L-system Implementation of Multiresolution Curves Based on Cubic B-Spline Subdivision

K. Poon*, L. Bateman, R. Karwowski, P. Prusinkiewicz and F. Samavati
University of Calgary

Abstract

It has been previously shown that L-systems can be used to generate subdivision and reverse subdivision curves [Prusinkiewicz et al. 2003]. In this paper we show that L-systems can also be used to generate multiresolution curves. The L-system description captures the locality of the concept of multiresolution curves.

1 Introduction

Finkelstein and Salesin [1994] introduced multiresolution curves as a curve representation method. The multiresolution representation supports the ability to change the overall "sweep" of a curve while maintaining its fine details, or "character". Finkelstein and Salesin used a wavelet-based notation, which uses "filters", which are represented as large matrices. Bartels and Samavati [2000] introduced a general approach to generate local filters of multiresolution curves based on reverse subdivision. In this paper, we present context-sensitive parametric L-systems as an alternative method for representing the local filters. This idea is an extension of the L-system based method for generating subdivision curves presented by Prusinkiewicz et al [2003]. The L-system notation for multiresolution curves leads to a simpler, more intuitive implementation of multiresolution curves by eliminating the need for index and matrix notation used in the traditional approach.

2 Multiresolution Curves

Multiresolution curves, introduced in [Finkelstein and Salesin 1994] allow editing of a curve's character without affecting its sweep (Figure 1) and the editing of a curve's sweep without affecting its character (Figure 2).

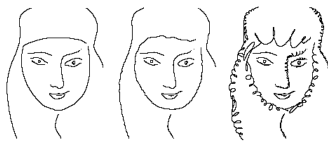

Figure 1: Editing a curve's character without affecting its sweep (from [Finkelstein and Salesin 1994]).

Multiresolution analysis can be broken into two parts: analysis and synthesis. During analysis, the original curve is coarsened. Detail information lost in this coarsening is stored. During synthesis the curve is rebuilt by performing subdivision on the coarse curve, then adding the stored detail information.

*e-mail: klpoon@cpsc.ucalgary.ca

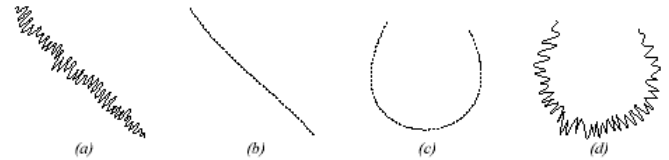

Figure 2: Editing a curve's sweep without affecting its character (from [Finkelstein and Salesin 1994]).

2.1 Matrix Notation

In matrix notation, the original curve, C^n, is represented as a vector of m points:

$$C^n = [C_1^n, C_2^n, C_3^n, \ldots, C_m^n]^T. \quad (1)$$

After one step of analysis the coarse curve, C^{n-1}, is a vector of m' points:

$$C^{n-1} = [C_1^{n-1}, C_2^{n-1}, C_3^{n-1}, \ldots, C_{m'}^{n-1}]^T. \quad (2)$$

The stored detail information, D^{n-1}, is a vector of $m - m'$ points:

$$D^{n-1} = [D_1^{n-1}, D_2^{n-1}, D_3^{n-1}, \ldots, D_{m-m'}^{n-1}]^T. \quad (3)$$

Analysis can be represented as two matrix multiplications:

$$C^{n-1} = A^n C^n \quad (4)$$

$$D^{n-1} = B^n C^n. \quad (5)$$

Synthesis can be represented with the matrix equation:

$$C^n = P^n C^{n-1} + Q^n D^{n-1}, \quad (6)$$

where P^n is the subdivision matrix and Q^n is the detail-restoring matrix, as determined by B-spline wavelets. A^n and B^n must satisfy biorthogonality condition:

$$\left[\frac{A^n}{B^n}\right] = [P^n | Q^n]^{-1}. \quad (7)$$

P^n is a known banded matrix for most curve schemes, however, it is not easy to compute the A^n, B^n and Q^n matrices. Finkelstein and Salesin focus on the cubic B-spline subdivision scheme. A banded but complicated Q^n is computed using B-spline wavelets. Consequently, Q^n is a local filter. A^n and B^n in that setting are full matrices, i.e., they are global filters. In order to have linear time analysis operations in equations 4 and 5, two banded linear systems are solved.

Analysis and synthesis can be carried out recursively (Figures 3 and 4).

2.2 Multiresolution Based on Reverse Subdivision

Samavati and Bartels [1999] introduced a general technique for generating multiresolution filters by reversing subdivision. This technique works for any subdivision scheme, and Q^n is a very simple matrix. However, A^n and B^n are still full matrices, i.e., global filters. In a subsequent work [Bartels and Samavati 2000], several sets of local multiresolution filters are generated based on reversing

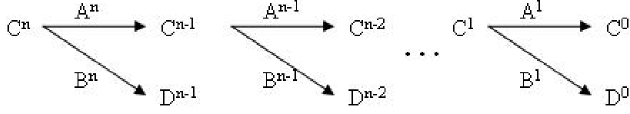

Figure 3: Applying analysis recursively (from [Finkelstein and Salesin 1994]).

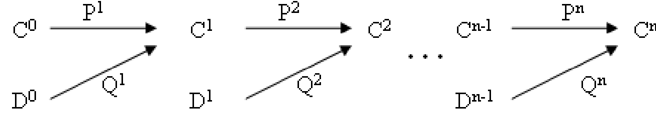

Figure 4: Applying synthesis recursively.

of subdivision schemes. We use here some of these filters. There are different filters for points near or at the endpoints of an open curve. For simplicity, we only discuss the general, not endpoint filters, although our implementation includes endpoint filters as well.

Analysis for a cubic B-spline multiresolution representation of a small, closed curve is given by the equations:

$$C^{n-1} = A^n C^n \tag{8}$$

$$\begin{bmatrix} C_1^{n-1} \\ C_2^{n-1} \\ C_3^{n-1} \\ C_4^{n-1} \end{bmatrix} = \begin{bmatrix} -\frac{1}{2} & 2 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{2} & 2 & -\frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\frac{1}{2} & 2 & -\frac{1}{2} & 0 \\ -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & -\frac{1}{2} & 2 \end{bmatrix} \begin{bmatrix} C_1^n \\ C_2^n \\ C_3^n \\ C_4^n \\ \vdots \\ C_8^n \end{bmatrix}$$

and

$$D^{n-1} = B^n C^n \tag{9}$$

$$\begin{bmatrix} D_1^{n-1} \\ D_2^{n-1} \\ D_3^{n-1} \\ D_4^{n-1} \end{bmatrix} = \begin{bmatrix} \frac{3}{2} & -1 & \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & -1 \\ \frac{1}{4} & -1 & \frac{3}{2} & -1 & \frac{1}{4} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{4} & -1 & \frac{3}{2} & -1 & \frac{1}{4} & 0 \\ \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & -1 & \frac{3}{2} & -1 \end{bmatrix} \begin{bmatrix} C_1^n \\ C_2^n \\ C_3^n \\ C_4^n \\ \vdots \\ C_8^n \end{bmatrix}$$

The corresponding synthesis equation is:

$$C^n = P^n C^{n-1} + Q^n D^{n-1} \tag{10}$$

$$\begin{bmatrix} C_1^n \\ C_2^n \\ C_3^n \\ C_4^n \\ \vdots \\ C_8^n \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ \frac{1}{8} & \frac{3}{4} & \frac{1}{8} & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & \frac{1}{8} & \frac{3}{4} & \frac{1}{8} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{8} & 0 & \frac{1}{8} & \frac{3}{4} \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ \frac{3}{4} & \frac{1}{8} & 0 & \frac{1}{8} \end{bmatrix} \begin{bmatrix} C_1^{n-1} \\ C_2^{n-1} \\ C_3^{n-1} \\ C_4^{n-1} \end{bmatrix}$$

$$+ \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{4} & \frac{1}{4} & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & \frac{1}{4} & \frac{1}{4} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{4} & \frac{1}{4} \\ 0 & 0 & 0 & 1 \\ \frac{1}{4} & 0 & 0 & \frac{1}{4} \end{bmatrix} \begin{bmatrix} D_1^{n-1} \\ D_2^{n-1} \\ D_3^{n-1} \\ D_4^{n-1} \end{bmatrix}$$

This matrix notation implies a global mapping of old points to new points at each step, but as we can see from the sparse, banded structure of the multiresolution matrices, it is possible to describe both the synthesis and analysis in local terms.

In the local approach, A, B, P and Q are filters that are applied only in the immediate neighborhood of the target points. At each iteration the filters remain the same. This contrasts the local approach from the global approach, in which the matrices change size at each analysis and synthesis iteration because they must operate on a different number of points at each iteration.

L-systems are consistent with the notion of a local algorithm. They directly capture the locality of the multiresolution algorithm.

3 L-systems

L-systems [Lindenmayer 1968] are string-rewriting systems. An L-system consists of an *alphabet*, V, an *axiom*, ω, and a set of *productions*, P, defined over V. Each production in P replaces one or more letters of V with zero or more letters in V. A *word*, x, in the system is a sequence of letters in V. The system's current state is represented by a word. The axiom, ω, is a special word, which represents the system's initial state. At each time step, the production rules are applied in parallel to each of the letters in the current word to produce a new word.

Parametric L-systems [Prusinkiewicz and Lindenmayer 1990] extend the basic concept of L-systems by assigning additional attributes (*parameters*) to L-system symbols. A parameter can be a number, or a C++ structure [Karwowski 2002]. A *module* in a parametric L-system consists of a letter in V and zero or more parameters. In parametric L-systems, a word consists of a sequence of modules.

We implemented multiresolution curves in the language L+C, which adds L-system constructs to C++ [Karwowski 2002]. In L+C, the syntax for declaring a module with parameters is:

```
module identifier(list of parameter types).
```

The following code declares a module, `C`, which represents a point. `C` has a parameter of type `V2f`, which is a pre-defined L+C type that represents a 2D point or vector:

```
module C(V2f);
```

L+C supports context-sensitive productions. A context sensitive production replaces a module, called the strict predecessor, using information from neighboring modules, called the left and right context. The syntax for a context sensitive production is

```
lcontext < strict predecessor > rcontext:
{
  ...
}
```

The following code replaces each point with two points, each of which is a linear combination of the point being replaced and its left or right neighbor.

```
P(vl) < P(v) > P(vr):
{
  produce P(0.25vl+0.75v) P(0.75v+0.25vr);
}
```

It is possible in L+C to look to the new string for context. The $<<$ and $>>$ symbols means look to the left and right, respectively, in the produced string.

L+C also supports table L-systems [Rozenberg 1973]. This allows us to divide productions into groups. We specify which group of productions should be used at each derivation step. For example, the following L+C code segment applies the production A→AB in even steps and A→AC in odd steps.

```
int step = 0;
StartEach:{
  if(step%2 == 0) UseGroup(0);
  else UseGroup(1);
  step++;
}
group 0:
A(): {
  produce A() B()
};
group 1:
A(): {
  produce A()C()
};
endgroup
```

4 Implementing Cubic B-spline Multiresolution Curves in L+C

4.1 Topology

During multiresolution analysis, detail information accumulates. We keep detail information associated with those points it will be used to restore, but after each iteration of analysis the amount of detail we need to store for each point will more than double. To solve this problem we use a tree data structure on which L+C operates.

Tree structures can be represented within an L-system string using special branch symbols [and]. The symbol [denotes the start of a branch and the symbol] denotes the end of a branch. The branches can be nested to create trees. For example, we interpret the string A [[B] C [D]] as the tree in Figure 5.

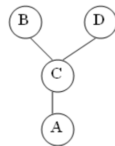

Figure 5: The tree represented by the string A [[B] C [D]].

Below we present, step-by-step, the operation of an L-system that performs the topological changes that occur during multiresolution analysis. The L-system converts a string of modules which represent points into a string in which every other module represents detail information. It then associates each piece of detail information with a point and repeats the process, storing detail information in a tree structure. During synthesis, we need to strips off layers of branching in order to access the detail information that is associated with each point.

Figure 6 shows changes in a string and the topological changes it represents. C is a point and D is detail information. The downward arrow shows the topology changes during analysis. The upward arrow shows the topology changes during the synthesis.

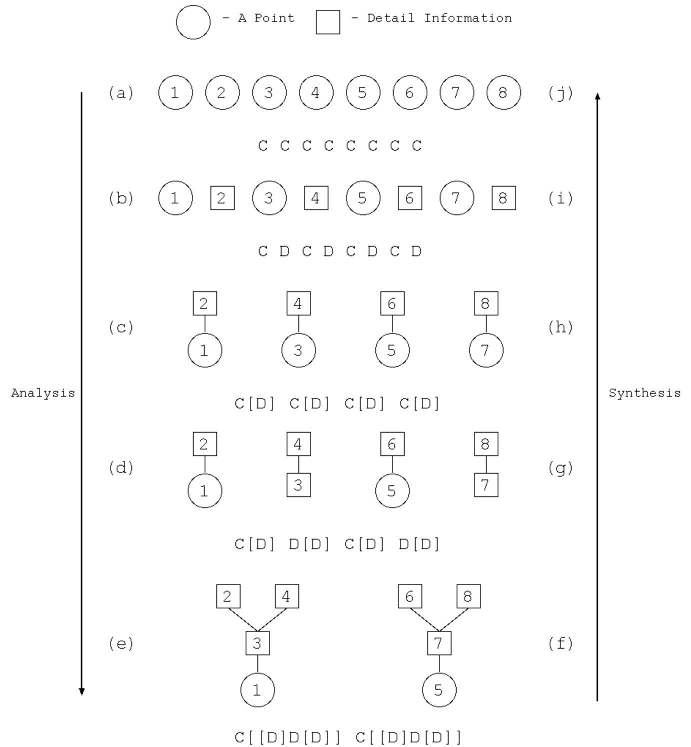

Figure 6: Topology changes during synthesis and analysis. (a) The original points. (b) and (d) Every other point is converted to detail information. (c) and (e) Each piece of detail information is associated with a point. Detail information from the previous steps is stored as branches of the detail trees. (f) Original string at start of synthesis. (g) and (i) Outer branches are stripped away. (h) and (j) Detail information is converted back into points.

4.2 L-systems Rules for Multiresolution

In this section we present our L+C implementation of multiresolution curves. This implementation is limited to closed curves. The full code listing for closed curves is given in Appendix A.

4.3 Analysis

During analysis the symbol C represents a point. The symbol Dt represents a point that is about to become a detail vector. The symbol D represents a detail vector.

We begin with a sequence of points that represents the original curve:

```
C C C C C C C ... C C
```

During analysis, even points become detail information and odd points become a point on the coarsened curve. There are two separate analysis phases.

During phase zero, we change the type of every other point to a type that represents detail information. The L+C production that changes every other point, `C`, into a point that is about to be converted into detail information, `Dt`, is:

```
C(vl)<< C(v) : {
  produce Dt(v);
}
```

If the module to the left of the strict predecessor in the produced string is a `C`, this rule replaces the strict predecessor with a `Dt`. After this phase, the string has the form:

```
C Dt C Dt C Dt C Dt...
```

In phase one we calculate the positions of the coarse points and the value of the detail vectors. The two main rules are the reverse subdivision rule, corresponding to the application of filter A, and the detail-storage rule, corresponding to the application of filter B. The reverse subdivision rule coarsens a set of fine points. An L-system implementation of reverse subdivision has been presented in [Prusinkiewicz et al. 2003]. This implementation replaces a pair of points with a single point. Our implementation replaces a single point with a point. This difference arises because we convert every other point to detail information, whereas in plain reverse subdivision, we do not need to store the detail information. Our L+C code to perform reverse subdivision is:

```
Dt(vl) < C(v) > Dt(vr):{
  produce EB() C(-0.5*vl + 2*v + -0.5*vr) SB() H();
}
```

This production calculates the new location of a point, based on the reverse subdivision coefficients given by the A matrix in equation 8. An end branch module is inserted before the new coarse point and a start branch module is inserted after the new coarse point. This puts the detail information to the right of each point into a tree associated with that point. An `H` module is placed after each start branch module as a block that prevents modules within the tree from "seeing" modules outside the tree.

The detail information lost in the reverse subdivision process is stored within the `D` modules. The following L+C production implements the detail storage filter B:

```
Dt(vll) C(vl) < Dt(v) > C(vr) Dt(vrr) :{
  produce
   D(0.25*vll + -1*vl + 1.5*v + -1*vr + 0.25*vrr);
}
```

This production calculates the detail information to store based on coefficients given by the B matrix in equation 9.

For example, after the first iteration of analysis, the string has the form:

```
C [ D ] C [ D ] C [ D ] C [ D ]... ,
```

and after three iterations of analysis the string has the form:

```
C [ [ [ D ] D [ D ] ] D [ [ D ] D [ D ] ] ]... .
```

4.4 Synthesis

There are two synthesis phases. In phase zero, the productions strip away one layer of bracketing. In phase one, detail information is used to replace each coarse point by one of its two original fine points. The corresponding detail vector is replaced with the other fine point.

The production rule that strips away the start brackets is:

```
C(v) < SB() H() : {
  produce ;}
```

This production rule removes all starting brackets that are directly to the right of a point. Only the outer brackets are removed.

The production rule that strips away the end brackets is:

```
EB() > C(v) : {
  produce ;}
```

This production rule removes all end brackets that are to the left of a point. Again, only the outer brackets are removed.

For example, the string with the form:

```
C [ [ [ D ] D [ D ] ] D [ [ D ] D [ D ] ] ] C...
```

will have the form:

```
C [ [ D ] D [ D ] ] D [ [ D ] D [ D ] ] C...
```

after one iteration of phase zero. Notice that only the outer brackets have been removed. Now the leftmost `C` can "see" the fourth `D` from the left.

In phase one, subdivision is performed on the coarse points, `C`. Detail information stored in adjacent `D` vectors are added to the subdivided points to restore the original fine points. The two productions for this phase calculate the new location of a fine point using coefficients from the P and Q matrices given in equation 10.

This is the rule that replaces each coarse point with one of the restored fine points:

```
C(vll) D(vl) < C(v) > D(vr) C(vrr) : {
  produce C(0.125*vll + 0.75*v + 0.125*vrr
              + 0.25*vl + 0.25*vr);}
```

The fine point is a combination of a point created by subdivision: $0.125 * vll + 0.75 * v + 0.125 * vrr$, and detail information, $0.25 * vl + 0.25 * vr$.

This is the rule that replaces each module representing detail with a module representing a restored fine point

```
C(vl) < D(v) > C(vr): {
  produce C(0.5*vl + 0.5*vr + v);}
```

The fine point is a combination of a point created by subdivision, $0.5 * vl + 0.5 * vr$, and detail information, v.

After one iteration of phase one, the string has the form:

```
C [ [ D ] D [ D ] ] C [ [ D ] D [ D ] ] C... .
```

Once synthesis has been performed the same number of times analysis was performed, the original fine curve is restored.

The productions we discussed above deal with closed curves (we assume the required left and right context is always present). The complete listing of the L+C code for multiresolution representation of closed curves based on the above productions is given in Appendix A.

5 Extensions

5.1 Open Curves

We have also implemented multiresolution for open curves. The difference between the code for the open curve case and the closed curve case is that there are special endpoint rules for the open curve case. These special endpoint rules are included in Appendix B.

5.2 Aligning Detail with Normal

When we restore detail during synthesis, it has the same x-y orientation as the detail in the original curve. If the modified curve has a different slope than the original curve, the detail will look incorrect. To address this problem, Finkelstein and Salesin [finkelstein:multi] also introduced the idea of aligning the detail with the normal of the curve.

We approximate the normal, $\vec{N}$ at a given point, v, by finding the vector perpendicular to an approximated tangent vector, $\vec{T}$. We approximate $\vec{T}$ by taking the difference of the two points adjacent to v:

$$\vec{T} = v_r - v_l \tag{11}$$

During synthesis, instead of adding the detail directly back into the curve, we multiply the signed magnitude of the detail by the normal vector and add this aligned detail to the curve.

We have included the modified synthesis rules for aligning detail with the normal in Appendix C.

6 Results

Figure 7 shows a open curve which is coarsened, has its sweep modified, then is reconstructed. Figure 8 shows a branching structure with a modified sweep. Figures 9 shows a leaf with two different modified sweeps. Figure 10 compares the results of sweep modification of a curve (a). In 10 (b), the detail is not aligned with the normal. In 10 (c), the detail is aligned with the normal.

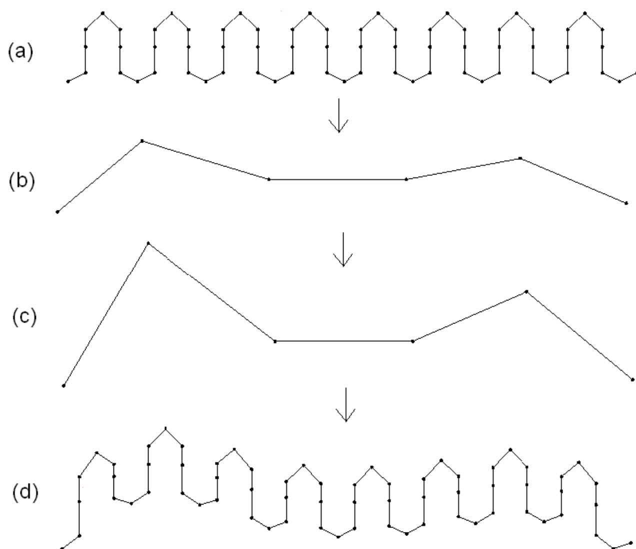

Figure 7: (a) Original curve. (b) Coarsened curve. (c) Modified coarse curve. (d) Reconstructed curve.

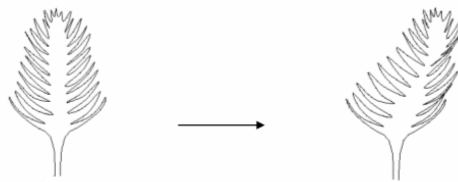

Figure 8: A branching structure with modified sweep.

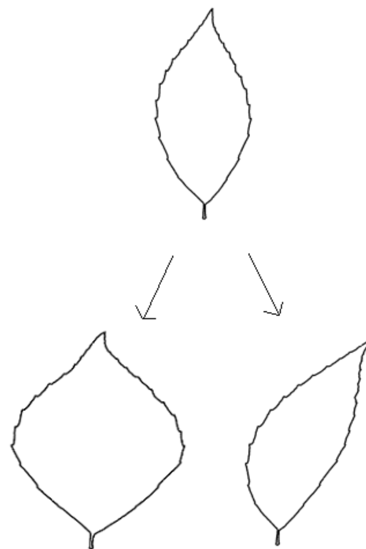

Figure 9: A scanned leaf with two modified sweeps.

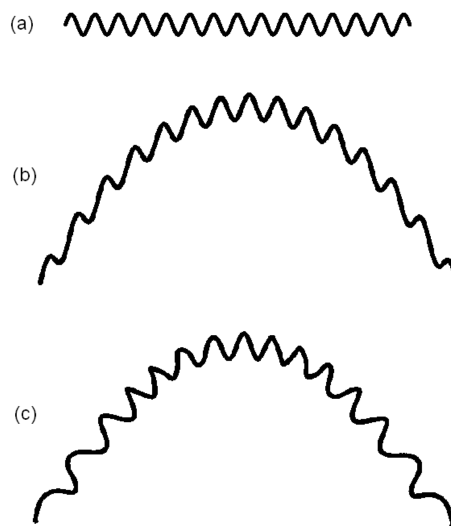

Figure 10: (a) Original curve. (b) Curve with modified sweep and detail not aligned with normal. (c) Curve with modified sweep and detail aligned with normal.

References

BARTELS, R. H., AND SAMAVATI, F. F. 2000. Reversing subdivision rules: Local linear conditions and observations on inner products. *Journal of Computational and Applied Mathematics 119*, 1–2, 29–67.

FINKELSTEIN, A., AND SALESIN, D. 1994. Multiresolution curves. In *Proceedings of SIGGRAAPH '94*, 261–268.

KARWOWSKI, R. 2002. *Improving the Process of Plant Modeling: The L+C Modeling Language*. PhD thesis, University of Calgary.

LINDENMAYER, A. 1968. Mathematical models for cellular interaction in development, Parts I and II. *Journal of Theoretical Biology 18*, 280–315.

PRUSINKIEWICZ, P., AND LINDENMAYER, A. 1990. *The Algorithmic Beauty of Plants*. Springer, New York.

PRUSINKIEWICZ, P., SAMAVATI, F., SMITH, C., AND KARWOWSKI, R. 2003. L-system description of subdivision curves. To appear in the International Journal of Shape Modeling.

ROZENBERG, G. 1973. T0L systems and languages. *Information and Control 23*, 357–381.

SAMAVATI, F. F., AND BARTELS, R. H. 1999. Multiresolution curve and surface representation by reversing subdivision rules. *Computer Graphics Forum 18*, 2, 97–120.

7 Appendix A - The L+C Code for Multiresolution B-spline Representation of Closed Curves

```
#include <lpfgall.h>

//Step at which to switch between coarsing and
refinement
#define SWITCH 8
#define NUMSTEPS 16

// The phase types
#define ANALYSIS_0 0
#define ANALYSIS_1 1
#define SYNTHESIS_0 2
#define SYNTHESIS_1 3

// String end marker
module E();
// An obstacle for context-matching purposes
module H();

// A point module
module C(V2f);
// Detail information module
D(V2f);
// A point that is about to be converted to detail
Dt(V2f);

int step;

Start: { step = 0;} StartEach: {
    // set the phase based on the step
    if(step < SWITCH)
        if(step%2==0) UseGroup(ANALYSIS_0);
        else UseGroup(ANALYSIS_1);
    else
        if(step%2==0) UseGroup(SYNTHESIS_0);
        else UseGroup(SYNTHESIS_1);
}

EndEach: { step++;}

derivation length: NUMSTEPS;

ring L-system: 1;

// axiom that defines initial curve goes here

/************************************************
 * Analysis (Reverse Subdivision)
 ***********************************************/

/////////////////////////////////////////////////
// ANALYSIS_0: change every other C into a D
/////////////////////////////////////////////////
group ANALYSIS_0:

C(vl) << C(v) : {
    produce Dt(v) ;
}

/////////////////////////////////////////////////
//ANALYSIS_1: Analysis (Perform reverse
//                       subdivision)
// (Store coarse points in C's and detail in D's)
/////////////////////////////////////////////////
group ANALYSIS_1:

// C rule
Dt(vl) < C(v) > Dt(vr): {
    produce EB() C(-0.5*vl + 2*v -0.5*vr) SB() H();
}

// D rule
Dt(vll) C(vl) < Dt(v) > C(vr) Dt(vrr) : {
    produce D(0.25*vll -1*vl + 1.5*v -
              1*vr + 0.25*vrr);
}

/************************************************
 * Synthesis (Subdivision)
 ***********************************************/

/////////////////////////////////////////////////
// SYNTHESIS_0: Eliminate outermost brackets
// [Branch] S [Branch] ]  --> [Branch] S [Branch]
/////////////////////////////////////////////////
group SYNTHESIS_0:

C(v) < SB() H() : {
    produce ;
}

EB() > C(v) : {
    produce ;
}

/////////////////////////////////////////////////
// SYNTHESIS_1: Synthesis (Perform subdivision)
//     (Use information from coarse points, C, and
//     details, D, to restore original points, C)
/////////////////////////////////////////////////
group SYNTHESIS_1:

// C rule
D(dl) C(vll) D(vl) < C(v) > D(vr) C(vrr) D(dr) : {
    produce C(0.125*vll + 0.75*v + 0.125*vrr
              + 0.25*vl + 0.25*vr);
}
```

```
// D rule
D(dl) C(vl) < D(v) > C(vr) D(dr): {
    produce C(0.5*vl + 0.5*vr + v);
}

endgroup

/***********************************************
 * Drawing
 ***********************************************/

interpretation:

C(v) : {
    produce SetColor(7) MoveTo2f(v) Circle(0.1) ;
}

Dt(v) : {
    produce SetColor(4) MoveTo2f(v) Circle(0.1) ;
}
```

8 Appendix B - Special Rules for Multiresolution B-spline Representation of Open Curves

```
/***********************************************
 * Analysis (Reverse Subdivision)
 ***********************************************

/* Left-most endpoint rules */

// C rules
E() < C(v) : {
    produce C(v);
}

E() C(vl) < C(v): {
    produce C(-1*vl + 2*v) SB() H();
}

// D rule

E() C(vll) C(vl) < Dt(v) > C(vr) Dt(vrr): {
    produce D(0.75*vll -1.5*vl + 1.125*v -
              0.5*vr + 0.125*vrr);
}

/* Right-most endpoint rules */

// C rules

C(v) > E(): {
    produce C(v);
}

C(v) > C(vr) E(): {
    produce EB() C(-1*vr + 2*v);
}

// D rule

Dt(vll) C(vl) < Dt(v) > C(vr) C(vrr) E(): {
    produce D(0.75*vrr -1.5*vr + 1.125*v -
              0.5*vl + 0.125*vll);
}

/***********************************************
 * Synthesis (Subdivision)
 ***********************************************/

/* Left Endpoint Rules */

// C rules

E() < C(v): {
    produce C(v);
}

E() C(vl) < C(v) : {
    produce C(0.5*vl + 0.5*v);
}

E() C(d) C(vll) D(vl) < C(v) > D(vr) C(vrr): {
    produce C(0.1875*vll + 0.6875*v + 0.125*vrr
              + 0.25*vl + 0.25*vr);
}

// D rule

E() C(d) C(vl) < D(v) > C(vr) : {
    produce C(0.75*vl + 0.25*vr + v);
}

/* Right Endpoint Rules */

// C rules

C(v) > E(): {
    produce C(v);
}

C(v) > C(vr) E(): {
    produce C(0.5*vr + 0.5*v);
}

C(vll) D(vl) < C(v) > D(vr) C(vrr) C(d) E(): {
    produce C(0.1875*vrr + 0.6875*v + 0.125*vll
              + 0.25*vr + 0.25*vl);
}

// D rule

C(vl) < D(v) > C(vr) C(d) E(): {
    produce C(0.75*vr + 0.25*vl + v);
}
```

9 Appendix C - Synthesis Rules for Alignment of Detail with Normal Vectors

```
// returns the length of vec
double vecLength(V2f vec) {
  double length = sqrt(vec.x*vec.x + vec.y*vec.y);
  return length;
}

// calculate a normal vector given a tangent vector
V2f normalFromTangent(V2f tangent) {
```

```
    // find the length of the tangent so we can
    // normalize
    double length = vecLength(tangent);

    V2f normal;
    if(length != 0){
        // the normal is the normalized tangent
        // rotated by pi/2
        normal.x = -(1.0/length)*tangent.y;
        normal.y = (1.0/length)*tangent.x;
    } else{
        normal.x = 0.0;
        normal.y = 1.0;
    }
    return normal;
}

// calculate the point with detail restored in the
// normal direction
 V2f getDetailAddedPoint(V2f point, V2f detail,
                       V2f vLeft, V2f vRight) {
    // approximate the tangent based on
    // neighboring points
    V2f tangent = vRight - 1*vLeft;

    // get the approximate normal vector
    V2f normal = normalFromTangent(tangent);

    // get the magnitude and direction of the
    // detail vector
    double length = vecLength(detail);
    if(detail.y < 0) length = -length;

    // produce the point
     return (point + length*normal);
}

//////////////////////////////////////////////////
// Phase 1: Synthesis (Perform subdivision)
// (Use information from coarse points, C, and
// details, D, to restore original points, C)
//////////////////////////////////////////////////
group SYNTHESIS_1:

/* Main Rules */

// C rule

D(dl)
    C(vll) D(vl) < C(v) > D(vr) C(vrr)
                                D(dr) : {
    V2f point = 0.125*vll + 0.75*v + 0.125*vrr;
    V2f detail = 0.25*vl + 0.25*vr;
    V2f dPoint =
        getDetailAddedPoint(point, detail, vll, vrr);
    produce C(dPoint);
}

// D rule

D(dl)
    C(vl) < D(v) > C(vr)
                D(dr): {
    V2f point = 0.5*vl + 0.5*vr;
    V2f detail = v;
    V2f dPoint =
        getDetailAddedPoint(point, detail, vl, vr);
    produce C(dPoint);
}
```