Subdivision Surface Evaluation as Sparse Matrix-Vector Multiplication

Michael Driscoll
Katherine A. Yelick, Ed.
Armando Fox, Ed.

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2014-231
http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-231.html

December 19, 2014

Acknowledgement

I would like to thank all whose ideas and discussion contributed to this work, including Armando Fox, Kathy Yelick, Carlo Séquin, Evangelos Georganas, Michael Anderson, David Sheffield, Nick Knight, Tony DeRose, Mark Meyer, Dirk Van Gelder, and the OpenSubdiv team.

Subdivision Surface Evaluation as Sparse Matrix-Vector Multiplication

by Michael B. Driscoll

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Katherine Yelick
Research Advisor

(Date)

* * * * * * *

Professor Armando Fox
Research Advisor

(Date)

Subdivision Surface Evaluation as Sparse Matrix-Vector Multiplication

Michael B. Driscoll

December 19, 2014

Abstract

We present an interpretation of subdivision surface evaluation in the language of linear algebra. Specifically, the vector of surface points can be computed by left-multiplying the vector of control points by a sparse *subdivision matrix*. This "matrix-driven" interpretation applies to any level of subdivision, holds for many common subdivision schemes (including Catmull-Clark and Loop), supports limit surface evaluation, allows semi-sharp creases, and complements feature-adaptive optimizations. It is primarily applicable to static meshes undergoing deformation (i.e. animation), in which case the subdivision matrix is invariant over time and the surface can be evaluated at each frame with a single sparse matrix-vector multiplication (SpMV). We describe techniques for building subdivision matrices on-the-fly using the recursive definition of the subdivision scheme and sparse matrix-matrix multiplication (SpMM) routines. The performance of our approach thus reduces to that of SpMV and SpMM, both of which have been studied extensively and are available in common packages for numerical linear algebra. We implemented our approach as an extension to Pixar's OpenSubdiv library using routines from Intel's Math Kernel Library and Nvidia's CUSPARSE library to target multicore CPUs and GPUs, respectively. We present performance results from off-the-shelf routines and our own "SpMV-like" routines that achieve 1.7-4.8x better performance than existing techniques on both platforms.We conclude by describing two major limitations of matrix-driven evaluation, namely difficulty computing vertex normals and complications in the presence of hierarchical edits, and suggest workarounds for both.

1 Introduction

Subdivision surfaces are popular primitives in modeling and animation applications. Simply put, they define a set of rules for refining a coarse, polygonal, 'control' mesh into a high-resolution one, the subdivision surface. It is especially desirable to render high-resolution subdivision surfaces interactively, thus providing maximum fidelity to artists and designers. However, many evaluation algorithms can't achieve real-time performance because of inefficient use of the memory hierarchy. To make matters worse, porting such algorithms to a variety of parallel architectures while maintaining acceptable performance remains a challenge. One way to achieve "portable performance" is to identify the primary computational pattern(s) exhibited by an application [ABC*06]. Then, when implementing the application on a particular architecture, the design can be informed by techniques associated with the pattern-architecture pair. In this paper, we use a pattern-driven approach to inspire a new algorithm for the evaluation of subdivision surfaces on multicore CPUs and GPUs.

Our algorithm is based on the well-known property of many subdivision schemes that surface points can be computed as linear combinations of control points, i.e. the vertices in the coarse mesh. Therefore, each model must have a *subdivision matrix* that captures the transformation from the control points directly to the surface points at a particular level. We form subdivision matrices explicitly, and we evaluate the surface by computing the product of the subdivision matrix and the vector of control points. By reducing the evaluation problem to matrix-vector multiplication, we can draw upon techniques from numerical linear algebra to implement our approach. We're specifically interested in sparse matrix-vector multiplication (SpMV) because subdivision matrices are mostly zero-valued in practice.

The contributions of this paper are:

- An interpretation of subdivision surface evaluation as sparse matrix-vector multiplication.
- An algorithm for generating subdivision matrices on-the-fly using matrix-matrix multiplication and the recursive definition of the subdivision scheme in use.
- Extensions for supporting limit surface evaluation and semi-sharp creases.
- New SpMV-like routines, optimized for surface evaluation, that target multicore CPUs and GPUs.
- Performance results demonstrating the utility of our approach in a real-time setting, namely computer animation.

This paper is organized as follows. Section 2 reviews existing techniques for subdivision surface evaluation. Section 3 presents our matrix-driven evaluation algorithm. Section 4 gives details of our CPU and GPU implementations, and Section 5 analyzes their performance on several metrics, including initialization cost, throughput (a variant of frame rate), and memory requirements. Section 6 describes limitations and future work.

2 Related Work

Numerous approaches have been proposed for the evaluation of subdivision surfaces, including recursive refinement [SJP05], parametric evaluation [Sta98], evaluation by precomputed basis functions [BS02], and approximation with Gregory patches [LSNCn09]. These approaches provide different trade-offs in terms of features and performance, so we focus here on those that are most similar to our approach. Neißner et al. [NLMD12] provide a survey of current techniques and compare their performance.

Bolz and Schröder [BS02] observe that surface points are linear combinations of control points, and hence a particular surface point can be evaluated as a dot product between control points in the local neighborhood and a vector of special coefficients. The coefficients are the result of evaluating basis functions at the appropriate parametric values and are independent of the geometric location of the control points. To generate the basis functions, the authors propose subdividing a unit pulse centered at the point of interest. Unique basis functions are required for every valence found in the mesh; the authors use approximately 5300 functions to cover a reasonable, but still limited, range of valences. Their approach is similar to ours in that they precompute coefficients which are later combined with control points at runtime; however, we observe that the collection of dot products can be represented as a single matrix-vector product. Furthermore, the number of basis functions is unbounded if the scheme is extended to support semi-sharp creases because the sharpness parameter can assume any value in $[0, 1]$. Consequently, basis functions must generally be computed on-demand. Our approach provides an efficient means of computing basis function coefficients while still supporting semi-sharp creases and other common features.

Feature-adaptive subdivision [NLMD12] exploits the fact that repeated subdivision of regular mesh regions (i.e. regions without extraordinary points) yields well-understood surface primitives. The authors give an algorithm that uses hardware tessellators to refine regular areas of the mesh and, near irregular points, they use a special table to guide the evaluation of surface points iteratively from the control mesh. Their "table-driven" approach is amenable to parallelization and supports semi-sharp creases and hierarchical edits. They also support limit-surface evaluation

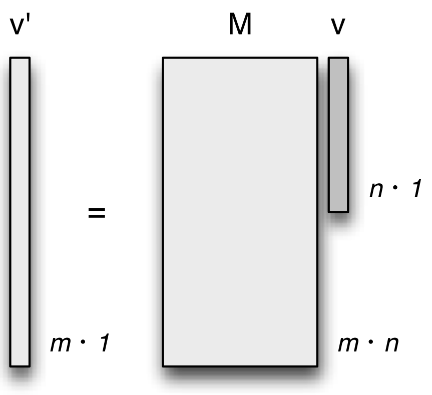


Figure 1: In matrix-driven evaluation, a vector of n control points v is left-multiplied by a subdivision matrix M, yielding m surface points. M is uniquely defined by n, the subdivision scheme, the subdivision level, the presence of semi-sharp creases, and the control mesh.

via a special vertex shader that moves surface points to their limit position. In the absence of hardware tessellators, their approach relies solely on table-driven evaluation. Our approach can be viewed as a replacement for table-driven evaluation because the subdivision matrix encodes much of the same information contained in the subdivision table. We expect our approach to be complementary to feature-adaptive optimizations, and therefore we compare the performance of table-driven and matrix-driven evaluation in Section 5. Lastly, our approach has the benefit of using established data structures—sparse matrix formats—whose performance implications are well understood, and for which many low-level optimizations exist.

3 Algorithm

In this section, we present the details of matrix-driven surface evaluation in the context of the Catmull-Clark and Loop subdivision schemes. We also give a method for incorporating limit surface evaluation and semi-sharp creases.

3.1 Abstract Interpretation

For purposes of exposition, we define a linear subdivision scheme as one in which the surface points are linear combinations of control points. In a linear scheme, the vector of surface points v' is equal to the product of a subdivision matrix $\mathbf{M}$ that captures the linear mapping and the vector of control points v, or mathematically

$$v' = \mathbf{M}v.$$

This relationship is illustrated in Figure 1. Aside from linearity, we make no assumptions about the subdivision scheme.

The subdivision matrix is uniquely defined by the subdivision scheme, the subdivision level, the number and connectivity of control points, and the presence of semi-sharp creases. Most importantly, it is independent of the geometric locations of the control points. For static meshes deforming over time, the subdivision matrix is invariant and can be re-used at each frame. For this reason, we anticipate our techniques will be most applicable in settings where the position of the control mesh is unpredictable, as in feature film animation. If the motion of the control mesh is known a priori, blending-based methods will probably be cheaper.

Assuming linearity hardly limits the practicality of our approach, as the Catmull-Clark, Loop, Butterfly, and $\sqrt{3}$ subdivision schemes are linear. In this paper, we focus on the aforementioned two, whose linearity is apparent upon inspection of the recursive definition of each scheme. We review both schemes here.

Catmull-Clark subdivision [CC98] refines a quadrilateral mesh into $C2$ continuous surfaces everywhere except at irregular (non-valence-4) points, where the surface is C^1 continuous. Each subdivision step simultaneously "smooths" existing points and adds new points along edges and at the centroid of faces. A subdivision mesh at level $i+1$ can be computed from level i points using the following stencils. Here, n refers to the valence of the point to be refined, sums are taken over neighbors, and "+" and "-" subscripts refer to points or faces on opposite sides of an edge.

- Face points: $f^{i+1} = \frac{1}{k}\sum_{j=1}^{k} v_j^i$
- Edge points: $e^{i+1} = \frac{1}{4}(v_+^i + v_-^i + f_-^{i+1} + f_+^{i+1})$
- Point points: $v^{i+1} = \frac{n-2}{n}v^i + \frac{1}{n^2}\sum_j e_j^i + \frac{1}{n^2}\sum_j f_j^{i+1}$

In static meshes, n is constant and the subdivision rules reduce to linear combinations of the control points.

Loop subdivision [Loo87] refines a triangle mesh into surfaces that approach box splines. Each subdivision step smooths existing points and introduces new points along edges in the base mesh. The subdivision stencils that compute level $i+1$ points of valence n are:

- Edge points: $e^{i+1} = \frac{1}{8}(3v_+^i + 3v_-^i + e_+^i + e_-^i)$
- Point points: $v^{i+1} = \frac{1}{8}(5v^i + 3\sum_j v_j^i)$.

The linearity of Catmull-Clark and Loop subdivision schemes can be show by a simple inductive argument: points at each level are linear combinations of those from the previous level. Therefore, surface points at any level must be linear combinations of the control points.

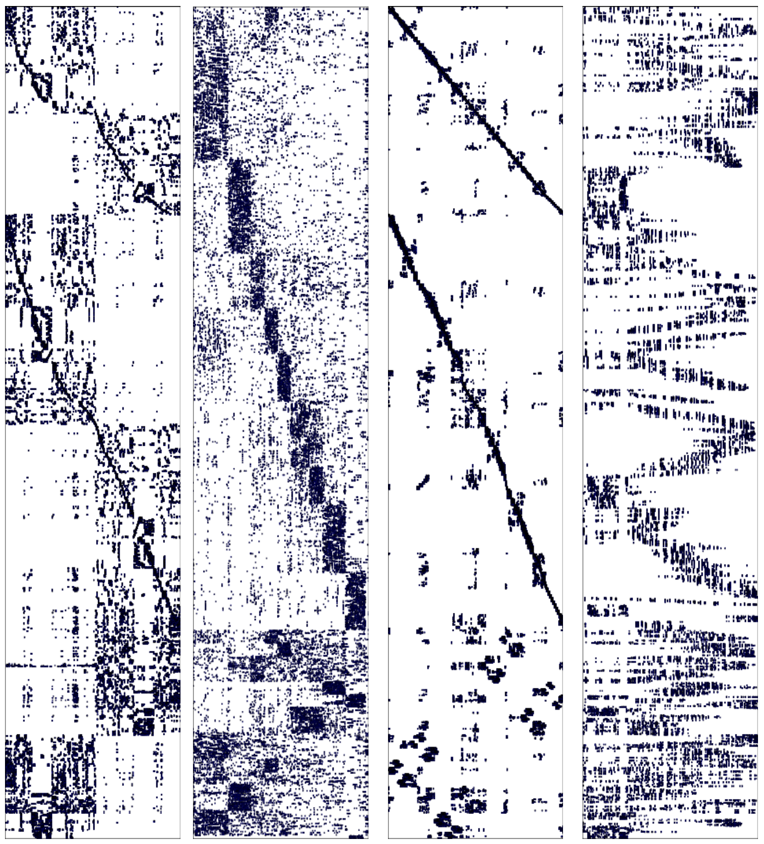
3.2 Intuition for Subdivision Matrices

Before describing how to construct subdivision matrices, we review their properties that reflect characteristics of the surface evaluation problem.

A subdivision matrix that is m-by-n in size maps n control points to m surface points. The matrix entry at (i, j) represents the contribution of control point j to surface point i. Because the number of surface points is exponentially larger than the number of control points ($m \gg n$), subdivision matrices have a characteristic "tall-skinny" aspect ratio. This reflects their purpose of projecting the influence of a few control points onto many surface points. For example, in the case of a single Catmull-Clark subdivision step on a perfectly regular mesh, the corresponding matrix is four times taller than wide.

For realistic models, the subdivision matrix is sparse, or mostly zero-valued. This is a direct reflection of the local support property of subdivision surfaces and B-splines in general. To understand why, consider a row vector of the subdivision matrix. The dot product of the row vector and the vector of control points yields a single surface point, which is known to be a combination of a small number of control points. For example, 16 control points influence a surface point in Catmull-Clark subdivision of a regular mesh. The remainder of the coefficients in the row vector must be zero. Since the row vector has length n and $n \gg 16$ in practice, the subdivision matrix must be sparse. The matrices we saw in practice have about 1% non-zero values. The converse argument can be made as well: column vectors are sparse because a control point only influences surface points in the local neighborhood.

Sparse matrices that represent real systems often have interesting or useful structure. This can be visualized using spy plots, which display the matrix with black dots over nonzero elements. Figure 2 shows spy plots for actual



(a) Big Guy
Level 2
30,456 Points
Catmull-Clark

(b) Bunny
Level 1
12,479 Points
Loop

(c) Monster Frog
Level 2
27,180 Points
Catmull-Clark

(d) Venus
Level 2
14,895 Points
Loop

Figure 2: Spy plots reveal matrix structure by drawing black dots over non-zero elements. Shown here are subdivision matrices for selected meshes, illustrated later in Figure 6. Surface points (or matrix rows) are sorted by type: for n control points, roughly the first n rows compute face-points (if applicable), the next $2n$ rows compute edge-points, and the last n rows compute vertex-points.

subdivision matrices. The diagonal patterns exhibited in the plots are indicative of nearest-neighbor stencil routines, which is exactly what subdivision schemes are. Another interesting feature is the relatively uniform distribution of non-zeros per row; this fact will be useful when trying to load-balance the parallel evaluation of the matrix-vector product.

3.3 Constructing the Subdivision Matrix

Until now, we've described our surface evaluation algorithm assuming an existing subdivision matrix. Here, we show how it can be generated efficiently for a particular mesh via the recursive definition of the subdivision scheme and matrix-matrix multiplication. We make a distinction between *single-level* subdivision matrices that capture the transformation from one subdivision level to the next and *multi-level* matrices that capture the transformation from the base mesh to an arbitrary refinement level. For the remainder of this paper, subdivision matrices can be assumed to be multi-level unless otherwise stated. We use multi-level matrices when we assess performance in Section 5.

Single-level subdivision matrices can be constructed straightforwardly from the recursive definition of the subdi-

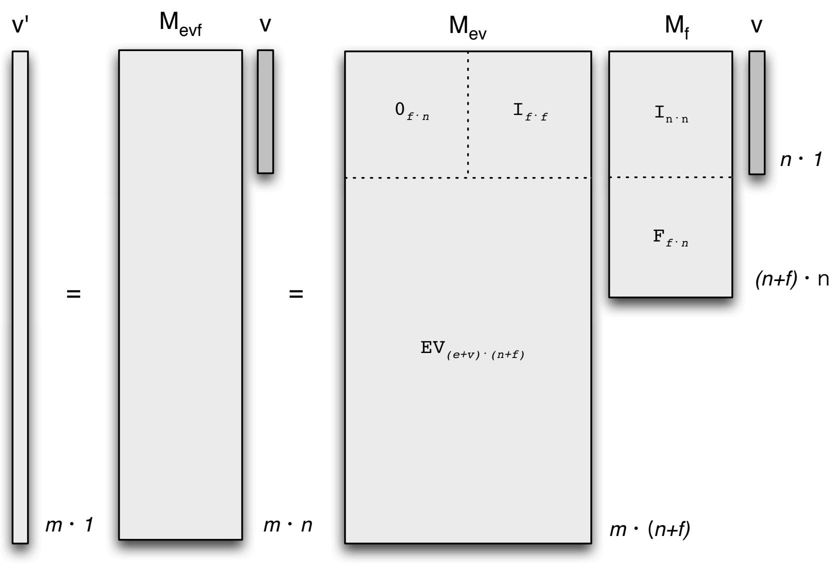


Figure 3: The rules for Catmull-Clark subdivision contain a data-dependence that requires face-points to be computed before edge-points and vertex-points. We can eliminate this dependence by using two intermediate subdivision matrices. M_f computes new face-points and copies control points; then, M_{ev} computes new edge-points and vertex-points in term of the control points and the new face-points. Their product captures the dependence.

vision scheme. For a particular row in the matrix, coefficients gleaned from the subdivision rules can be placed in the appropriate columns to 'pull' a fraction of the corresponding control point's value. Loop subdivision matrices can be constructed in this way.

Complications arise in schemes that have data dependencies between points within a given level. For example, in Catmull-Clark subdivision, edge-points and vertex-points at level $i+1$ are weighted sums of face-points on level $i+1$ and points in the level i mesh. We see two ways around the dependence: 1) rewrite the scheme's definition to be exclusively in terms of previous-level points, or 2) use a product of intermediate matrices to capture the dependence. The former is less straightforward in the presence of boundaries and semi-sharp creases, so we opt for the latter. To compute the subdivision matrices for a single Catmull-Clark subdivision step at level i, we first generate an intermediate matrix that copies the full set of points at level i and simultaneously computes the face-points at level $i+1$. Then, we generate a second matrix that copies the level $i+1$ face-points and computes the level $i+1$ edge-points and vertex-points in terms of the level $i+1$ face-points and level i points. The product of these matrices yields a single-level, Catmull-Clark subdivision matrix. This process is illustrated in Figure 3. Schemes with more dependencies can be handled by additional intermediate matrices.

Multi-level subdivision matrices capture the transformation from the base mesh to the surface at an arbitrary level. They are, quite simply, the product of all single-level matrices up to the desired level. Figure 4 shows how three single-level matrices can be collapsed into a multi-level matrix. We show in Section 5 that multi-level matrices are cheap enough to be computed on-the-fly.

3.4 Limit surface evaluation

Matrices constructed using our technique, as described thus far, compute the approximate subdivision surface. For applications that require limit surface evaluation, we provide a simple extension here.

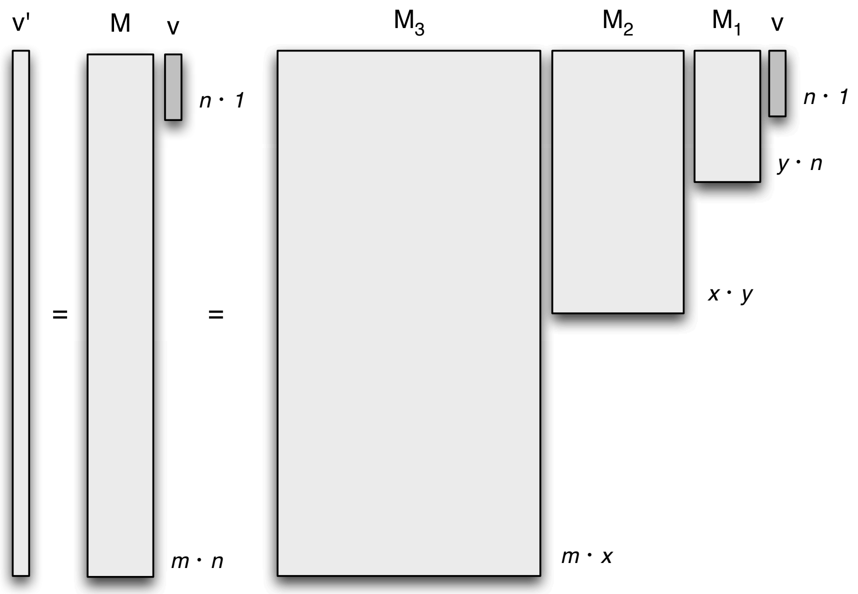


Figure 4: Multiple subdivision steps are captured by the product of single-level subdivision matrices. Here, M_1, M_2, and M_3 combine to form M, a transformation from the control points directly to the level-3 surface points.

It has been known that points on the approximate surface can be ‘pushed’ to the limit surface via the application of special limit stencils. The stencils are a natural fit for our approach because they update points based on weighted sums of neighboring points—in other words, they’re linear and local. We use the limit stencils given by Halstead, Kass, and DeRose [HKD93] for Catmull-Clark subdivision, and Hoppe et al. [HDD*94] for Loop subdivision.

The coefficients of the stencil depend only on the valence of the point in question and are invariant for static meshes. Therefore, we can capture the effect of applying all stencils to all points in a square ‘limiting’ matrix that transforms points from the approximate surface to the limit surface. Figure 5 shows the limiting matrix and its relationship to single-level matrices. A multi-level matrix that includes a limiting matrix captures the direct transformation from the control points to the limit surface.

The extension for limit surface evaluation has negligible impact on the performance of the surface evaluation routine. This is because the cost of sparse matrix-vector multiplication is proportional to number of non-zeroes in the matrix, which is approximately equal in both cases. The primary difference is in the *values* of the non-zero elements. We enable limit-surface evaluation when we assess performance in Section 5.

It is moderately more expensive to generate exact subdivision matrices because the limiting matrix must be generated and multiplied into the approximating matrix. In practice, we found the extra cost to be no more than one second, as shown in detail in Figure 9, so the technique is applicable in an interactive setting. In that case, the total time was heavily dominated by traversal of the half-edge mesh, not the matrix-matrix multiplication.

3.5 Sharp and Semi-sharp Creases

Many real-world models contain sharp features that are difficult to represent with a simple control mesh. To capture sharp details efficiently, Hoppe et al. [HDD*94] augment the usual set of subdivision stencils with a secondary set that maintains sharpness at corners or creases. Edges can be tagged to indicate whether smooth or sharp stencils should be applied during evaluation. Our algorithm supports sharp stencils because, like the usual smooth stencils, they’re linear and local. Thus, a subdivision step with both smooth and sharp features can be captured in a single matrix.

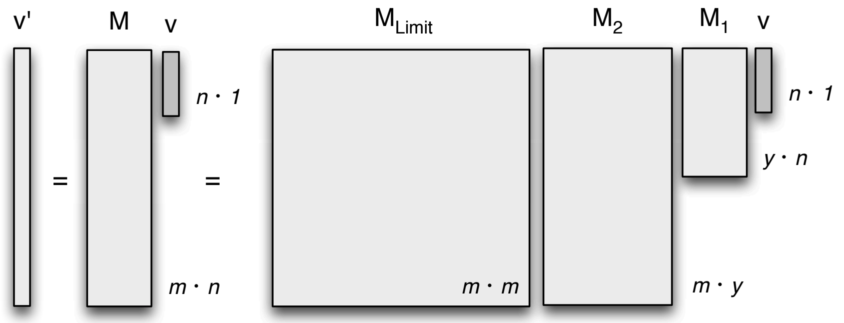


Figure 5: Limit stencil coefficients for all points can be captured in a square matrix. When the square matrix is multiplied into the approximate subdivision matrix, the resulting exact subdivision matrix transforms the control points directly to the limit surface.

Real-world models often contain “semi-sharp” features that appear sharp from a distance but are smooth up close. DeRose et al. [DKT98] achieve this effect by applying sharp subdivision masks for some number of subdivision steps, and then switching to smooth masks for the remaining steps. A sharpness value s controls the switching point. In the case of non-integer s, point positions are computed as a linear blend between the positions suggested by the sharp and smooth stencils. The operations remain linear, so we can capture the effect of semi-sharp creases in a matrix.

For meshes with static sharpness values, we can compute subdivision matrices that capture the effect of semi-sharp creases. Note that we maintain no additional data structures at runtime for sharpness tags—the subdivision matrix merely contains different coefficients that reflect the application of the tags. Thus, the performance of evaluation routine is independent of the number of sharpness tags. Construction of the subdivision matrix is slightly more costly because sharpness data must be considered, but the difference is negligible.

4 Implementation

We implemented our approach as alternative evaluation kernels in the OpenSubdiv package [Pix12]. In the absence of hardware tessellators, OpenSubdiv’s default kernels implement table-driven subdivision of quadrilateral and triangle meshes. Like OpenSubdiv, we support quadrilateral and triangle meshes, and we target two platforms: multicore CPUs and GPUs. We implemented two kernels for each platform: a ‘vanilla’ kernel that uses off-the-shelf libraries and a custom kernel that applies optimizations specific to our task. All matrix-driven kernels are agnostic to the subdivision scheme in use, unlike the corresponding table-driven kernels.

4.1 Vanilla Subdivision Kernels

Because we have reduced the problem to common linear algebra operations, we can implement matrix-driven evaluation using existing libraries. Many chip vendors provide tuned versions of such libraries for maximum performance. We build on Intel’s Math Kernel Library (MKL) to target multicore CPUs and Nvidia’s CUSPARSE library to target GPUs. Various free and open-source alternative also exist, such as CSparse [Dav06] and OSKI [VDY05].

4.1.1 Sparse Matrix Formats

Subdivision matrices are inherently sparse and can be represented efficiently by a variety of common formats. Sparse formats achieve space-efficiency by only storing non-zero entries and their corresponding locations in the matrix.

Different formats provide trade-offs in terms of space, ease of construction, and compute performance. We use a collection of formats, described here, in our implementation.

The Coordinate List format (COO) stores non-zero elements as *(row,column,value)* triplets. It uses space proportional to the number of non-zeroes, specifically $3 \cdot \#nnz$. COO performs relatively poorly, but is insensitive to irregularity in the structure of the data.

The Compressed Sparse Row format (CSR) stores an m-by-n matrix as a vector of length $m+1$ that indexes into sorted arrays of column indices and values. CSR requires memory proportional to $m + 2 \cdot \#nnz$. When row lengths are relatively uniform, as in subdivision matrices, CSR matrices can be evaluated efficiently in parallel, and yield relatively good performance.

The ELLPACK format (ELL) stores an m-by-n matrix with no more than k non-zeroes per row as a dense m-by-k matrix of values and an associated dense matrix of column indices. Rows with fewer than k non-zeroes are zero-padded, yielding memory usage that scales with $2 \cdot m \cdot k$. ELL is especially suited for GPU platforms because of its regularity.

4.1.2 Sparse Matrix Routines

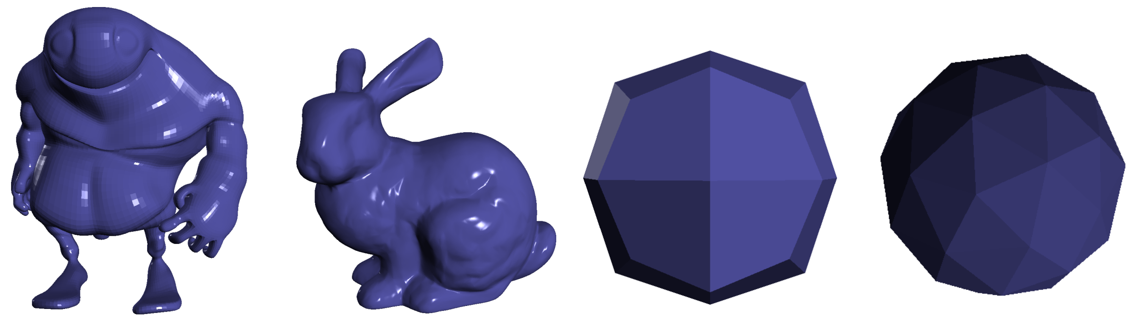
Sparse matrix libraries provide routines for operating on matrices in their sparse representation. We use two routines within our vanilla kernels. To multiply two sparse matrices, we convert the matrices to CSR format, if necessary, and use `mkl_scsrmultcsr` or `cusparseScsrgemm`. One advantage of using off-the-shelf libraries is that their routines are often internally parallelized. In fact, our vanilla GPU kernel contains no user-level CUDA code at all, yet matches the performance of the expertly-coded, table-driven CUDA kernel.

To evaluate the surface, we need to multiply the subdivision matrix by the vector of control points. However, points themselves are vectors (of six single-precision floating point numbers, in our case, corresponding to spatial and texture coordinates). Furthermore, the subdivision matrix is in terms of ‘logical’ points, i.e. the coefficient at location (i,j) should be applied to all six elements of control point j and added to all six elements of surface point i. In effect, we want to perform an SpMV with “multiple right-hand sides”—the column vectors of vertex elements. Fortunately, the sparse matrix-matrix multiplication (CSRMM) routines `mkl_scsrmm` and `cusparseScsrmm` accomplish this task. Although the operation is technically no longer SpMV, the memory access pattern is similar and SpMV optimizations provide benefits to our custom kernels.

4.2 Custom Subdivision Kernels

Because we have knowledge about the structure and dimensions of the matrices and vectors, we can write custom, “SpMV-like” routines that are optimized for our particular dimensions and sparsity patterns. This task is greatly simplified by the vast amount of literature on SpMV, studied because it lies at the heart of many iterative solvers. We draw on this knowledge to implement our own routines that run more than twice as fast as table-driven subdivision. In both cases, the only difference between our routines and standard SpMV routines is that each non-zero multiplies all six elements of a control point, and the result is added element-wise to all six elements of a surface point. In contrast, each non-zero in standard SpMV multiplies one element in the source vector and the result is added into one element in the destination vector. Note that our custom kernels still use MKL and CUSPARSE for matrix construction; we only optimize the surface evaluation routine.

Our Custom CPU kernel targets multicore CPUs and implements optimizations suggested by Williams et al. [WOV*07]. We parallelize the computation by assigning threads to blocks of matrix rows. Such an assignment ensures load balance because rows have about the same number of non-zeroes (and hence work). We use 4-wide



(a) Big Guy (b) Bunny (c) Cube (d) Icosahedron

Model	Primitive	Scheme	Faces	Vertices	Extraordinary Vertices	Manifold
Big Guy	Quads	Catmull-Clark	2,900	1,452	13%	Yes
Bunny	Triangles	Loop	2,915	1,494	68%	No
Cube	Quads	Catmull-Clark	6	8	100%	Yes
Icosahedron	Triangles	Loop	20	12	100%	Yes

Figure 6: Models used in performance analysis. We selected models composed of few or many vertices, quadrilaterals or triangles, and few extraordinary points or many. We didn't include models with semi-sharp creases because their presence doesn't affect the performance of matrix-driven evaluation, as described in Section 3.5. The Cube and Icosahedron models are shown after one level of subdivision.

SIMD intrinsics to process vertex elements in parallel. Sparse matrices are usually traversed using two loops, but we reduce it to one to avoid extra control overhead. We have the option of prefetching column indices and values (which are accessed sequentially), but we didn't because the kernels were running at peak memory bandwidth and thus it would provide little benefit. Our custom kernel achieves the best performance among those running on the CPU.

Our Custom GPU kernel mirrors the SpMV implementation of Bell and Garland [BG09]. We use their hybrid matrix format that splits an m-by-n matrix, with about k non-zeroes per row, into an ELL matrix and a COO matrix. Rows with fewer than k non-zeroes are zero-padded, and rows with more than k non-zeroes spill the extra entries to the COO matrix. We use empirical search to determine the best value of k in the range 4-30, as described in more detail in Section 5.2.1. Our custom GPU kernel achieves the best absolute performance overall.

5 Performance

Matrix-driven evaluation provides better per-frame performance than table-driven evaluation on both multicore CPUs and GPUs. Furthermore, we believe the construction of subdivision matrices is cheap enough to do online. Here, we assess the utility of our approach by analyzing it on several metrics, including per-frame performance, matrix construction cost, memory usage, and numerical accuracy.

5.1 Testing Methodology

We implemented our matrix-driven routines in OpenSubdiv Beta, Version 1.1. OpenSubdiv provides table-driven subdivision routines that target multicore CPUs via OpenMP and GPUs via Cuda and OpenCL. The beta version doesn't implement limit-surface evaluation in the table-driven kernels, but we still enabled it within our matrix-driven kernels. We expect our performance advantage to be even greater when compared against table-driven limit surface evaluation. We ran our experiments on a quad-core, 2.67 GHz Intel Core i7-920 CPU with an attached Nvidia GeForce

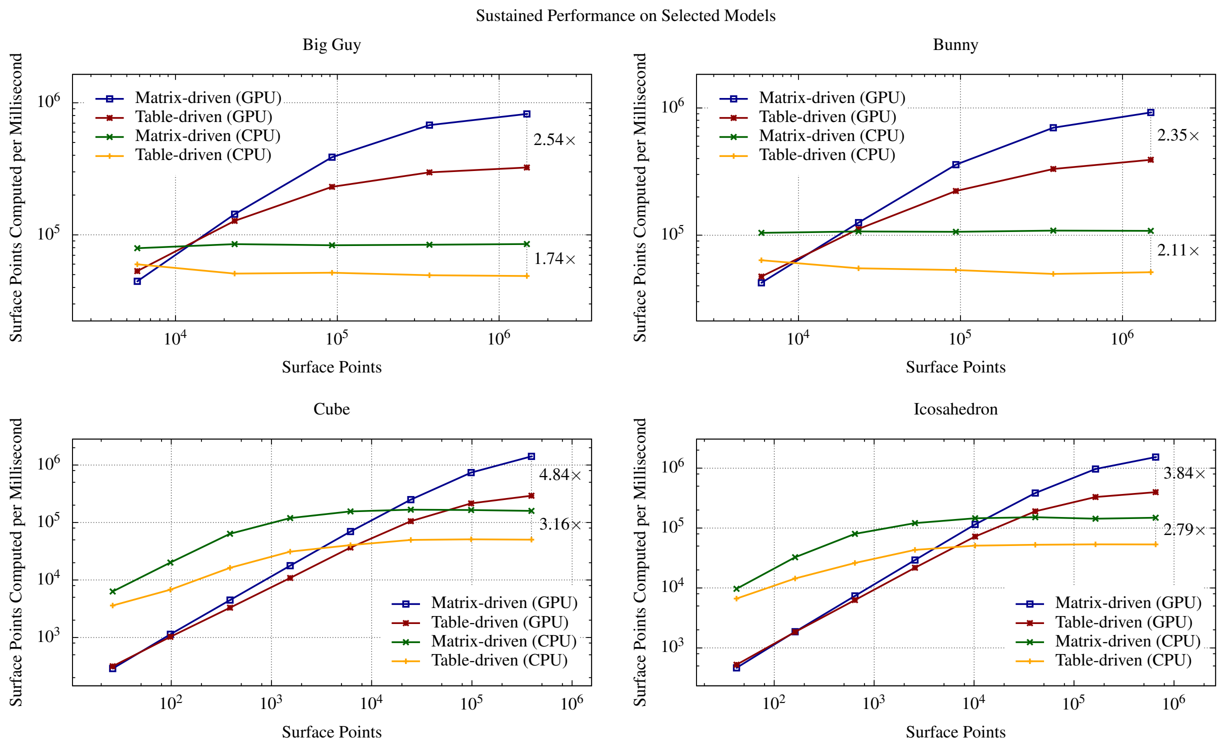


Figure 7: Performance of surface evaluation routines as a function of subdivision level, starting from the first. The table-driven results are taken from OpenSubdiv's 'OpenMP' and 'Cuda' kernels; the matrix-driven results show the performance of our custom, SpMV-like kernels.

GTX480 GPU. Our driver code rendered frames using OpenGL, but the times we report are restricted to the surface evaluation subroutine or the matrix-generation routines, as appropriate.

We evaluated our algorithm on four models, including two quad meshes and two triangle meshes. Within each category, we chose one small, fundamental shape and one larger, real-world object. The models are shown in Figure 6. The Bunny and Big Guy models are simplified versions of the ones provided by the Stanford Graphics Laboratory.

5.2 Surface Evaluation Performance

The goal of casting surface evaluation as SpMV was to accelerate the per-frame evaluation routine. To provide a pure comparison between table-driven and matrix-driven approaches on different platforms, we measure performance in terms of surface points calculated per unit time instead of the typical frames per second. The rates we report are averaged over one thousand frames. Figure 7 shows the performance of the best table-driven and matrix-driven kernels, specifically the 'OpenMP' and 'Cuda' kernels from OpenSubdiv and our custom CPU and GPU kernels. The GPU-backed kernels attain higher absolute performance than the CPU-backed ones, so we analyze those classes separately.

The CPU-backed kernels show consistent performance across a variety of problem sizes. The custom, matrix-driven kernel achieves 2× better performance than the table-driven kernel on the large models at the finest subdivision level. It performs especially well on the small models, where it gets 3× better performance than the table-driven kernel. The small models yield a larger improvement because their control points are few enough to remain L1-resident as the surface points are computed.

The GPU-backed kernels outpace the CPU-backed kernels in terms of absolute performance on the largest problem sizes. As a rough metric, problems require about 10,000 surface points before seeing a benefit from GPU parallelism. The matrix-driven GPU kernel prevails in this range, getting more than $2\times$ better performance that the table-driven kernel. On the small models, the matrix-driven kernels see up to $4.8\times$ better performance. We attribute this to two factors: 1) the control points fit in cache, and 2) all control points have the same valence, so Bell and Garland's hybrid matrix format is especially suitable.

We haven't plotted the performance of the vanilla matrix-driven subdivision kernels because they perform poorly. The vanilla CPU kernel, built on MKL, merely matches the performance of the serial table-driven kernel. This is probably because MKL's CSRMM routine isn't optimized for the particular matrix dimensions in use—it's more likely optimized for dense matrices that are large and square rather than the small, skinny ones we provide. The vanilla GPU kernel exhibits better relative performance, matching the speed of the best table-driven GPU kernel. The difference in performance between the custom and vanilla kernels suggests that treating the problem as SpMV, instead of CSRMM, is a profitable interpretation.

5.2.1 Tuning k

The hybrid matrix format introduces a tunable parameter k that determines the balance of non-zeroes between the ELL and COO matrices. We use offline empirical search to determine the best value of k between 4 and 30, a range which represents the typical number of control points that influence a surface point. Figure 8 illustrates the effect of k on overall performance. When k is large, most non-zeroes are represented in the ELL matrix, at the expense of explicit zeroes in some matrix rows. When k is small, most non-zeroes are stored in the COO format, which avoids explicit zeroes but performs relatively poorly. Consequently, there exists a happy medium in which we capture enough irregularity in the COO matrix without too much padding in the ELL matrix.

Alternatively, we can use the heuristic given by Bell and Garland to predict a good value of k. In practice, we found that the predicted value yields performance that was 35% and 23% slower than the optimal value for Big Guy and Bunny, respectively. The heuristic tends to underestimate the best value of k which, unfortunately, is much worse than overestimating it.

5.3 Cost of Generating Subdivision Matrices

Our algorithm has a non-trivial cost associated with computing the subdivision matrices; however, this cost is dominated by traversal of the OpenSubdiv half-edge mesh representation. Both the table-driven and matrix-driven approaches require a traversal, so here we focus on the additional time spent processing subdivision matrices. Specifically, we time routines that convert sparse matrices from one format to another and routines that multiply sparse matrices. This effectively captures the extra cost of initializing a matrix-driven kernel. We use the same routines for matrix-matrix multiplication and matrix format conversion in both our vanilla and custom kernels, so we only report times from the former.

The cost of generating subdivision matrices is shown in Figure 9. Our worst-case times fall within a few seconds, so we believe subdivision matrices can be calculated on-the-fly. If such times aren't tolerable, subdivision matrices could be computed offline and loaded at runtime for near-instant rendering. Alternatively, they could be computed on a cluster (online or offline) using distributed-memory routines for matrix-matrix multiplication [BG12].

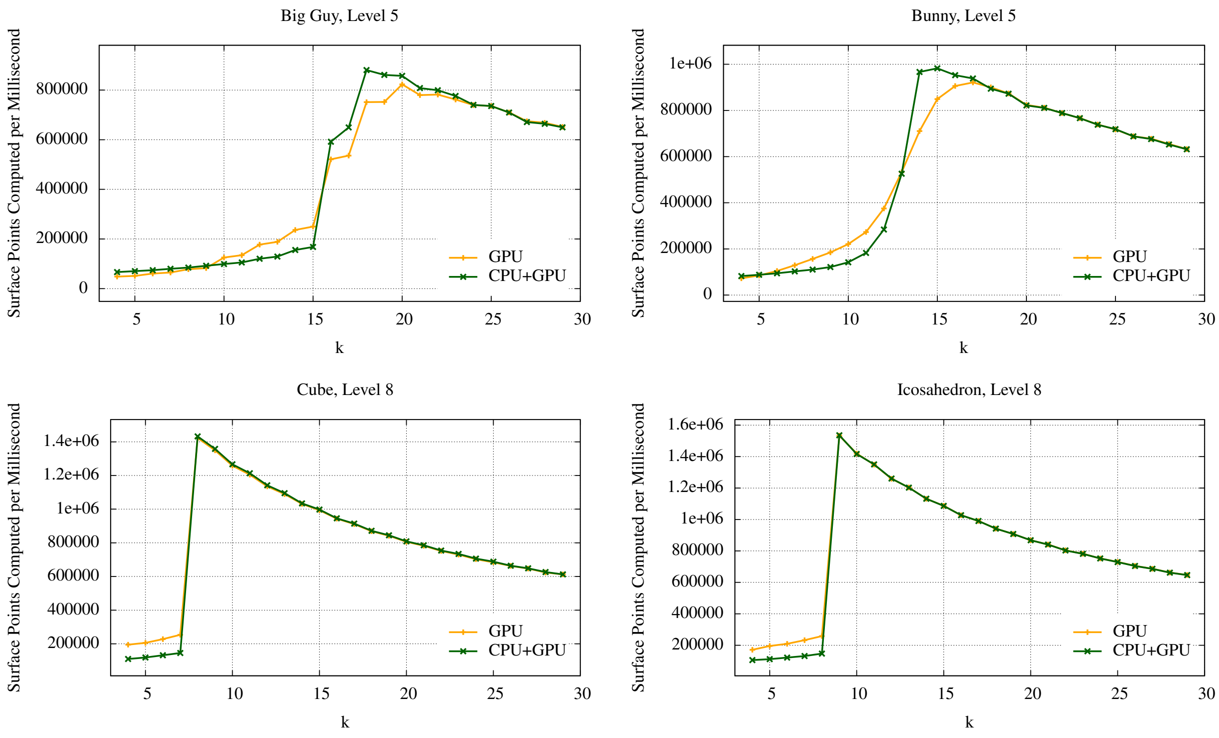


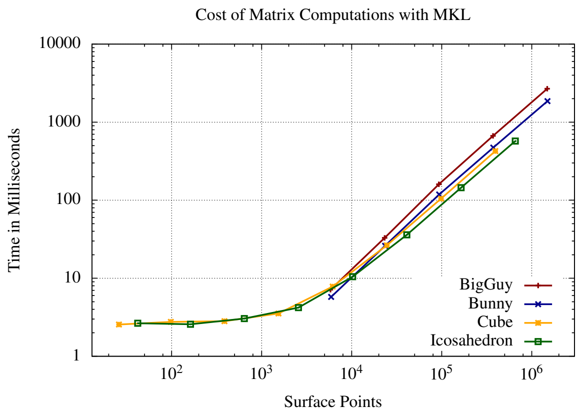
Figure 8: Performance as a function of split parameter k, which determines the balance of non-zeroes between the ELL and COO matrices (see Section 5.2.1). We plot performance for ELL and COO routines running back-to-back on the GPU, and in parallel with COO on the CPU and ELL on the GPU. The latter includes the time to copy the partial result across the PCI bus and combine it via vector-vector addition.

5.4 Memory Requirements and Numeric Accuracy

Our algorithm requires non-trivial space for storage of the subdivision matrix. The largest model, Big Guy, uses 187 MB at the finest subdivision level. The storage requirements of the subdivision matrix scale linearly with the surface resolution, as desired. The memory requirements are large, but they are reasonable given modern memory capacities and comparable to OpenSubdiv's. In contrast to table-driven subdivision, matrix-driven evaluation requires 1.2-1.8x more memory on the models we tested, but yields about a ~2x speedup. In many applications, the performance benefits may justify using half-again as much memory.

Full-featured subdivision surface evaluation is known to be heavily memory-bound. Thus, one would expect the performance of matrix-driven evaluation to suffer because it uses more about 50% more memory than table-driven evaluation. However, we find that this is not the case in practice. By casting the problem as SpMV, we confine non-sequential memory accesses to the vector of control points, which is small enough to remain cache-resident during evaluation. In our experiments, matrix-driven evaluation achieved 95-98% of sustainable CPU memory bandwidth (measured using the STREAM benchmark [McC95]), whereas table-driven evaluation achieved 36% at best. On the GPU, matrix-driven evaluation saw 58-82% of sustainable bandwidth, versus 22% for table-driven evaluation.

Because floating-point arithmetic is not associative, we must ensure that our approach—which reorders operations—is numerically stable. Experiments on the models presented and others revealed that our answers are always within 10^{-5} units of the table-driven solution. This result holds in single-precision arithmetic when the models are between 1 and 100 units in each dimension. Neither matrix-driven nor table-driven evaluation are approximation schemes, and both compute the same, mathematically-correct answer assuming exact arithmetic.



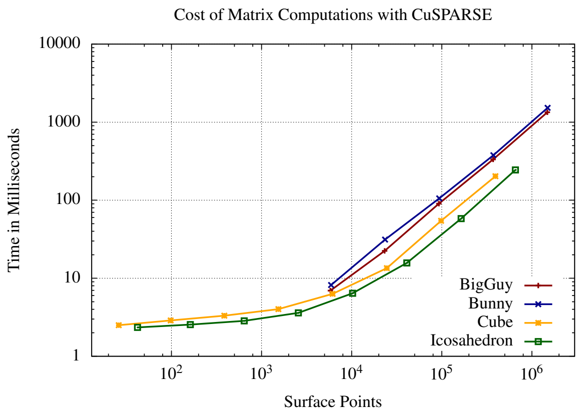


Figure 9: Time spent in MKL routines for matrix-matrix multiplication and conversion between sparse matrix formats. This metric captures the extra cost of initializing a matrix-driven evaluation kernel. The same computation implemented with CuSPARSE performs similarly in the latency-limited regime, and roughly $2\times$ better in the bandwidth-limited regime.

6 Limitations and Future Work

We foresee two obstacles to the adoption of matrix-driven evaluation as a replacement for table-driven evaluation. Both arise because matrix-driven evaluation can only compute surface points, or more generally attributes, that are linear in the control points.

First, practitioners often require both the positions and normals of surface points. Surface normals aren't linear in the control points, so they can't be evaluated directly with matrix-driven techniques. Despite this shortcoming, our approach does suggest a partial solution. Because surface tangents are linear in the control points, we can form matrices to evaluate the complete set of u- and v-tangents, from which the normals can be readily computed. Matrix-driven evaluation is obviously a component in such a framework, but at a higher level, it suggests how to organize the computation to maximize use of the memory hierarchy.

Second, many meshes use *hierarchical edits* to represent fine details on the surface. In their basic form, hierarchical edits allow the displacement of any vertex at any level in the recursive subdivision process. When the displacement is applied in the local surface frame, as it usually is, the surface is no longer linear in the control points. Hierarchical edits at intermediate subdivision levels pose the biggest problem for our scheme because we can no longer capture the full transformation in a single matrix. In such a case, we propose using one matrix to transform the base points to the intermediate-level mesh, applying the edits, and using another matrix to transform the intermediate points to the surface. The performance implications of this approach are unclear.

7 Conclusion

We have presented an algorithm that evaluates subdivision surfaces by left-multiplying the vector of control points by a mesh-dependent subdivision matrix. The existence of the matrix follows directly from the linearity of subdivision, but it wasn't clear that this interpretation is useful in practice. By drawing on techniques from numerical linear algebra, we were able to develop matrix-driven evaluation kernels that double the performance of table-driven kernels, yet still maintain support for semi-sharp creases and limit surface evaluation. Furthermore, we've shown that subdivision matrices are cheap enough to be generated in an interactive setting. They may also be useful in an analysis setting because they capture the direct transformation from the control points to the limit surface. We expect matrix-driven evaluation to be especially applicable to feature-adaptive subdivision, where it can provide input data to hardware tessellators more than twice as fast as table-driven evaluation.

Source Code

The source code used in our experiments is available online at https://github.com/mbdriscoll/OpenSubdiv/.

Acknowledgments

I would like to thank all whose ideas and discussion contributed to this work, including Armando Fox, Kathy Yelick, Carlo Séquin, Evangelos Georganas, Michael Anderson, David Sheffield, Nick Knight, Tony DeRose, Mark Meyer, Dirk Van Gelder, and the OpenSubdiv team.

References

[ABC*06] Asanovic K., Bodik R., Catanzaro B. C., Gebis J. J., Husbands P., Keutzer K., Patterson D. A., Plishker W. L., Shalf J., Williams S. W., Yelick K. A.: *The Landscape of Parallel Computing Research: A View from Berkeley*. Tech. Rep. UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006. URL: `http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html`.

[BG09] Bell N., Garland M.: Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis* (New York, NY, USA, 2009), SC '09, ACM, pp. 18:1–18:11. URL: `http://doi.acm.org/10.1145/1654059.1654078`, `doi:10.1145/1654059.1654078`.

[BG12] Buluç A., Gilbert J. R.: Parallel sparse matrix-matrix multiplication and indexing: Implementation and experiments. *SIAM Journal of Scientific Computing (SISC) 34*, 4 (2012), 170 – 191. URL: `http://gauss.cs.ucsb.edu/~aydin/spgemm_sisc12.pdf`, `doi:10.1137/110848244`.

[BS02] Bolz J., Schröder P.: Rapid evaluation of catmull-clark subdivision surfaces. In *Proceedings of the seventh international conference on 3D Web technology* (New York, NY, USA, 2002), Web3D '02, ACM, pp. 11–17. URL: `http://doi.acm.org/10.1145/504502.504505`, `doi:10.1145/504502.504505`.

[CC98] Catmull E., Clark J.: Seminal graphics. ACM, New York, NY, USA, 1998, ch. Recursively generated B-spline surfaces on arbitrary topological meshes, pp. 183–188. URL: `http://doi.acm.org/10.1145/280811.280992`, `doi:10.1145/280811.280992`.

[Dav06] Davis T. A.: *Direct Methods for Sparse Linear Systems (Fundamentals of Algorithms 2)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2006.

[DKT98] DeRose T., Kass M., Truong T.: Subdivision surfaces in character animation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1998), SIGGRAPH '98, ACM, pp. 85–94. URL: http://doi.acm.org/10.1145/280814.280826, doi:10.1145/280814.280826.

[HDD*94] HOPPE H., DEROSE T., DUCHAMP T., HALSTEAD M., JIN H., MCDONALD J., SCHWEITZER J., STUETZLE W.: Piecewise smooth surface reconstruction. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1994), SIGGRAPH '94, ACM, pp. 295–302. URL: http://doi.acm.org/10.1145/192161.192233, doi:10.1145/192161.192233.

[HKD93] HALSTEAD M., KASS M., DEROSE T.: Efficient, fair interpolation using catmull-clark surfaces. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1993), SIGGRAPH '93, ACM, pp. 35–44. URL: http://doi.acm.org/10.1145/166117.166121, doi:10.1145/166117.166121.

[Loo87] LOOP C. T.: *Smooth subdivision surfaces based on triangles*. Master's thesis, University of Utah, Salt Lake City, Utah, USA, 1987.

[LSNCn09] LOOP C., SCHAEFER S., NI T., CASTAÑO I.: Approximating subdivision surfaces with gregory patches for hardware tessellation. *ACM Trans. Graph. 28*, 5 (Dec. 2009), 151:1–151:9. URL: http://doi.acm.org/10.1145/1618452.1618497, doi:10.1145/1618452.1618497.

[McC95] MCCALPIN J. D.: Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter* (Dec. 1995), 19–25.

[NLMD12] NIESSNER M., LOOP C., MEYER M., DEROSE T.: Feature-adaptive gpu rendering of catmull-clark subdivision surfaces. *ACM Trans. Graph. 31*, 1 (Feb. 2012), 6:1–6:11. URL: http://doi.acm.org/10.1145/2077341.2077347, doi:10.1145/2077341.2077347.

[Pix12] PIXAR ANIMATION STUDIOS: OpenSubdiv. http://graphics.pixar.com/opensubdiv/, 2012. Accessed: 2013-05-01.

[SJP05] SHIUE L.-J., JONES I., PETERS J.: A realtime gpu subdivision kernel. *ACM Trans. Graph. 24*, 3 (July 2005), 1010–1015. URL: http://doi.acm.org/10.1145/1073204.1073304, doi:10.1145/1073204.1073304.

[Sta98] STAM J.: Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1998), SIGGRAPH '98, ACM, pp. 395–404. URL: http://doi.acm.org/10.1145/280814.280945, doi:10.1145/280814.280945.

[VDY05] VUDUC R., DEMMEL J. W., YELICK K. A.: Oski: A library of automatically tuned sparse matrix kernels. In *Institute of Physics Publishing* (2005).

[WOV*07] WILLIAMS S., OLIKER L., VUDUC R., SHALF J., YELICK K., DEMMEL J.: Optimization of sparse matrix-vector multiplication on emerging multicore platforms. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing* (New York, NY, USA, 2007), SC '07, ACM, pp. 38:1–38:12. URL: http://doi.acm.org/10.1145/1362622.1362674, doi:10.1145/1362622.1362674.