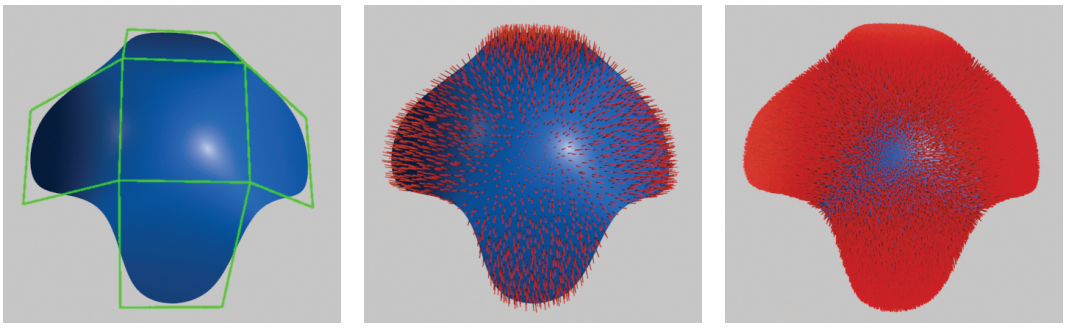

(a) (b) (c)

Figure 1: (a) Control mesh (green) with 12 boundary EVs, and limit surface (blue). (b) 20x20 and (c) 70x70 hairs per face, evaluated at jittered random (u, v) locations using our method.

Exact Evaluation of Catmull-Clark Subdivision Surfaces Near B-spline Boundaries

Dylan Lacewell[1,2] Brent Burley[1]

[1] *Walt Disney Animation Studios*
[2] *School of Computing, University of Utah*

Abstract

We extend the eigenbasis method of Jos Stam to evaluate Catmull-Clark subdivision surfaces near extraordinary vertices on B-spline boundaries. Source code to generate eigenbasis data and do the runtime evaluation is available online.

1 Introduction

In a seminal paper [5], Jos Stam gave a method for evaluating Catmull-Clark subdivision surfaces [1] at parameter values near an interior extraordinary vertex (EV). The basic idea is to subdivide recursively until the (u, v) parameter to be evaluated is contained in a regular 4×4 grid of control points which define a bicubic B-spline patch. The subdivision steps can be computed very efficiently in the eigenbasis of the subdivision matrix, i.e., by diagonalizing. Stam did not provide details for boundaries, where diagonalization is not always possible, but in related work on Loop subdivision, he used the Jordan Normal Form to evaluate EVs of valence 3 [4]. Evaluation near boundaries was evidently implemented in Maya for Catmull-Clark surfaces but never published. Zorin et al. later published a method for evaluating a family of Loop subdivision schemes, including near boundaries, using a different decomposition [7]. They worked on a similar method for Catmull-Clark but did not publish it.

We first encountered the problem of evaluation near boundaries while working on a hair system for production rendering. We needed to grow hair at arbitrary (u, v) locations on a control cage, and it was important that the hair origin lie on the limit surface when rendered. Artists were at first encouraged to avoid growing hair near boundaries of open cages, but inevitably the demand for this capability arose, e.g., for eyebrows and some hair styles.

It might be possible in some cases to subdivide to a high level and bilinearly interpolate over each face, instead of using exact evaluation. However, the subdivide-and-interpolate approach requires memory for the high resolution mesh, whereas our exact evaluation only requires memory for the base mesh, plus a constant amount of static eigenbasis data. Bilinear interpolation introduces errors which are dependent on the mesh resolution, and although hair is perhaps forgiving of errors, applications involving texture and displacement maps are less so. For example, we have successfully used exact evaluation to bake vector displacement maps between two subdivision surfaces with a common structure (e.g., where one surface is a subdivided and sculpted version of the other). Our method could also be used to project textures onto subdivision surfaces, and to compute curvature.

Our studio uses Catmull-Clark subdivision with cubic B-spline boundaries similar to [2], but without any pinned vertices; *all* new boundary edge points are the average of the two adjacent boundary vertices. The subdivision matrix is not always diagonalizable for boundary EVs of valence not equal to 3. In practice, most boundary EVs are convex or concave corners of valence 2 or 4, but we wanted a method that would work for higher valences as well should the need arise. Our solution was to extend Stam's method using the Jordan Normal Form. We assume in the remainder of this paper that the reader is familiar with Stam's paper, as we use similar notation.

faceTileIds[i] = 0; // single tile for entire mesh

2 Algorithm

In the regular boundary case shown in Figure 2, we extrapolate an extra row of control points past the boundary and evaluate the B-spline patch $s(u, v)$ directly. The extrapolated points cause the boundary $s(u, 0)$ to be a cubic B-spline curve.

However, direct evaluation is not possible when there are EVs. Figure 3 (left) shows control

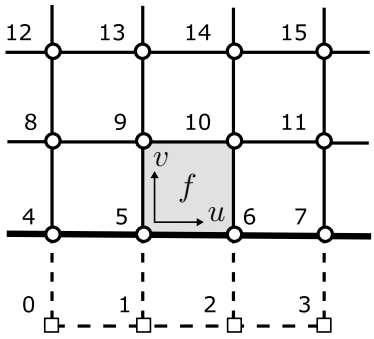

Figure 2: A regular boundary face, f, is evaluated as a bi-cubic B-spline patch: $s(u, v) = \sum_{i=0}^{15} p_i b_i(u, v)$. Control points 0 through 3 are extrapolated, e.g., $p_0 = 2p_4 - p_8$.

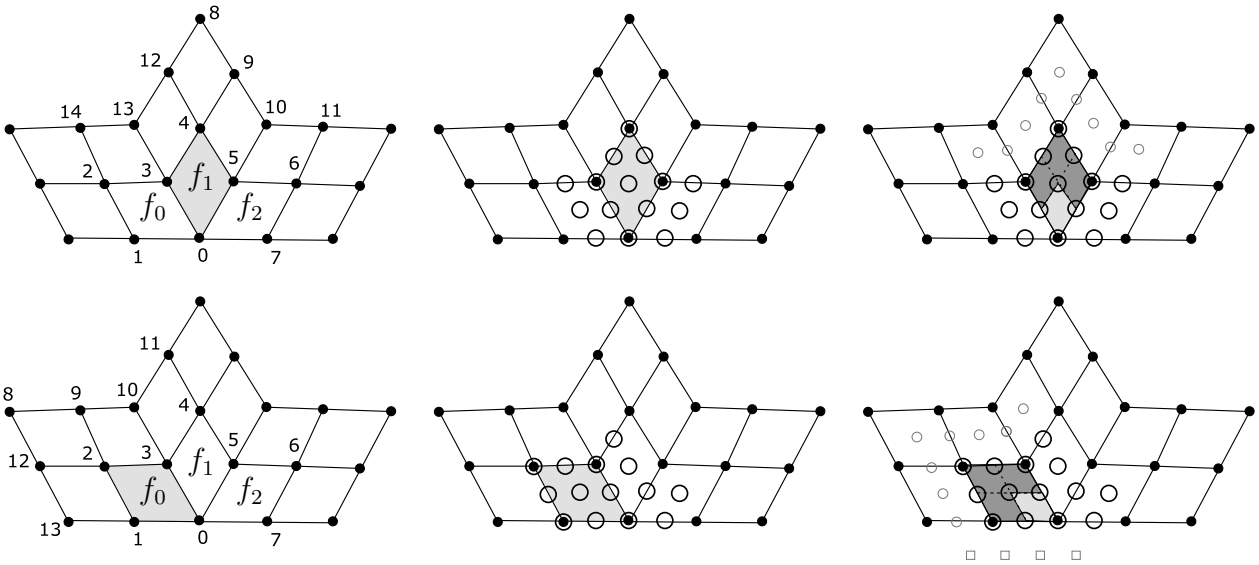

Figure 3: Step-by-step evaluation of face f_1 (top row) and face f_0 (bottom row), both of which contain a boundary EV of valence 4 (marked as vertex 0). The initial control points (left) are subdivided (middle), then extra control points are created by subdivision or linear extrapolation (right), giving enough points to perform B-spline evaluation on the dark shaded regions. Note that extrapolated points are marked as squares (lower right). To evaluate more of the face, the middle step can be repeated recursively via repeated multiplication by a subdivision matrix.

points near a boundary EV of valence $N = 4$. There are $K = 2N + 6$ control points for face f_0 and $K = 2N + 7$ points for face f_1. The $K \times K$ subdivision matrix has the form

$$\mathbf{A} = \begin{pmatrix} \mathbf{S} & \mathbf{0} \\ \mathbf{S}_{11} & \mathbf{S}_{12} \end{pmatrix}$$

To quickly sketch Stam's method, multiplying a vector $\mathbf{C}_0$ of the control points by $\mathbf{A}$ gives us a new vector $\mathbf{C}_1$ of points closer to the EV (Figure 3, center). Moreover, the new control points define three bicubic B-spline patches which can be explicitly evaluated. Any (u, v) of interest can be evaluated by subdividing n times until we have a supporting set of control points $\mathbf{C}_n$. An extra layer of control points (Figure 3, right) is also needed, so the final multiplication is by $\bar{\mathbf{A}}$ which contains extra B-spline knot insertion rows:

$$\bar{\mathbf{C}}_n = \bar{\mathbf{A}}\mathbf{A}^{n-1}\mathbf{C}_0$$

We select the 16 B-spline control points by multiplying $\bar{\mathbf{C}}_n$ with one of three picking matrices, $\mathbf{P}_k$, logically similar to those shown in Figure 6 of Stam. One difference is that the picking matrix may also need to extrapolate points across the boundary, such as the 4 points marked by small squares in Figure 3, lower right.

For Stam's method to be practical, the computation of $\mathbf{A}^{n-1}$ needs to be very fast. In the interior, Stam diagonalizes $\mathbf{A}$ using an eigenvector decomposition, so that computing $\mathbf{A}^{n-1}$ takes constant time:

$$\mathbf{A}^{n-1} = \mathbf{V}\Lambda^{n-1}\mathbf{V}^{-1} = \mathbf{V}\texttt{diag}(\lambda_0^{n-1}, \lambda_1^{n-1}, ..., \lambda_{K-1}^{n-1})\mathbf{V}^{-1}$$

where $\mathbf{V}$ has eigenvectors in the columns and Λ is a diagonal matrix of eigenvalues.

However, for boundary EVs of most valences, $\mathbf{A}$ is not diagonalizable; there is one eigenvalue of algebraic multiplicity 2 and geometric multiplicity 1 (meaning the eigenvalue occurs twice, but there is only one corresponding eigenvector up to normalization). We turn instead to the Jordan Normal Form (JNF) of $\mathbf{A}$:

$$\mathbf{A}^{n-1} = \mathbf{T}\mathbf{J}^{n-1}\mathbf{T}^{-1}$$

where $\mathbf{T}$ has $K-1$ eigenvectors and one so-called 'generalized' eigenvector in the columns, and $\mathbf{J}$ is nearly diagonal, so that $\mathbf{J}^{n-1}$ has only one extra off-diagonal term. We show below how to find the generalized eigenvector for any valence. See for example [6] for theoretical background on the JNF.

There are a few other complications with boundaries. Portions of matrices $\mathbf{A}$ and $\mathbf{T}$ depend on the face index about the EV, although some blocks are constant. Because of symmetry there are only $\lfloor N/2 \rfloor$ unique cases, e.g., faces f_0 and f_2 in Figure 3 are symmetric and have the same subdivision matrix provided the control points are reordered. Face f_0, which has a boundary edge, is a special case because there is one fewer initial control point, and 4 of the final control points are extrapolated across the boundary (Figure 3, lower right). Corners with valence 2 (not shown) have $2N+5$ initial control points and need 8 extrapolated points.

3 Implementation Details

N	4	5	6	7	8	9	10	$\cdots$	$4i$	$4i+1$	$4i+2$
index(r)	4	2	6		8	4	10	$\cdots$	$4i$	$2i$	$4i+2$
λ_r	0.25	0.5	0.25		0.25	0.5	0.25	$\cdots$	0.25	0.5	0.25

Figure 4: Indices and values of the 'defective' eigenvalue of $\mathbf{A}$, depending on valence, N. The indices are zero-based, with eigenvalues sorted in decreasing order for each valence.

3.1 Precomputation

In this section we compute the JNF of $\mathbf{A}$ numerically, by first finding the JNF of the upper left block:

$$\mathbf{S} = \mathbf{U}_0 \Sigma \mathbf{U}_0^{-1}$$

(The lower right block, $\mathbf{S}_{12} = \mathbf{W}_1 \Delta \mathbf{W}_1^{-1}$ is diagonalizable and is the same as in Stam except when $N = 2$). To compute $(\mathbf{U}_0, \Sigma)$ we first compute $\texttt{eig}(\mathbf{S})$ numerically in Matlab. We observe empirically that for valences $3, 7, 11, \ldots$, $\mathbf{S}$ is diagonalizable, and for other valences there is one 'defective' eigenvalue, λ_r, that occurs twice but has only one eigenvector. If the eigenvalues for a given valence are sorted in decreasing order, the index r and value of λ_r occur in the pattern shown in Figure 4. (We have verified this pattern through valence $N = 100$). We next find a generalized eigenvector, $\mathbf{w}$ to go with the existing eigenvector, $\mathbf{v}_r$, such that

$$(\mathbf{S} - \lambda_r \mathbf{I})\mathbf{w} = \mathbf{v}_r$$

There are many choices for $\mathbf{w}$, since adding any multiple of $\mathbf{v}_r$ onto a solution gives another solution. The $\mathbf{w}$ that is orthogonal to $\mathbf{v}_r$ is the shortest solution, and we find it using the pseudoinverse (`pinv()` in Matlab):

$$\mathbf{w} = \texttt{pinv}(\mathbf{S} - \lambda_r \mathbf{I})\mathbf{v}_r$$

We then insert a '1' in Σ, so the final decomposition of $\mathbf{S}$ is

$$\mathbf{U}_0 = \begin{pmatrix} \mathbf{v}_0 & \mathbf{v}_1 & \cdots & \mathbf{v}_r & \mathbf{w} & \cdots & \mathbf{v}_{2N-1} \end{pmatrix}; \quad \Sigma = \begin{pmatrix} \lambda_0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 \\ 0 & \lambda_1 & \cdots & 0 & 0 & \cdots & 0 & 0 \\ \vdots & & \ddots & & & & & \vdots \\ 0 & 0 & \cdots & \lambda_r & 1 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & \lambda_{r+1} = \lambda_r & \cdots & 0 & 0 \\ \vdots & & & & & & & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & \lambda_{2N-1} \end{pmatrix}$$

The decomposition of $\mathbf{A}$ is given by

$$\mathbf{T} = \begin{pmatrix} \mathbf{U}_0 & \mathbf{0} \\ \mathbf{U}_1 & \mathbf{W}_1 \end{pmatrix}; \quad \mathbf{J} = \begin{pmatrix} \Sigma & \mathbf{0} \\ \mathbf{0} & \Delta \end{pmatrix}$$

We solve for the $2N$ unknown columns of $\mathbf{U}_1$ by solving

$$(\mathbf{S}_{12} - \lambda_i \mathbf{I})\mathbf{u}_i = -\mathbf{S}_{11}\mathbf{U}_0 \ \{ + \mathbf{u}_{i-1} \}$$

where $\mathbf{u}_i$ denotes column i of $\mathbf{U}_1$, and the rightmost term in braces is only included when Σ is not diagonal and $i = r + 1$. As a minor complication, for odd valences there is one eigenvalue ($\lambda_i = 1/8$) which is also an eigenvalue of $\mathbf{S}_{12}$, so there are multiple solutions for the corresponding column, $\mathbf{u}_i$. We use the pseudoinverse again here.

We have chosen to follow Stam's blockwise derivation, but another option would be to call `eig()` directly on the entire matrix $\mathbf{A}$, then insert the generalized eigenvector in one column of $\mathbf{T}$. This would require shuffling and rescaling the eigenvectors afterwards to restore the block structure of $\mathbf{T}$.

We tested the accuracy of the JNF by computing $|\mathbf{A} - \mathbf{T}\mathbf{J}\mathbf{T}^{-1}|$, and observed errors on the order of 10^{-9} to 10^{-14} even for valences as high as 100.

3.2 Storage and Runtime Costs

For the $\lfloor N/2 \rfloor$ faces of each valence we precompute and store the $K \times K$ matrix $\mathbf{T}^{-1}$ and the three $16 \times K$ coefficient matrices $(\mathbf{P}_k \bar{\mathbf{A}} \mathbf{T})^T, (k = 1, 2, 3)$. At first glance this is $O(N^3)$ storage per valence, but we decrease this to $O(N^2)$ by breaking out the blocks of $\mathbf{T}^{-1}$ that do not vary per face index. We store one upper left block ($\mathbf{U}_0^{-1}$) per valence, and store three lower right blocks ($\mathbf{W}_1^{-1}$) independent of valence (one version for f_0, one for $f_{j>0}$, and one for the corner case). We also do not store the upper right block of zeros. The bulk of the storage is the lower left block which varies per face.

At runtime we transform the control points into the eigenbasis, $\hat{\mathbf{C}}_0 = \mathbf{T}^{-1}\mathbf{C}_0$, using the three stored blocks of $\mathbf{T}^{-1}$. Then we evaluate the patch as in Equation (15) from Stam, except with an

extra term (in braces) when $\mathbf{J}$ is not diagonal ($N \notin \{7, 11, \ldots\}$):

$$\mathbf{s}(u,v) = \left(\sum_{i=0}^{K-1} (\lambda_i)^{n-1} x_i(\mathbf{t}_{k,n}(u,v), k)\mathbf{p}_i \right) \quad \left\{ + (n-1)(\lambda_r)^{n-2} x_r(\cdots)\mathbf{p}_{r+1} \right\}$$

where $\mathbf{t}_{k,n}(u,v)$ normalizes the (u,v) range as defined in Stam, $\mathbf{p}_i$ is a projected control point (a row in $\hat{\mathbf{C}}_0$), and $\mathbf{x}(u,v,k) = (\mathbf{P}_k \bar{\mathbf{A}} \mathbf{T})^T \mathbf{b}(u,v)$ are the so-called eigenbasis functions. ($\mathbf{b}(u,v)$ are the 16 B-spline basis functions). There is a singularity at $(u,v) = (0,0)$, so in practice we clamp both u and v to a small nonzero lower bound. The cost of the evaluation is comparable to Stam.

Matlab code to generate boundary and interior eigenvalues and coefficients is available online at the address listed at the end of this paper, along with precomputed data, and C code for runtime evaluation. (**For reviewers:** http://www.cs.utah.edu/~lacewell/subdeval).

4 Remarks

The JNF is seldom found numerically for two reasons. First, it can be difficult to tell whether two eigenvalues are identical or just very close, with finite precision arithmetic. We do not have that problem here because of the pattern in Figure 4; we already know the positions of the repeated eigenvalues, and their exact values. Second, adding small amounts of noise to a matrix can cause large changes in its JNF. This does not apply here either because we know the elements of $\mathbf{A}$ exactly according to the Catmull-Clark subdivision rules.

Recently we became aware of unpublished work which analytically derives a JNF near EVs on boundaries (including interior creases), as well as pinned EVs of Catmull-Clark surfaces [3]. We believe our numerical way of finding the JNF is much simpler to understand and implement for boundary/crease EVs, and could perhaps be extended to pinned EVs. Also, our blockwise storage method requires only $O(N^2)$ per valence, versus $O(N^3)$ per valence in [3].

5 Appendix

Matrix $\mathbf{S}$ has the same pattern for any valence and face:

$$\mathbf{S} = \begin{pmatrix}
\frac{3}{4} & \frac{1}{8} & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \frac{1}{8} \\
\frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\
\frac{3}{8} & \frac{1}{16} & \frac{1}{16} & \frac{3}{8} & \frac{1}{16} & \frac{1}{16} & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{4} & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\
\vdots & & & & & & & \ddots & & & & & \vdots \\
\frac{3}{8} & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & \frac{1}{16} & \frac{1}{16} & \frac{3}{8} & \frac{1}{16} & \frac{1}{16} \\
\frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\
\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \frac{1}{2}
\end{pmatrix}$$

Matrix $\mathbf{S}_{12}$ is the same as in the interior case, except when $N = 2$ there are only 6 rows and columns and the last row is equal to $(0, 0, \ldots, 1/8)$. Likewise, $\mathbf{S}_{11}$ is the same as in the interior case

for $N > 2$, except for face f_0; then there are only 6 rows, the last row is equal to $(1/8, 3/4, 0, 0, \ldots)$, and columns $3, \ldots, 2N-1$ of the rows above are shifted left by two columns. For $N = 2$, $\mathbf{S}_{11}$ is given by

$$\mathbf{S}_{\mathbf{11,N=2}} = \begin{pmatrix} \frac{1}{64} & \frac{3}{32} & \frac{9}{16} & \frac{3}{32} \\ \frac{1}{16} & \frac{1}{16} & \frac{3}{8} & \frac{3}{8} \\ \frac{1}{8} & 0 & 0 & \frac{3}{4} \\ \frac{1}{16} & \frac{3}{8} & \frac{3}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{3}{4} & 0 & 0 \end{pmatrix}$$

Here is the picking order for face f_0 and valences $N > 2$; see the Matlab code for the other cases. Note that $R_{m,n}$ denotes an extrapolated vertex: $v = 2v_n - v_m$.

$$\begin{aligned} \mathbf{q}^3 = \quad (\quad & R_{3,0}, 0, 3, 2N+2, R_{2,1}, 1, 2, 2N+1, \\ & R_{2N+4,2N+5}, 2N+5, 2N+4, 2N, \\ & R_{2N+12,2N+13}, 2N+13, 2N+12, 2N+11) \end{aligned}$$

References

[1] CATMULL, E., AND CLARK, J. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-Aided Design 10*, 6 (1978), 350–355.

[2] HOPPE, H., DEROSE, T., DUCHAMP, T., HALSTEAD, M., JIN, H., MCDONALD, J., SCHWEITZER, J., AND STUETZLE, W. Piecewise smooth surface reconstruction. *Computer Graphics 28*, Annual Conference Series (1994), 295–302.

[3] SMITH, J., EPPS, D., AND SÉQUIN, C. Exact evaluation of piecewise smooth catmull-clark surfaces using jordan blocks, 2004. Technical Report, UC Berkeley.

[4] STAM, J. Evaluation of loop subdivision surfaces, 1998. SIGGRAPH '98 CDROM proceedings.

[5] STAM, J. Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1998), ACM Press, pp. 395–404.

[6] STRANG, G. *Linear Algebra and Its Applications*. Brooks Cole, February 1988.

[7] ZORIN, D., AND KRISTJANSSON, D. Evaluation of piecewise smooth subdivision surfaces. *The Visual Computer 18*, 5-6 (2002), 299–315.