Advances in Multiresolution for Geometric Modelling,
N. A. Dodgson, M. S. Floater, M. A. Sabin (editors),
Springer, 2005, ISBN 3-540-21462-3, pp.283–271

Reverse Subdivision

Mohamed F. Hassan and Neil A. Dodgson

Computer Laboratory, University of Cambridge, UK
{mfh20|nad}@cl.cam.ac.uk

Summary. We present a reverse Chaikin algorithm which generates a multiresolution representation of any line chain. It has applications in multiresolution editing and compression. We also sketch how this might be extended to the bivariate Loop subdivision algorithm.

1 Introduction

Subdivision methods for curves were introduced and mathematically analysed for the first time by de Rham in 1947. Their re-invention in 1974 by Chaikin made them available to the computer graphics community. Chaikin used them to derive a simple algorithm for the high-speed generation of curves. He first proposed the binary 2-point approximating scheme in [1], which was shown to produce the quadratic B-spline in the limit [6].

Here we present an algorithm, based on Chaikin's algorithm, to decimate a polygonal curve so that, when the uniform subdivision scheme is applied to the decimated curve, a good approximation to the original curve is achieved. The errors between the reconstructed and original polygons are stored so that we can reconstruct the original polygon exactly. By continuing this process, we can construct a hierarchy giving a multiresolution representation of the original curve. Samavati and Bartels have previous explored reverse Chaikin [7]. Also B-spline wavelets [9] solve the same problem but require the construction of the high-frequency synthesis filter matrix, **Q**, and the solving of a sparse linear system to perform the reverse subdivision. Our approach is much simpler. The applications for this include lossy and lossless compression, multiresolution editing, and animation.

In Sects. 2–6 we present the Reverse Chaikin method; in Sect. 7 we show how this can be extended to the bivariate case, using Loop [5] as our subdivision scheme.

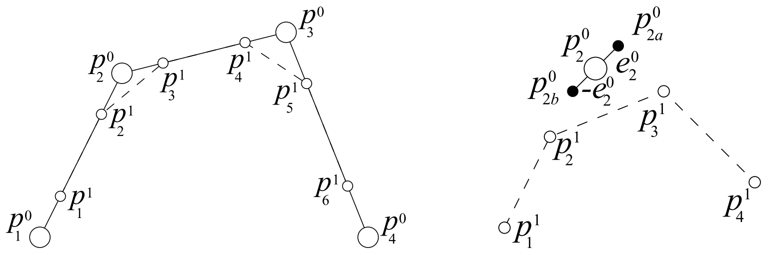

Fig. 1. Left: Chaikin's scheme – the large circles are points of the original polygon and the small circles are the points after one subdivision step. Right: Reverse Chaikin – the small circles are the original points and the large circle is the point after one reverse subdivision step. The filled circles are the two candidate points using the two different pairs of original points.

2 Reverse Chaikin

Chaikin's scheme can be defined formally by

$$\begin{aligned} p_{2i-1}^{n+1} &= \tfrac{3}{4}p_i^n + \tfrac{1}{4}p_{i+1}^n, \\ p_{2i}^{n+1} &= \tfrac{1}{4}p_i^n + \tfrac{3}{4}p_{i+1}^n, \end{aligned} \tag{1}$$

where p_m^n is the position in $\mathbb{R}^3$ of point m after n subdivision steps (see Fig. 1(left)).

We would like to reverse this process. Now, solving (1) for p_i^n gives

$$p_i^n = \frac{3}{2}p_{2i-1}^{n+1} - \frac{1}{2}p_{2i}^{n+1}. \tag{2}$$

However, solving (1) for p_{i+1}^n and re-indexing gives

$$p_i^n = \frac{3}{2}p_{2i-2}^{n+1} - \frac{1}{2}p_{2i-3}^{n+1}. \tag{3}$$

Geometrically, this means that there are two sets of pairs of points in the subdivided polygon that can be used to calculate the position of a single point in the original. If Chaikin's scheme was used to produce the refined polygon there is no problem, as the two positions will coincide. However, this cannot be guaranteed in the general case. Our solution is to take the average of the two positions and store the error vectors (see Fig. 1(right)).

Formally this gives

$$p_i^n = -\frac{1}{4}p_{2i-3}^{n+1} + \frac{3}{4}p_{2i-2}^{n+1} + \frac{3}{4}p_{2i-1}^{n+1} - \frac{1}{4}p_{2i}^{n+1}, \tag{4}$$

$$e_i^n = \frac{1}{4}p_{2i-3}^{n+1} - \frac{3}{4}p_{2i-2}^{n+1} + \frac{3}{4}p_{2i-1}^{n+1} - \frac{1}{4}p_{2i}^{n+1}. \tag{5}$$

The forward Chaikin step then becomes:

$$\begin{aligned} p_{2i-1}^{n+1} &= \tfrac{3}{4}(p_i^n + e_i^n) + \tfrac{1}{4}(p_{i+1}^n - e_{i+1}^n), \\ p_{2i}^{n+1} &= \tfrac{1}{4}(p_i^n + e_i^n) + \tfrac{3}{4}(p_{i+1}^n - e_{i+1}^n). \end{aligned} \tag{6}$$

2.1 Wavelet Formulation

At first sight the formulation above may appear not to be a wavelet transformation because the errors, e_i^n, are added to the points, p_i^n, *before* the subdivision step rather than after, as would be the case with a wavelet transformation. However, some straightforward algebraic manipulation converts it to a wavelet formulation.

If we write $\mathbf{p}^{n-1} = \mathbf{A}^n\mathbf{p}^n$ and $\mathbf{e}^{n-1} = \mathbf{B}^n\mathbf{p}^n$, where $\mathbf{p}^n = [p_0^n\ p_1^n\ p_2^n \ldots]^T$ (and similarly for $\mathbf{e}^n$), then, from (4) and (5), the wavelet analysis filters, $\mathbf{A}^n$ and $\mathbf{B}^n$ are:

$$\mathbf{A}^n = \frac{1}{4}\begin{bmatrix} \ddots & & & & & & & & & \\ & -1 & +3 & +3 & -1 & & & & & \\ & & & -1 & +3 & +3 & -1 & & & \\ & & & & & -1 & +3 & +3 & -1 & \\ & & & & & & & & & \ddots \end{bmatrix}$$

$$\mathbf{B}^n = \frac{1}{4}\begin{bmatrix} \ddots & & & & & & & & & \\ & +1 & -3 & +3 & -1 & & & & & \\ & & & +1 & -3 & +3 & -1 & & & \\ & & & & & +1 & -3 & +3 & -1 & \\ & & & & & & & & & \ddots \end{bmatrix}$$

Similarly, if we write the synthesis step as $\mathbf{p}^n = \mathbf{P}^n\mathbf{p}^{n-1} + \mathbf{Q}^n\mathbf{e}^{n-1}$ then the wavelet synthesis filters, $\mathbf{P}^n$ and $\mathbf{Q}^n$ are derived from (6) as:

$$\mathbf{P}^n = \frac{1}{4}\begin{bmatrix} \ddots & \vdots & & & & \\ & 3 & 1 & & & \\ & 1 & 3 & & & \\ & & 3 & 1 & & \\ & & 1 & 3 & & \\ & & & 3 & 1 & \\ & & & 1 & 3 & \\ & & & & \vdots & \ddots \end{bmatrix} \qquad \mathbf{Q}^n = \frac{1}{4}\begin{bmatrix} \ddots & \vdots & & & & \\ & +3 & -1 & & & \\ & +1 & -3 & & & \\ & & +3 & -1 & & \\ & & +1 & -3 & & \\ & & & +3 & -1 & \\ & & & +1 & -3 & \\ & & & & \vdots & \ddots \end{bmatrix}$$

The advantage of this reverse Chaikin method over many wavelet methods is computation speed: both the analysis and the synthesis filters are *sparse*, whereas in many wavelet methods (e.g. the B-spline wavelets [9]) the analysis filters, $\mathbf{A}^n$ and $\mathbf{B}^n$, are dense, thus requiring quadratic rather than linear time to perform the analysis (reverse subdivision) step naïvely, or else requiring the solution of a linear system [9].

In addition, there are a number of subtleties in the practical implementation of reverse subdivision which make the wavelet formulation difficult, but which can be handled in the algorithmic formulation by simple modifications to the algorithm, as described in the following section.

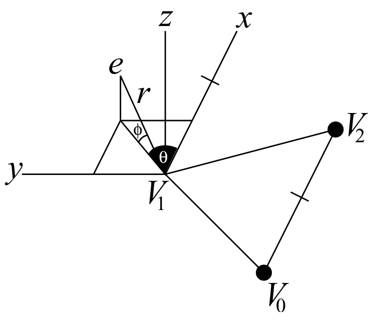

Fig. 2. Local spherical coordinate frame. The local cartesian frame for the error at V_1 is defined by the reverse subdivided points V_0, V_1, and V_2: The x basis vector is parallel to $V_2 - V_0$. The z basis vector is perpendicular to the plane defined by $\{V_0, V_1, V_2\}$, and the y basis vector is given by $z \times x$. r and θ are the standard spherical coordinates, ϕ is angle from the xy-plane.

3 Subtleties

3.1 Local Coordinate Frame for Errors

If the error vectors are stored naïvely, one of the primary uses of this algorithm is lost. The reason for this is that for editing purposes we would like to edit the reverse subdivided curve and have the changes manifest in the detailed curve, without changing the error vectors. We can see immediately that if we apply a similarity transformation to the reverse subdivided curve, we will have to apply the same transformation to the error vectors in order to get the transformed version of the original curve, which is undesirable. Our solution to this problem is to use a local coordinate frame for the error vectors (see Fig. 2).

The local cartesian frame for the error at a reverse subdivided point is defined by that point and its immediate neighbours. The x unit basis vector is chosen to be parallel to the vector between the two neighbouring points. The z unit basis vector is perpendicular to the plane defined by these three points, and the y basis vector is given by $z \times x$. r and θ are the standard spherical coordinates, ϕ is the angle from the xy-plane. The x vector and the definition for ϕ were chosen such that θ and ϕ would be close to 0. Finally r can be stored as a fraction of the magnitude of the vector between the neighbouring points. A similar co-ordinate system is used by Stollnitz et al. [9].

Now, if we apply a similarity transformation to the reverse subdivided points and subdivide them using the same stored errors, we will get the same transformation of the original polygon. An example is shown in Fig. 3.

3.2 Odd Vertex Number

There is another subtlety. Suppose we have n vertices. After one Chaikin subdivision step we will have either $2n$ vertices, for closed loops, or $2n-2$ for

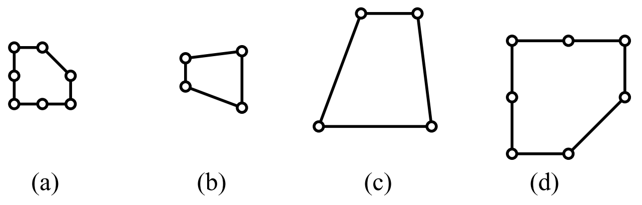

Fig. 3. An example of a similarity transformation. (**a**) The original outline. (**b**) The reverse subdivision of (**a**). (**c**) A similarity transformation of (**b**). The subdivision of (**c**) using the same errors as the original reverse subdivision. The circles indicate the position of the vertices. To perform reverse subdivision the centre vertex on the left hand side of (**a**) must be duplicated to give an even number.

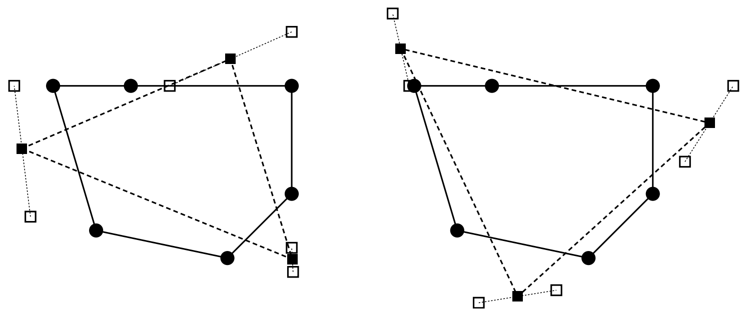

Fig. 4. One step of reverse Chaikin can produce two possible solutions. In this example, a six-vertex polygon (*solid line*) is reduced to two alternative triangles (*heavy dashed line*). The solid circles are the original vertices; the positions of the computed vertices are open squares, these are averaged to give the final position, which is shown by a solid square.

open line chains. This means that Reverse Chaikin has a problem if we have an odd number of vertices. This problem is simply overcome by duplicating the last vertex and tagging that we have done so in the error file. See Fig. 3, where the central vertex on the left hand side is duplicated.

3.3 Alternative Configurations

A problem, which is best illustrated on closed loops, is that there are multiple possible simplifications. Fig. 4 shows an example of a closed loop of six vertices and the two possible three-vertex polygons which can be produced by the reverse Chaikin algorithm. A choice between the two alternatives must be made at each level of reverse subdivision, leading to 2^k possible decimated line chains after k steps of reverse subdivision. This is not ideal, but it cannot be avoided. This is handled by generating both possible decimated line chains at each step of reverse subdivision, and then picking the one with the smallest error vectors. This is especially important for line chains which were originally generated by Chaikin subdivision, as the correct version will have zero error vectors, while the alternative version will, in general, have non-zero error vectors.

This problem can also manifest in an open chain with an odd number of vertices, as either end can be considered to be the start of the chain, with the other end having the duplicated vertex mentioned in Sect. 3.2.

This problem is specific to the univariate case. In the bivariate case, the extraordinary vertices fix the topology of the reverse subdivided mesh, although they bring in their own problems, described in Sect. 8.

4 Examples

An example of a similarity transformation is given in Fig. 3. The original line chain is reverse subdivided, transformed, and then subdivided using the same errors as the original reverse subdivision. We can see that, after this procedure, the final line chain is the similarity transformation of the original.

An example of local editing can be seen in Fig. 5. The left side, top to bottom, is the reverse subdivided outline of England (585 vertices to 37 vertices). The right side, bottom to top, is the modified outline, subdivided using the same errors as the left side. We can see that we achieve a local editing of the original curve without loss of detail.

5 Examining Stability

It is important to examine the behaviour of the algorithm when applied multiple times; as the algorithm relies on local extrapolation, errors could tend to be magnified. Hence the distance to the original curve may grow rapidly causing it to become unstable.

To examine the stability of our algorithm qualitatively, we perform the following experiment: we construct a subdivision curve from a coarse line chain by applying Chaikin's scheme. After a fixed number of subdivisions we add white noise to each vertex. Then we smooth the curve by removing the error coefficients at the finest levels.

The results of the noise-removal experiment can be seen in Fig. 6. We see that the algorithm slowly reduces the amplitude of the noise over a number of smoothing steps and hence is stable for practical applications. This also means that the algorithm can be used for lossy compression as described in the next section.

6 Storage and Compression

The reverse Chaikin algorithm presented in Sect. 2 is suitable for multiresolution editing, provided that the error vectors are stored in the format described

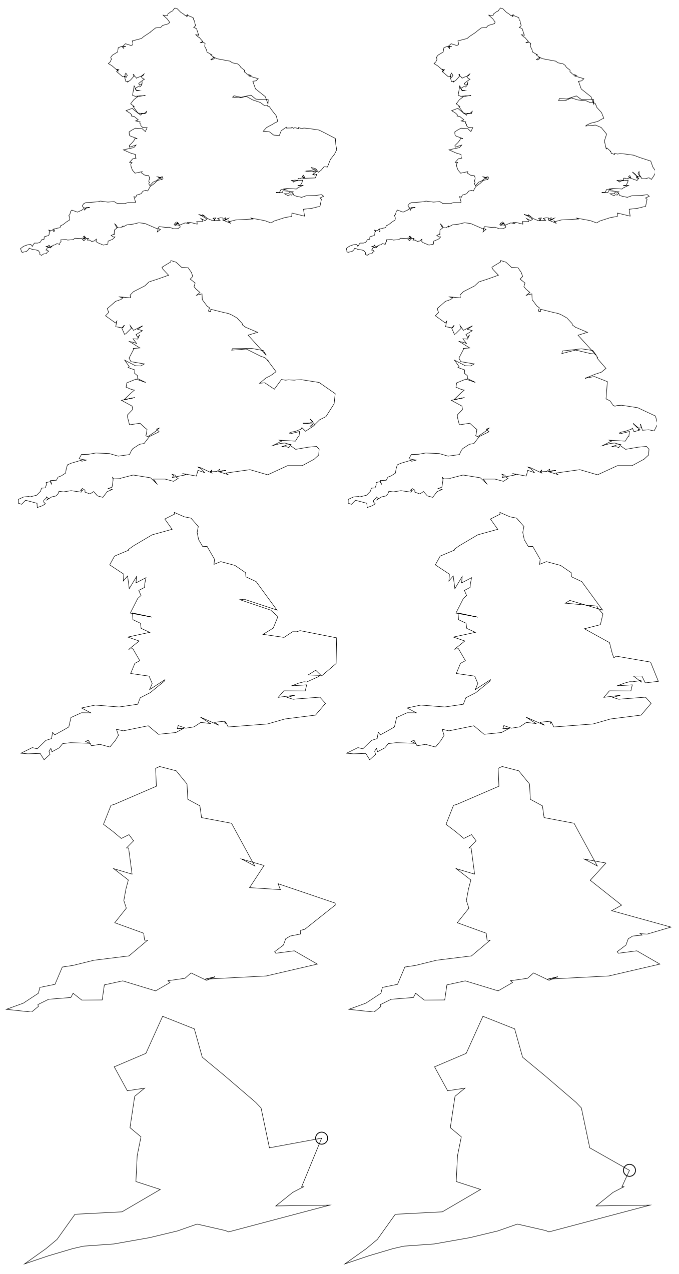

Fig. 5. Example: Local editing. The left side, top to bottom, is the reverse subdivided outline of England (585 vertices to 37 vertices). The right side, bottom to top, is the modified outline, reconstructed using the same errors as the left side. The circle indicates the modified vertex.

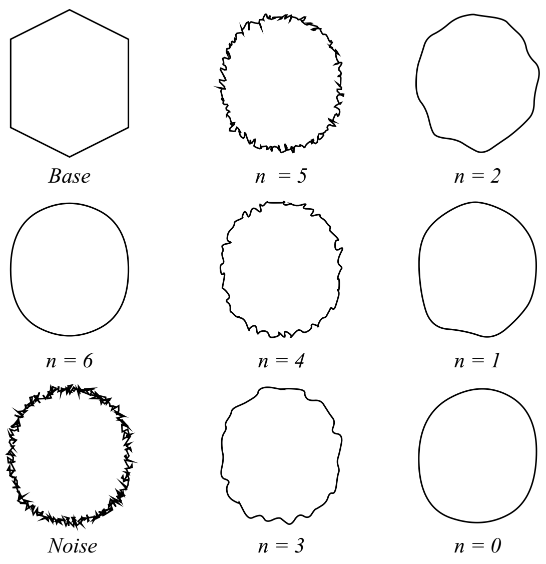

Fig. 6. Noise-removal experiment: Top-left shows the base curve. Middle-left shows the base curve subdivided 6 times (384 vertices). Bottom-left shows the curve with added noise – the average magnitude of the noise is 3 times the average distance between vertices. Middle and right show the results after the error coefficients at n and above steps have been set to zero.

in Sect. 3. The assumption is that vertex coordinates and error vectors are stored to the floating-point precision of the computer. The reverse Chaikin representation thus uses the same amount of storage space as the original set of vertices. Indeed, it can be seen as a simple wavelet transform (Sect. 2.1): transforming one set of floating-point coordinates into another, both sets containing the same number of values.

For lossless compression, it is necessary to store the reverse Chaikin error vectors in less space than is required to store vertex coordinates. The local coordinate system introduced in Sect. 3 allows us to compress error vectors more efficiently than vertex coordinates because we can expect the r, θ, and ϕ coordinate values to be close to zero with high probability. Such a distribution of coordinate values allows for efficient coding, as shorter codes can be used for the more likely values. In particular, we note that ϕ should be very close to zero in most cases, because we can expect vertices at level $(n+1)$ to lie very close to the plane through the three nearest vertices at level n. By similar reasoning θ should be close to zero.

Lossy compression can be achieved in two complementary ways. Firstly, the quantisation of vertex coordinates and error vectors can be made increasingly coarse. There is obviously some loss of resolution in the position of vertices, but the more important disadvantage is that errors in position will accumulate as coarsely-quantised error vectors are added to intermediate-level vertices which have themselves been calculated using error vectors: the absolute error in position increasing as the subdivision level increases.

The second way of achieving lossy compression is simply to throw away error vectors. For example, throwing away the error vectors at just the highest subdivision level will remove (almost) half of the data. This means that the highest level is now approximated by the uncorrected Chaikin subdivision of the next highest level. If the original line chain is of sufficiently high resolution, then a reasonable approximation to the original may be achieved by the uncorrected Chaikin subdivision generated from several levels of reverse Chaikin without any error vectors being stored. This significantly reduces the storage required for the line chain. A related compression technique for line chains, with similar motivation, is the Douglas–Pücker algorithm [2], which replaces a line chain by a coarser line chain which approximates the original to some tolerance but which does not allow for multiresolution editing.

7 Theory of the Bivariate Case

This section provides the theory to extend these ideas to the bivariate case. Samavati et al. [8] have produced an algorithm to generate multiresolution surfaces by a reverse Doo subdivision method using a local least-squares method. Khodakovsky et al. [4] use a wavelet transform based on Loop subdivision [5] for geometry compression. As with all wavelet methods a high-pass reconstruction filter **Q** must be constructed, and also there is no theory for wavelet constructions around irregular vertices so this is done ad hoc. Finally computing the forward wavelet transform requires solving linear systems. Our method is simpler and is also based on Loop subdivision.

First, we have to define the constraints of the starting mesh. Here we present an algorithm which works on triangular meshes with binary subdivision connectivity. This is not such a heavy constraint as there are many algorithms that will transform an arbitrary mesh to one with subdivision connectivity (e.g. [10]). Our algorithm naturally splits into two parts.

The first part is to detect whether the given mesh has subdivision connectivity and construct the connectivity of the coarse level mesh. Two algorithms achieve this. Taubin's algorithm [11] is based on the concept of the *covering surface* in algebraic topology. The algorithm proposed by Hormann in [3] is simpler and uses the observation that most of the vertices of a mesh with subdivision connectivity are regular and all irregular vertices are guaranteed to be vertices of the coarse level mesh. Both these algorithms give a set of candidate vertices for the coarse level to which we can apply the next stage of the algorithm.

The next stage uses the inverse of the subdivision matrix of the given uniform scheme in order to construct the geometry of the coarse level vertices. There are a number of practical considerations in the implementation. One interesting issue is that the extraordinary vertices must be preserved between

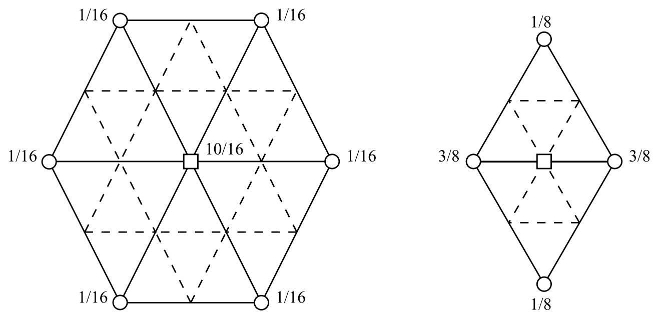

Fig. 7. Stencils for the Loop subdivision scheme for regular vertices. The square indicates the vertex being calculated.

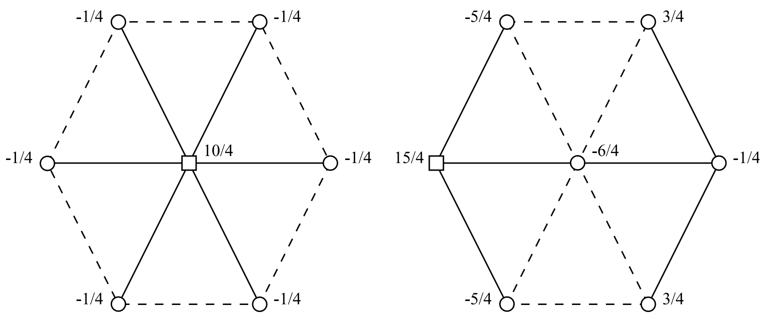

Fig. 8. Coefficients for the reverse subdivision scheme for regular vertices. Left: the stencil for one set of points: calculating the position of the central vertex. Right: the stencil for an alternative set of points: calculating the position of one of the non-central vertices. In both cases, the square indicates the vertex being calculated.

subdivision levels. This limits the form of a reverse subdivided mesh and may mean that, in some cases, there is no possible valid mesh which can be created at a coarser level. There are various potential solutions to this, including splitting the mesh into sub-meshes, which are then stitched together in some way.

7.1 Geometry of Regular Vertices

Fig. 7 shows the stencil for the standard Loop subdivision scheme in the regular case. We use this to form the subdivision matrix around a vertex. The inverse of this matrix gives us the coefficients for the reverse subdivision scheme (shown in Fig. 8(left)).

However the inverse subdivision matrix also shows that we can use other points to define the same vertex (see Fig. 8(right)). This is analogous to the univariate case. In this case there are seven candidate vertices (see Fig. 9), rather than the two in the univariate case. So now we take the Loop subdivision of the seven cases to be our actual vertex and store the errors. Again,

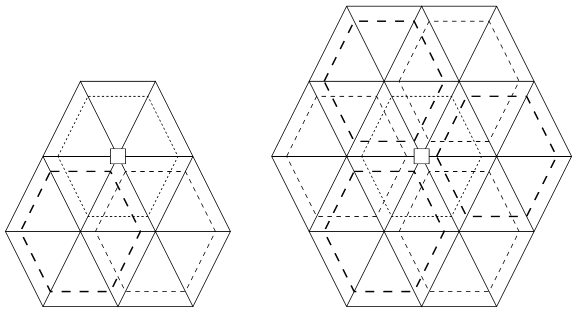

Fig. 9. Each dashed hexagon represents a set of vertices that can be used to calculate the central reverse subdivided vertex (indicated by the square). At left are three (out of the seven) sets of vertices and at right all seven are shown.

and for the same reasons, we use a local coordinate frame similar to that used in the univariate scheme.

7.2 Geometry of Extraordinary Vertices

Fig. 10(a) shows the Loop subdivision scheme for n-valent vertices. This time we cannot simply take the inverse of the subdivision matrix to calculate the coefficients of the reverse subdivision scheme as the values and size of the matrix are dependent on n, the valency. Instead we formulate the linear system as follows:

$$p'_0 = (1 - a_n)p_0 + \sum_{i=1}^{n} \frac{a_n}{n} p_i \tag{7}$$

$$p'_i = \frac{3}{8}(p_0 + p_i) + \frac{1}{8}\left(p_{i-1(n)} + p_{i+1(n)}\right) \tag{8}$$

where p_0 is the extraordinary vertex and $\{p_i | i = 1, \ldots, n\}$ are its immediate neighbours.

Solving this system for p_0 gives

$$p_0 = (1 - b_n)p'_0 + \sum_{i=1}^{n} \frac{b_n}{n} p'_i \tag{9}$$

where

$$b_n = 1 + \frac{5}{8a_n - 5} \tag{10}$$

This is independent of the actual a_n used. Hence the reverse subdivided scheme is given by Fig. 10(b).

As with the regular case, these points can also be used to calculate the neighbouring points (see Fig. 10(c)). The coefficients are given by $\{c_n^i | i = 0, \ldots, n\}$, where n is the valency. Unfortunately we have not found a closed

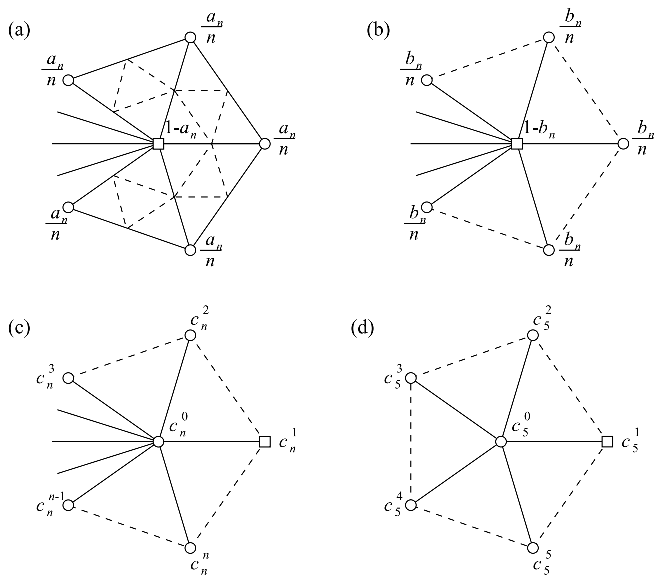

Fig. 10. Extraordinary vertices. (**a**) shows the standard Loop subdivision at extraordinary vertices, where usually $a_n = \frac{1}{64}\left(40 - \left(3 + 2\cos\frac{2\pi}{n}\right)^2\right)$. (**b**) shows the reverse Loop subdivision at extraordinary vertices, where $b_n = 1 + \frac{5}{8a_n-5}$ (independent of the expression for a_n). (**c**) shows the alternative stencil for the general n-valent vertex. (**d**) shows the alternative stencil for a 5-valent vertex. The square indicates the vertex being calculated.

form for the $\{c_n^i\}$ so the inverse subdivision matrix has to be calculated for each valency.

However, we find that $c_n^0 = \frac{3}{8a_n-5}$ for all n and we can see by symmetry that $c_n^{n-i} = c_n^{i+2}$, $i = 0, \ldots, n-2$. The same mechanism is used as in the regular case to handle the candidate vertices.

8 Future Work

We have yet to implement and test the bivariate algorithm given in the previous section, and this is the obvious next step. We will use Hormann's algorithm [3] as the basis for the topology step. We expect the bivariate case to find uses in the same applications as the univariate case: multiresolution editing and compression.

Acknowledgement

This work was supported in part by the European Union research project "Multiresolution in Geometric Modelling (MINGLE)" under grant HPRN–CT–1999–00117.

References

1. G.M. Chaikin. An algorithm for high-speed curve generation. *Computer Graphics and Image Processing*, 3:346–349, 1974.
2. D. Douglas and T. Pücker. Algorithms for the reduction of the number of points required to represent a digitised line or its caricature. *The Canadian Cartographer*, 10:112–122, 1973.
3. K. Hormann. An easy way of detecting subdivision connectivity in a triangle mesh. Technical Report 3, Department of Computer Science 9, University of Erlangen, May 2002.
4. Andrei Khodakovsky, Peter Schröder, and Wim Sweldens. Progressive geometry compression. In *Proc. ACM SIGGRAPH 2000*, pages 271–278, 2000.
5. C. T. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah, Department of Mathematics, 1987.
6. R. F. Riesenfeld. On Chaikin's algorithm. *Computer Graphics and Image Processing*, 4:304–310, 1975.
7. F. Samavati and R. Bartels. Multiresolution curve and surface representation by reversing subdivision rules. *Computer Graphics Forum*, 18(2):97–119, 6 1999.
8. F. Samavati, N. Mahdavi-Amiri, and R. Bartels. Multiresolution surfaces having arbitrary topologies by a reverse Doo subdivision method. *Computer Graphics Forum*, 21(2):97–119, 6 2002.
9. E. J. Stollnitz, T. D. DeRose, and D. H. Salesin. *Wavelets for computer graphics.* San Francisco; Morgan Kaufmann Publishers, 1996.
10. Vitaly Surazhsky and Craig Gotsman. Explicit surface remeshing. In *Proc. Eurographics Symposium on Geometry Processing*, pages 17–28, June 2003.
11. G. Taubin. Detecting and reconstructing subdivision connectivity. *Visual Computer*, 2002.