming the rich data describing their various characteristics into graphical form. The representations are automatically built from a finite set of glyphs, which have been created manually or using the genetic algorithm.

1 Introduction

Humans use symbols for communication of complex information in many contexts. One example is written language, where every letter or combination of letters typically represents a sound used in spoken language. Using these building blocks, words, sentences, and papers can be generated which convey a wealth of information. Another example is the use of symbols in circuit diagrams. Here a symbol represents an electrical component with well-defined physical properties, which can be connected to other components to create a circuit diagram, from which the functioning of a complex electronic device can be deduced. Circuit diagram symbols could be deemed less general than language symbols; for example they could not be used to convey the same information that a word or a sentence does. On the other hand, a circuit diagram can be described using only words and sentences, without the loss of information, albeit at the cost of loosing its conciseness. Our aim in this paper is to present methods that allow the same expressive power of graphical symbols to be used in other applications, to replace or at least complement written language as the only form of dissemination of information.

A lot of applications already use graphical symbols, a testament to their usefulness in conveying information. In our application we would like to go a bit

A. Butz et al. (Eds.): SG 2005, LNCS 3638, pp. 90–102, 2005.

further and allow graphical symbols to be seamlessly combined, in the same way that words are combined to create sentences, so that more complex information can be conveyed. There are great advantages to taking this extra step. For example, individual symbols for electronic components would be rather useless if they couldn't be connected together to form complete electronic circuits. In the methods we have developed, we have focused on enabling this approach, whereby various graphical symbols, or glyphs, representing concise information on their own, can be combined to express the properties or behaviour of more complex entities.

In developing these methods, we have focused on the communication of information about large organic molecules such as proteins, which are the building blocks of the cells in biological organisms. A wealth of information is available for proteins, for example their structural form, where in (or close to) a cell they are typically found, what their function and behaviour is, what other molecules they commonly bind to, and what subcomponents they are built from. Development of applications where such information is presented has been increasing, and we imagine that it would be very useful to have methods that allow the information to be conveyed in a symbolic yet concise way through graphical representations. We hope that developing such methods for one such specific application can lead to similar methods being exploited in both biology and non-biology settings.

2 Overview and Related Work

The use of glyphs for conveying quantitative information is common. For example, Wittenbrink et al. use varied glyphs that illustrate the uncertainty in magnitude and direction within vector fields [25]. Rose and Wang similarly use a method that creates dense iconic plots so that the user may discern patterns in multivariate data [20]. Automatic generation of circular glyphs that communicate information about individual textures is explored by Battiato et al. [4].

Creating glyphs to communicate more abstract information that cannot be quantified seems to be a process that requires more human involvement. Ribarsky et al. have developed an interface that allows users to create glyphs that can be linked to specific 'domain knowledge' [19]. In our approach, the glyph geometry can also be specified by the user, though the specification is done using a context free grammar similar to the one used in L-systems. L-systems were introduced by Lindenmayer to describe biological organisms [12], and later used by Prusinkiewicz in the generation of plant geometries [17]. They can also be used to describe fractals, and the variety of geometries that can be generated using such systems is immense.

A glyph is simply created and stored as a string, in which characters specify the various geometrical operations that create the glyph's geometry. It is thus easy to create new geometries simply by changing or concatenating such strings. We exploit this form within a genetic algorithm that can create a large set of diverse glyphs with minimal effort on the part of the user. Genetic algorithms have been previously used for the generation of geometrical forms by Sims [22],

as well as for other design problems [9,18]. In our work, the many resulting geometries are automatically filtered into a library of distinct glyphs that have the desired visual characteristics.

To automatically compare glyph geometries, we have looked at common geometry comparison methods. There are variety of such methods, and a good survey is provided by Alt and Guibas [1]. Most of the typical distance measures such as the Hausdorff distance, don't typically give a good measure of how visually distinct two geometries are. A better approach is to use the iterative closest point (ICP) algorithm [6]. This algorithm tries to find the best alignment between two geometries such that the sum of the squared distances between a number of corresponding sample points is minimized. We use a normalized form of this sum as a measure of the visual distinctness between two glyphs.

Having a quantitative measure that ranks the visual distinctness between glyphs also allows us to organize them in clusters according to their visual appearance. There are various clustering algorithms [7,11]; for example, one common approach for clustering points is to assign a point to the cluster whose mean it is closest to. Because it would be hard to compute the mean of a number of geometries, we cannot use this approach. Instead we use the approach of Voorhees [23], where clusters are formed based on average distances between the individuals in each cluster.

We apply these methods to create graphical representations for proteins that communicate their various properties. The most direct way to graphically portray a protein is to draw its molecular structure, as many programs do [5,14,24]. These detailed representations are useful in many ways, however they are too cumbersome to use when we just want to show the general characteristics of the protein. A simpler glyph representation would be easier to understand and comprehend all at once. Moreover the glyph representation could also be used to portray other knowledge about the object that is not obvious from its physical description.

Symbolic representations of proteins are already currently used in some applications [3,13,21]. These applications diagram the building blocks that proteins consist of, however they use very few geometrical shapes and colors. We explore ways to include more information into similar graphical representations. We have created two different types glyphs for representing proteins, which we have named *ontoglyphs* and *proteoglyphs*. Ontoglyphs make use of annotations for proteins in the form of terms from the Gene Ontology dictionary [8]. Proteoglyphs on the other hand communicate information about the structural components of the protein. Ontoglyphs and proteoglyphs are built by combining a number of smaller sub-glyphs, which are either hand-designed by users or generated by a genetic algorithm.

3 Glyph Geometry

In this work we focus on glyphs that are composed of a single continuous line that can take any 2-dimensional geometry, and can be either closed or open.

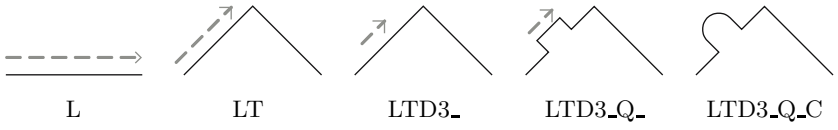

Fig. 1. A glyph is generated using geometrical operations. L creates the first line segment. Each new segment (dotted line) is affected another operation which follows: T triangle, D3 divide into 3, _ skip, Q quadrilateral, C circle.

Internally, each segment in a glyph is represented using a Bezier curve [10], thus the glyph can contain both straight and curved segments. The glyph geometry is specified as a string, in which characters are used to represent geometrical operations. Figure 1 diagrams how a glyph is constructed in this way.

In the bracketed string notations of Lindenmayer [12], a glyph string has the following form:

$$a := \mathrm{L}p_l\,(b) \tag{1}$$

$$b := c(b^n) \mid d \tag{2}$$

$$c := \mathrm{T}p_t \mid \mathrm{C}p_c \mid \mathrm{Q}p_q \mid \mathrm{D}p_d \mid \mathrm{B} \mid \mathrm{Z} \tag{3}$$

$$d := \_ \mid \mathrm{R}p_r \mid \mathrm{S}p_s \tag{4}$$

The glyph string must start with the line operation L (1), to which another operation is applied from (2). An operation taken from (3), e.g. triangle T, circle C, quadrilateral Q, divide D, back-segment B, or zig-zag Z, replaces the line segment with n new segments, to which (2) is recursively applied in succession. Alternatively, an operation from (4) may be applied to a segment, such as _ which leaves it as it is and skips to the next segment, R which creates reflecting geometry (which cannot be further modified), or S which shortens the segment from the beginning or end. The parameters p_l, p_t, p_c, ... specify values used by the geometrical operations, though they may be omitted in which case default values are used. All the operations and their parameters are listed and described in the Appendix. Note that the brackets in equations (1-4) are not needed since the grammar is unambiguous, and are usually left out of the actual glyph strings for simplicity.

4 Glyph Design Using a Genetic Algorithm

Genetic algorithms mimic the evolutionary process observed in nature, which creates a diverse population of individuals that tend to become more complex over time. A genetic algorithm involves mutation and combination of genotypes, where the genotypes contain the code necessary to create living organisms or phenotypes. Due to natural selection, only phenotypes that are fit survive. With this pruning process, nature has found a way to create many diverse individuals without exhausting every possible genotype, instead only propagating the genotypes that are likely to create fit phenotypes.

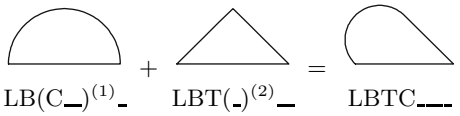

Fig. 2. A complete geometrical operation taken from one parent glyph (1) is introduced into another (2) to create a new 'offspring' glyph

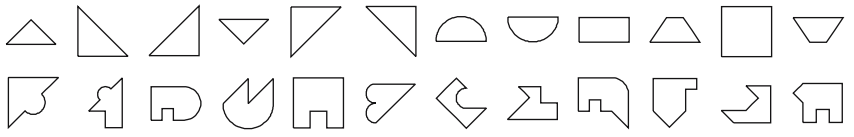

Fig. 3. Glyphs in the starting population are in the top row. The bottom row shows some of the resulting glyphs after ∼300 steps with the genetic algorithm.

Using the same evolutionary approach, we can create many glyph geometries that are varied and satisfy some desirable fitness criteria. For this we imagine that glyph strings are genotypes and that glyph geometries are the resulting phenotypes. Starting with a set of varied, user-specified genotypes, new genotypes are created by combining random pairs of existing genotypes. The combination process is illustrated in Fig. 2.

The glyph resulting from a combination operation is added to the population if it meets some fitness criteria. We define the fitness of a glyph based on its visual appearance: a glyph is considered fit if all its segments are non-intersecting, and if non-connected segments are at least some minimum distance d from each other. These criteria are meant to ensure that the resulting glyphs have the form of a single continuous line that doesn't intersect itself. The constant d is chosen depending on the size of the glyphs being created; for example if the glyphs are to fit into a square measuring 100x100 units, choosing $d = 5$ results in glyphs having geometries that don't appear to intersect, even when scaled down substantially. Figure 3 shows an example starting population and some of the resulting geometries.

5 Comparison of Glyph Geometries

The genetic algorithm can generate a very large population of glyphs, and even if every genotype is different, it is possible that some phenotypes will be similar and possibly even identical (e.g. two stacked triangles form a quadrilateral). To prune similar glyphs from the population, a measure that computes the visual distinctness of two glyphs is needed. Such a measure would be equal to zero if the glyphs are visually identical, or otherwise take a value that increases proportionally with the dissimilarity between the two glyphs. To compute this similarity measure, we use the iterative closest point (ICP) algorithm [6].

The ICP algorithm computes the alignment between two geometries which is optimal (in a least-squares sense). An initial alignment is performed such that the centers of the the two geometries coincide. A set of points is created by point-sampling one of the geometries, and the closest point on the other geometry to each sample point is found and added to a second set of points. The distance between closest points represents a local misalignment, or dissimilarity, between the two geometries. A normalized distance measure between the two glyphs can thus be computed using the root mean square deviation (RMSD) formula:

$$d(g_i, g_j) = \sigma_{ij} = \sqrt{\frac{1}{n}\sum_{k=1}^{n}(p_{ik} - p_{jk})^2}, \tag{5}$$

where g_i and g_j are the two glyphs being compared, p_{ik} and p_{jk} are corresponding closest points (one on each glyph), and n is the number of corresponding points.

After the initial alignment, a transformation is found which when applied to one of the glyphs decreases (on average) the distance between corresponding points. After computing the new set of corresponding closest points for the same initial sample points, the process is repeated as long as the RMSD decreases significantly with each iteration. The original ICP algorithm also considers rotational transformations that decrease the RMSD, however for our purposes this is not necessary. This is because we consider two identical glyphs with different orientations to be visually distinct, since glyphs are always displayed with the same orientation.

Figure 4 shows several glyph comparisons along with the computed RMSD. The number of sample points in each comparison can vary, since more complex geometries require more sample points, however because the RMSD value is normalized this value is factored out of the overall score. Thus pairs of glyphs can be ranked based on how similar they are, regardless of their complexity.

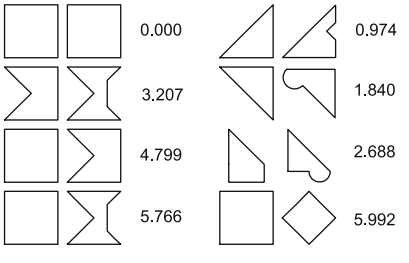

Fig. 4. Comparison of pairs of glyphs using the RMSD as a measure of visual distinctness. The RMSD is the number to the right of each pair of glyphs. In both columns, the similarity between pairs of glyphs decreases from top to bottom while the RMSD value increases.

6 Clustering of Glyph Geometries

The distance measure between glyphs can be used to automatically organize a large number of glyphs into clusters. Doing this allows us to observe which glyphs in a large set are similar to each other. It is also useful when the number of glyphs needed is smaller than the size of the set, since by selecting only one glyph from each cluster, glyphs that are more distinct from each other can be used.

Our method creates a desired number of clusters from a large number of glyphs, so that glyphs in one cluster are on average more similar to each other than to glyphs in other clusters. The algorithm, based on the work of Voorhees [23], first creates a cluster for every glyph, and then successively merges clusters until the desired number of clusters is reached. The clusters to be merged are picked so as to minimize the average-link cost function:

$$c_{AL} = \frac{1}{|C_1||C_2|} \sum_{g_i \epsilon C_1} \sum_{g_j \epsilon C_2} d(g_i, g_j), \tag{6}$$

where $d(g_i, g_j)$ is the distance measure between two glyphs g_i and g_j, defined in Eqn. (5), with each glyph being from two different clusters C_1 and C_2. $|C_1|$ and $|C_2|$ refer to the sizes of the two clusters. As Eqn. (6) indicates, to compute the average-link cost of merging two clusters, we sum the distances between every possible pair of glyphs, one from each cluster, and divide this value by the total number pairs of glyphs considered. Figure 5 shows clusters created using this approach. Other cost functions could also be used, such as the minimum or maximum distance between any two glyphs from different clusters, which would give slightly different results [7].

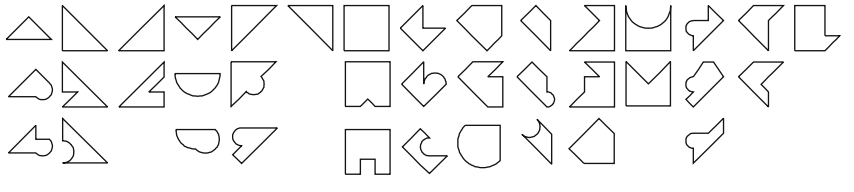

Fig. 5. Clusters of glyphs taken from a population generated using the genetic algorithm. Each cluster is shown on a separate column. Note that it is possible that some clusters contain only one glyph, if no other glyphs in the population are sufficiently similar in geometry.

7 Applications

7.1 Ontoglyphs

An ontoglyph is used to communicate *ontological* information about the protein it represents. Information about each protein is available as a list of annotations,

where each annotation is a term from the Gene Ontology dictionary [8]. Gene Ontology terms relate to one another through parent-child relationships that form a directed acyclic graph (DAG), which can also be seen as a hierarchy where a child can have more than one parent. As the hierarchy is traversed outward from the root, the terms become more and more specific; conversely traversing toward the root we would encounter more and more general terms.

To graphically represent such annotations, we use a small number of categories, where each category is assigned one or more GO terms and a unique user-designed glyph. The glyph is designed so that it is very distinct, and so that it portrays the information associated with the category it represents in an intuitive way. Alternatively, it should have a form that users already associate with this information. To create an ontoglyph, the GO graph is searched with each annotating term for its closest ancestor which belongs to a category. (The ancestor term encompasses the same information as the annotation, albeit in a more general way.) This results in a list of categories that best represent the protein's annotations, within the expressive power of the set of categories previously defined.

The ontoglyph is built by inserting the glyphs of the computed cateogories into the segments of a rectangle. The categories for an ontoglyph are divided into three classes: localization, binding (or specificity), and function. The glyphs belonging to localization categories are embedded into the left segment of the rectangle, while the binding and function glyphs are embedded into the top segment. Thus the ontoglyph is created with a string that has the form:

$$\mathrm{LBQD}n S_1..S_n \mathrm{D}m T_1..T_m \mathrm{R}, \tag{7}$$

where $S_1..S_n$ are strings for the glyphs to be embedded into the side segment and $T_1..T_m$ are strings for the glyphs to be embedded into the top segment. Note that the glyphs embedded into the left segment are reflected onto the right segment by the 'R' operation, and the 'B' operation creates the bottom segment of the rectangle. Figure 6 shows an example ontoglyph.

For a complete list of categories and glyphs that can be included into an ontoglyph, and the biological information that each one communicates, we refer the reader to the web site: http://seqhound.blueprint.org/cgi-bin/ontoglyphs.pl. Table 1 lists the glyphs and categories used in the ontoglyph in Fig. 6. Because the glyphs are always the same for each category, once the user has learned

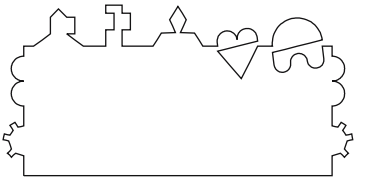

Fig. 6. Ontoglyph for the protein with Gene Identifier (GI):29466. The sub-glyphs along the side and top segments communicate ontological information about the protein.

Table 1. Some of the categories and glyphs for representing information in ontoglyphs

Binding	Function	Location
Calmodulin binding	Development	Actin cytoskeleton
Cytoskeletal protein binding	Cell motility and structural activity	Cytoplasm
ATP binding		

this basic alphabet, the characteristics of any protein can be discerned from its corresponding ontoglyph. The total number of categories is kept relatively small (at about 84), and so while an ontoglyph may not represent the information contained in the annotations for a protein exactly, it still provides a general view which can be very useful.

7.2 Proteoglyphs

Proteoglyphs are used to show any subcomponents that a protein may contain. The largest subunits within a protein are known as *conserved domains*, and each domain has its own structural and binding properties. In a proteoglyph, each domain is shown with a separate glyph which communicates these structural and binding properties. A domain can belong to one of several structural classes based on [16]. Each class is associated with a base glyph, which illustrates the properties of the class, and which is used to construct the domain glyph. These classes are listed in Fig. 7. Binding properties for a domain are available in the form of GO terms, and as for ontoglyphs, a number of binding categories can be used to represent this information.

There are some ten thousand conserved domains in our database, and we have generated a unique geometry to represent each one. We have used the

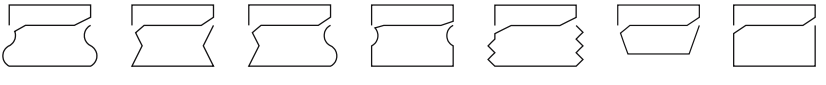

Fig. 7. Base glyphs for representing conserved domains. The left and right segments in the bottom half of each base glyph are different depending on the class. From left to right, the classes are: alpha, beta, alpha+beta, alpha/beta, membrane, small, and structure unknown. The shape of the segments mimics the physical structure: alpha structures are helix-like, and thus circular segments are used; beta structures are sheet-like, and thus straight line segments are used. Classes encompassing both structures use both curving and straight segments. The zig-zags in the base glyph for membrane domains mimics how membranes are usually depicted in diagrams, and the base glyph for small domains has smaller bottom segments.

genetic algorithm to generate them and the geometry comparison and clustering techniques to ensure they are all visually distinct. The unique geometry can effectively show whether two domain glyphs shown side by side refer to different domains, even if they have the same binding and structural characteristics. Also, we think that users may be able to identify a domain they are familiar with just from its unique geometry, much as we readily recognize familiar faces amongst many unfamiliar ones.

To create a domain glyph, the binding glyphs are inserted along the top segment of the base glyph corresponding to its class, similarly to how they are inserted into an ontoglyph, and the unique geometry for the domain is added to the bottom segment. Once the domain glyphs for a protein have been determined, the complete proteoglyph can then be constructed. A proteoglyph is composed of a single line segment which mimics the form of a protein, which is simply a chain of smaller molecules known as amino acids. Glyphs representing conserved domains are inserted along this line at their respective positions, with each glyph having a width matching the length of the domain that it represents. A proteoglyph string has the following form:

$$\mathrm{LL}r\mathrm{D}(n*2+1)\mathrm{L}s(\mathrm{L}l_1\mathrm{L}i_1..\mathrm{L}l_n\mathrm{L}i_n)\_(G_1\_..G_n\_), \tag{8}$$

where r is the number of residues the protein contains, n is the number of domains, s is the residue at which the first domain starts, $l_1..l_n$ are the lengths of domains $1..n$, $i_1..i_n$ are the number of residues between a domain and the next domain or the end of the sequence, and $G_1..G_n$ are the glyphs for domains $1..n$ respectively. An example proteoglyph is shown in Figure 8.

Fig. 8. Proteoglyph showing 4 conserved domains in a protein (GI:1081), their positions in the protein and relative sizes. Each domain glyph shows what structural class the domain belongs to and its binding properties when known. It can be seen that all the domains are different since their unique geometries differ.

8 Conclusion

We have presented methods for creating graphical symbols, or glyphs, which communicate various characteristics of complex entities such as proteins. Glyphs are created and stored as strings, so combining them together into more complex glyphs is a straightforward process. We have introduced two new graphical representations for proteins, ontoglyphs and proteoglyphs, which make use of

an alphabet of sub-glyphs as well as a large set of unique glyphs to graphically communicate some of the rich information associated with such entities. Both representations are already used in the web interfaces of BIND [2], which is a database of protein interactions, and Seqhound [15], which stores other types of biological information. Ontoglyphs are also used in a graphical application supported by BIND, the Bind Interaction Viewer (BIV), which diagrams protein interaction networks.

We believe that once users have learned the basic geometrical properties of ontoglyphs and proteoglyphs, and the sub-glyphs that are used to create them, they can quickly get an idea of a protein's functional, localization, behavioral, and structural characteristics from these representations. Because sub-glyphs intuitively depict the characteristics they communicate, and also having used symbols which biologists are already familiar with, we also believe that this system would not take a long time to become familiar with. A more concrete measure of its effectiveness is yet to be determined. While this is pursued, we hope that the methods we have presented can be further applied and developed to create more effective graphical communication tools.

Acknowledgements

This work was supported by grants to C.W.V.H. from Genome Canada through the Ontario Genomics Institute, from the Canadian Institute of Health Research and the Ontario R&D Challenge fund.

References

1. Alt, H. Guibas, L. J.: Discrete Geometric Shapes: Matching, Interpolation, and Approximation. In: Sack, J. R., Urrutia, J. (eds.): Handbook of Computational Geometry. Elsevier Science Publishers, Amsterdam, Holland, B.V. North (1999) 121–153
2. Alfarano, C., Andrade, C.E., Anthony, K., Bahroos, N., Bajec, M., ...: The Biomolecular Interaction Network Database and related tools 2005 update. Nucleic Acids Res. **33**(Database Issue) (2005) D418–D424
3. Bateman, A., Birney1, E., Cerruti, L., Durbin, R., Etwiller1, L., Eddy, S.R., Griffiths-Jones, S., Howe, K.L., Marshall, M., Sonnhammer, E.L.L.: The Pfam Protein Families Database. Nucleic Acids Research **30**(1), (2002) 276–280
4. Battiato S., Gallo G., Nicotra,S.: Glyph Representation of Directional Texture Properties. Journal of WSCG **10**(1-3) (2002) 48–54
5. Bergman, L. D., Richardson, J. S., Richardson, D. C., Brooks. J.F.P: VIEW - an exploratory molecular visualization system with user-definable interaction sequences. Proceedings of SIGGRAPH **71** (1993) 117–126
6. Besl, P. J., MacKay, N. D.: A Method for Registration of 3-D Shapes. IEEE Transactions on Pattern Analysis and Machine Inteligence **14** (1992) 239–256
7. Fasulo, D.: An Analysis of Recent Work on Clustering Algorithms. Department of Computer Science & Engineering, University of Washington (1999)
8. The Gene Ontology Consortium: Gene Ontology: tool for the unification of biology. Nature Genetics **25** (2000) 25–29

9. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading Massachusetts (1989)
10. Hoffmann, C.M.: Geometric and solid modeling: an introduction. Morgan Kaufmann Publishers Inc., San Francisco, California (1989)
11. Jain, A.K., Dubes, R.C.: Algorithms for clustering data. Prentice-Hall (1988)
12. Lindenmayer, A.: Mathematical models for cellular interaction in development, Parts I and II. Journal of Theoretical Biology **18** (1968) 280–315
13. Marchler-Bauer, A., Panchenko, A.R., Shoemaker, B.A., Thiessen, P.A., Geer, L.Y., Bryant, S.H.: CDD: a database of conserved domain alignments with links to domain three-dimensional structure. Nucleic Acids Research **30**(1) (2002) 281–283
14. Martz, E.: 3D Molecular Visualization with Protein Explorer. In: Krawetz, S.A., Womble, D.D. (eds.): Introduction to Bioinformatics, Humana Press, Totowa NJ (2003) 565–586
15. Michalickova K., Bader G.D., Dumontier M., Lieu H., Betel D., Isserlin R., Hogue C.W.: Seqhound: biological sequence and structure database as a platform for bioinformatics research. BMC Bioinformatics, **3**(1), (2002) 32–45
16. Murzin, A.G., Brenner, S.E., Hubbard, T., Chothia, C.: SCOP: a structural classification of proteins database for the investigation of sequences and structures. J. Mol. Biol. **247** (1995) 536–540
17. Prusinkiewicz, P., Hanan, J.: Visualization of botanical structures and processes using parametric L-systems. In: Scientific Visualization and Graphics Simulation. J. Wiley & Sons, Chichester (1990) 183–201
18. Renner G., Ekrt A.: Genetic algorithms in computer aided design. Computer-Aided Design **35**(8) (2003) 709–726
19. Ribarsky, W., Ayers, E., Eble, J., Mukherjea, S.: Glyphmaker: Creating customized visualizations of complex data. IEEE Computer **27**(7) (1994) 57–64
20. Rose, S.J., Wong, P.C.: DriftWeed: a visual metaphor for interactive analysis of multivariate data. Visual Data Exploration and Analysis VII (Proceedings of SPIE) **3960** (2000) 114–121
21. Schultz, J., Copley, R.R., Doerks, T., Ponting, C.P., Bork, P.: SMART: A Web-based tool for the study of genetically mobile domains. Nucleic Acids Res **28** (2000) 231–234
22. Sims, K.: Evolving Virtual Creatures. Computer Graphics (1994) 15–22
23. Voorhees, E. M.: Implementing Agglomerative Hierarchic Clustering Algorithms for use in Document Retrieval. Information Processing & Management **22** (6) (1986) 465–476
24. Wang Y., Geer L.Y., Chappey C., Kans J.A., Bryant S.H.: Cn3D: sequence and structure views for Entrez. Trends Biochem Sci. **25**(6) (2000) 300–302
25. Wittenbrink, C.M., Pang, A.T., Lodha, S.K.: Glyphs for Visualizing Uncertainty in Vector Fields. Proceedings of IEEE Transactions on Visualization and Computer Graphics **2**(3) (1996) 266–279

Appendix: Geometry Operations and Examples

Operation	Effect	Parameters	# segments created
L	line	L: length (L100) A: angle (A0)	1
T	triangle	h[1]|H: height (h0.5) o|O: position of peak along base line (o0.5)	2
C	circle	h|H: height (h0.5) o|O: perp. offset of center from base line (o0)	2 or 4
Q	quadrilateral	h|H: height (h0.5) w|W: top width (w1) o|O: offset of top segment along base line (o0)	3
Dn	divide	n: number of segments (1) [l|L]^n: length of each segment ([l1/n]^n)	n
S	shorten	b|B: length to shorten from beginning (b0) e|E: length to shorten from end (e0)	1
B	add segment going backward		2
Z	zigzag (same as BB)		3
R	reflect	b|B: length to shorten reflected geometry from beginning	x [2]
_	none		0

L_	 LT_	 LTh1o1__	 LC_	 LCh-.5o-.5	 LQ___	 LQh.75w.5o.25
 LD2T_C_	 LD3l.2l.6Qh1__ Th-.25__Ch.5o.5	 LBC_D2T_Q	 LZTT_Th-.5o1_Q_D3_ Ch-.5o-.5____Se5_Rb5			 LBQh.25w.5 o-.25_To-.25__R

[1] Small caps indicates the value is multiplied by the length of the segment.
[2] Number of segments reflected, which are all subsequently skipped.