Efficient Evaluation of Semi-Smooth Creases in Catmull-Clark Subdivision Surfaces

M. Nießner[1] and C. Loop[2] and G. Greiner[1]

[1]University of Erlangen-Nuremberg
[2]Microsoft Research

Abstract

We present a novel method to evaluate semi-smooth creases in Catmull-Clark subdivision surfaces. Our algorithm supports both integer and fractional crease tags corresponding to the RenderMan (Pixar) specification. In order to perform fast and efficient surface evaluations, we obtain a polynomial surface representation given by the semi-smooth subdivision rules. While direct surface evaluation is applied for regular patches, we perform adaptive subdivision around extraordinary vertices. In the end, we are able to efficently handle high-order sharpness tags at very low cost. Compared to the state-of-the art, both render time and memory consumption are reduced from exponential to linear complexity. Furthermore, we integrate our algorithm in the hardware tessellation pipeline of modern GPUs. Our method is ideally suited to real-time applications such as games or authoring tools.

1. Introduction

Catmull-Clark subdivision surfaces [CC78] are now standard in today's computer generated feature films. Semi-smooth creases [DKT98] as specified by RenderMan [Pix05] are an important extension that allow realistic edges to be defined while keeping memory footprint small. This is particularly useful when animating characters since only a small number of control vertices need to be updated. Hardware tessellation on today's GPUs makes the use of Catmull-Clark subdivision surfaces attractive for real-time applications. However, hardware tessellation requires direct surface evaluation rather than iterative refinement as provided by the subdivision rules. Nießner et al. [NLMD12] perform adaptive subdivision around features such as extraordinary vertices or semi-smooth creases and process the resulting nested regular bicubic patches with hardware tessellation. This provides better performance than uniform subdivision; however, at semi-smooth crease tags there is still an exponential growth in the number of patches being processed.

The key idea of our method is to analyze the polynomial structure of semi-smooth creases and directly evaluate these rather than applying iterative subdivision. While we directly evaluate regular patches with sharpness tags, we use adaptive subdivision around extraordinary vertices. This turned out to be faster than performing a Stam-like evaluation [Sta98] for irregular patches. Compared to previous feature adaptive subdivision, the number of patches being created by subdivision is only linear with respect to sharpness tags and tessellation density (instead of exponential). This results in a considerable performance gain and reduces memory consumption significantly (see Figure 2).

2. Previous Work

Catmull-Clark subdivision surfaces [CC78] are a generalization of regular bicubic B-splines to arbitrary 2-manifold control meshes. Nasri [Nas87] extended the Catmull-Clark scheme by introducing boundary (a.k.a. sharp) subdivision rules. These additional rules can also be applied to the edges of non-boundary patches in order to obtain creases [HDD*94]. Further, DeRose et al. [DKT98] introduced semi-sharp creases that allow modelling edges with tighter radii of curvature, without a significant increase of memory. A semi-sharp crease is defined by subdividing s times using sharp (i.e., boundary) subdivision rules, followed by subdivision using the standard rules (smooth rules).

The capabilites of modern GPUs allow for massively parallelized subdivision. This can be done by either using the graphics pipeline [SJP05] or using GPGPU APIs such as CUDA (e.g., [PEO09]). However, performance is limited by the high memory-to-compute ratio. Hardware tessellation

DOI: 10.2312/conf/EG2012/short/041-044

creates geometry on-chip and allows rendering with a minimal amount of memory I/O. While this is ideal in terms of performance, only direct parametric patch evaluation is supported.

A method to directly evaluate Catmull-Clark surfaces at arbitrary parametric values was developed by Stam [Sta98]. Its key idea is to perform an eigenvalue decomposition of the subdivision matrices to obtain a set of bicubic *eigenbasis functions* for Catmull-Clark patches. While orginally proposed for the CPU, it is easily mapped to the paradigm of hardware tessellation. Compared to approximate patching [LS08], or feature adaptive subdivision [NLMD12], the performance of Stam's evaluation procedure is poor. This can be attributed to its branching and heavy use of floating point computation.

Nießner et al. [NLMD12] adaptively subdivide around features such as extraordinary vertices or semi-smooth creases using GPGPU compute kernels in order to obtain regular patches exclusively. With the resulting patches being all regular, the tessellator can then be used to efficiently evaluate these patches. They have shown that their feature adaptive subdivision is significantly faster than Stam evaluation and any iterative subdivision scheme due to its minimal amount of memory I/O. However, adaptive subdivision creates an exponential number of patches at edges with semi-smooth creases, harming performance.

We combine feature adaptive subdivision at extraordinary points and direct evaluation for regular patches. This allows us to reduce the number of patches being created by adaptive subdivision from an exponential to a linear amount with respect to the number of subdivision steps. Compared to the original approach by Nießner et al., this gives a significant performance improvement (see Figure 2).

3. Evaluation of Semi-Smooth Creases

First, we assume that a patch contains at most a single semi-smooth crease and does not contain any extraordinary vertices. This is achieved by adaptive subdivision following Nießner et al. [NLMD12] (see Section 4). We evaluate bicubic B-spline patches using their tensor product form with parameters u and v. That allows us to simplify the problem to the curve case with a single semi-smooth crease tag. Our goal is to transform the control points of a cubic B-spline curve such that it exactly corresponds to the semi-sharp crease rules of Catmull-Clark subvidision defined by DeRose et al. [DKT98].

A uniform cubic B-spline curve can be refined applying the refinement matrix $\overline{R}$ (or $\overline{R_p}$ at a curve boundary):

$$\overline{R} = \frac{1}{8}\begin{pmatrix} 4 & 4 & 0 & 0 \\ 1 & 6 & 1 & 0 \\ 0 & 4 & 4 & 0 \\ 0 & 1 & 6 & 1 \\ 0 & 0 & 4 & 4 \end{pmatrix} \quad \text{and} \quad \overline{R_p} = \frac{1}{8}\begin{pmatrix} 4 & 4 & 0 & 0 \\ 1 & 6 & 1 & 0 \\ 0 & 4 & 4 & 0 \\ 0 & 0 & 8 & 0 \\ 0 & 0 & 4 & 4 \end{pmatrix}.$$

Thus, subdividing the initial curve control points $\vec{P} = (P_0, P_1, P_2, P_3)^T$ can be represented as $\vec{P'} = \overline{R}\vec{P}$. The matrices $\overline{R}$ and $\overline{R_p}$ correspond to the smooth and sharp Catmull-Clark subdivision rules, respectively.

We now split the cubic B-spline curve $f(t) = N(t)\vec{P}$ (with $N(t)$ a 1×4 matrix containing the cubic B-spline basis functions and $\vec{P}$ defining the B-spline control points) into two curve segments: the infinitely sharp segment $f_\infty(t)$ defined for $0 \leq t \leq 1 - 2^{-s}$ and the transition segment $f_s(t)$ to the crease defined for $1 - 2^{-s} < t \leq 1$.

$$f(t) = \begin{cases} f_\infty(t) = N(t)\vec{P}_\infty & \text{for } 0 \leq t \leq 1 - 2^{-s} \\ f_s(t) = N(t)\vec{P}_s & \text{for } 1 - 2^{-s} < t \leq 1 \end{cases}$$

In order to directly evaluate the curve we need to obtain the control points for both curve segments $\vec{P}_\infty$ and $\vec{P}_s$. The Catmull-Clark subdivision rules for boundaries can be transfered to the curve case using the transformation matrix

$$M_\infty = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 2 & 0 \end{pmatrix}.$$

The control points of the infinitely sharp section of the curve are then given by $\vec{P}_\infty = M_\infty \vec{P}$. For $f_s(t)$ we use modified refinement matrices R and R_p derived from $\overline{R}$ and $\overline{R_p}$:

$$R = \frac{1}{8}\begin{pmatrix} 1 & 6 & 1 & 0 \\ 0 & 4 & 4 & 0 \\ 0 & 1 & 6 & 1 \\ 0 & 0 & 4 & 4 \end{pmatrix} \quad \text{and} \quad R_p = \frac{1}{8}\begin{pmatrix} 1 & 6 & 1 & 0 \\ 0 & 4 & 4 & 0 \\ 0 & 0 & 8 & 0 \\ 0 & 0 & 4 & 4 \end{pmatrix}.$$

These reduced matrices still subdivide the curve, however, the set of resulting control points defines only a part of the curve. This corresponds to $f_s(t)$ which is only valid for $t \in]1 - 2^{-s}, 1]$.

The control points required to define $f_s(t)$ can be obtained by $R_p^s \vec{P}$. However, this results in a wrong parametrization since the curve's velocity is changed. Thus, we back-transform these interim control points using R^{-1} in order to maintain the original parametrization:

$$\vec{P}_s = (R^{-1})^s R_p^s \vec{P}$$

The resulting control points $\vec{P}_s$ define an extrapolated curve, however, with $t \in]1 - 2^{-s}, 1]$ that curve exactly matches the desired shape with the parametrization corresponding to the initial curve. Examining the eigenstructures of R and R_p (both non-defective) allows us to define $M_s = (R^{-1})^s R_p^s$ and diagonalize:

$$R_p^s = V_{R_p} \Lambda_{R_p}^s V_{R_p}^{-1} \quad \text{and} \quad (R^{-1})^s = V_R^{-1} \Lambda_R^{-s} V_R.$$

Thus, we obtain a simplified M_s (with $\sigma = 2^s$):

$$M_s = \frac{1-\sigma}{6\sigma}\begin{pmatrix} \frac{6\sigma}{1-\sigma} & 6\sigma^2-5\sigma+1 & -12\sigma^2+10\sigma-2 & 6\sigma^2-5\sigma+1 \\ 0 & \frac{2\sigma^2+3\sigma+1}{1-\sigma} & 4\sigma-2 & 1-2\sigma \\ 0 & \sigma+1 & \frac{2\sigma^2+6\sigma-2}{1-\sigma} & \sigma+1 \\ 0 & 1-2\sigma & 4\sigma-2 & \frac{2\sigma^2+3\sigma+1}{1-\sigma} \end{pmatrix}$$

Further, $f(t)$ is given by:

$$f(t) = \begin{cases} N(t)M_\infty \vec{P} & \text{for } 0 \le t \le 1-2^{-s} \\ N(t)M_s\vec{P} & \text{for } 1-2^{-s} < t \le 1 \end{cases}$$

Two such curves with sharpness 1 and 2 are shown in Figure 1. In the end we can directly evaluate regular patches with a single semi-smooth crease using $f(t)$ to construct the tensor product ($f_i(u)$ refers to the evaluation of one row of the 4×4 control points of a bicubic patch):

$$S(u,v) = (f_0(u), f_1(u), f_2(u), f_3(u))N^T(v)$$

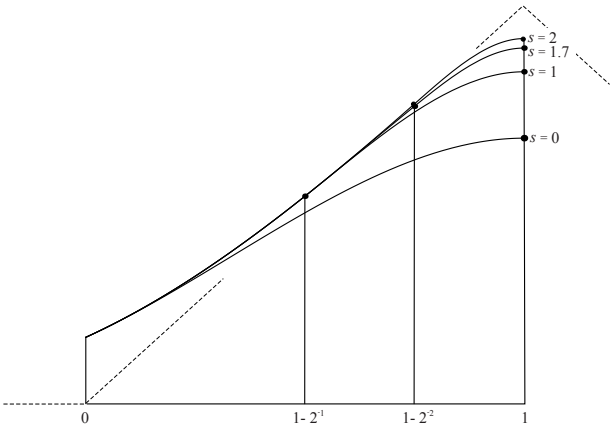

Figure 1: *Curve segments generated for a particular control polygon; the sharpness tags shown are $s = 0,1,1.7,2$. For $s = 0$ the curve is a single segment defined on $[0,1]$; for $s = 1,2$ the curves have two sements defined on $[0,1-2^{-s}]$ and $[1-2^{-s},1]$; for $s = 1.7$ the curve has three segments defined on $[0,1-2^{-\lfloor s\rfloor}]$,$[1-2^{-\lfloor s\rfloor},1-2^{-\lceil s\rceil}]$, and $[1-2^{-\lceil s\rceil},1]$*

3.1. Fractional Sharpness

DeRose et al. [DKT98] also specify fractional sharpness as a linear blend between integer sharpness levels. Hence, it is required to compute a separate $M_{\lfloor s\rfloor}$ and $M_{\lceil s\rceil}$ and apply the linear blend manually. This yields three curve segments, an infinitely sharp part, a linear blend between sharp and the smaller sharpness factor, and a linear blend between the smaller and higher sharpness factor; see Figure 1. These segments are defined by the respective control points $\vec{P}_\infty$, $\vec{P}_{\hat{\infty}}$ and $\vec{P}_{\hat{s}}$:

$$\vec{P}_\infty = M_\infty \vec{P}$$

$$\vec{P}_{\hat{\infty}} = (1-(s-\lfloor s\rfloor))(R^{-1})^{\lfloor s\rfloor}R_p^{\lfloor s\rfloor}\vec{P} + (s-\lfloor s\rfloor)M_\infty\vec{P}$$

$$\vec{P}_{\hat{s}} = (1-(s-\lfloor s\rfloor))(R^{-1})^{\lfloor s\rfloor}R_p^{\lfloor s\rfloor}\vec{P} + (s-\lfloor s\rfloor)(R^{-1})^{\lceil s\rceil}R_p^{\lceil s\rceil}\vec{P}$$

Multiple transform operations can be avoided by directly computing the required transformation matrices $M_{\hat{\infty}}$ and $M_{\hat{s}}$:

$$M_{\hat{\infty}} = (1-(s-\lfloor s\rfloor))M_{\lfloor s\rfloor} + (s-\lfloor s\rfloor)M_\infty$$

$$M_{\hat{s}} = (1-(s-\lfloor s\rfloor))M_{\lfloor s\rfloor} + (s-\lfloor s\rfloor)M_{\lceil s\rceil}$$

Note that these transformation matrices correspond to the semi-smooth subdivision rules with fractional sharpness tags. Now the initial control points are transformed and the resulting function $f(t)$ is given by the three curve segments:

$$f(t) = \begin{cases} f_\infty(t) = N(t)M_\infty\vec{P} & \text{for } 0 \le t \le 1-2^{-\lfloor s\rfloor} \\ f_{\hat{\infty}}(t) = N(t)M_{\hat{\infty}}\vec{P} & \text{for } 1-2^{-\lfloor s\rfloor} < t \le 1-2^{-\lceil s\rceil} \\ f_{\hat{s}}(t) = N(t)M_{\hat{s}}\vec{P} & \text{for } 1-2^{-\lceil s\rceil} < t \le 1 \end{cases}$$

Figure 1 shows an example curve with a fractional sharpness of 1.7. The computation of the tensor product surface $S(u,v)$ is the same as for integer sharpness.

4. GPU Implementation using Hardware Tessellation

Since hardware tessellation is fast and memory efficient, we use it for Catmull-Clark subdivision surface rendering. Nießner et al. [NLMD12] perform adaptive subdivision iteratively around extraordinary vertices (using GPGPU) and then directly evaluate the resulting bicubic B-spline patches using hardware tessellation. This turned out to be faster than Stam's direct evaluation method. The reason is that the number of adaptively created patches around extraordinary vertices is only linear with respect to the number of subdivision steps k ($3N\cdot k$ where N is the vertex valence). They also use adaptive subdivision around features such as semi-smooth creases, however this creates an exponential number of subdivided patches after k subdivisions (2^k).

We replace adaptive subdivision for creases in regular regions by direct evaluation as shown in Section 3. Adaptive subdivision is still applied at extraordinary vertices and semi-smooth creases with varying sharpness. It is also used to enforce the condition that regular patches must not contain more than one semi-smooth crease tag. The benefit of this is that the number of child patches created by adaptive subdivision becomes linear instead of exponential with respect to the sharpness tag.

The implementation for regular patches with one edge tagged sharp is straight forward. In the hull shader we obtain the sharpness according to the patch id and compute the respective transformation matrix M_s. Further, the transformed control points of the different curve segments of $f(t)$ are computed according to M_∞ and M_s, or in the fractional case according to M_∞, $M_{\hat{\infty}}$ and $M_{\hat{s}}$. The two resulting sets

(three with fractional sharpness tags) of control points are then passed to the domain shader, where we determine which set of control points is required in order to evaluate the subpatch according to the domain parameters u, v.

5. Results

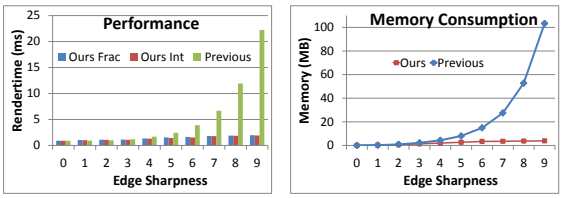

Figure 2: *Performance (rendered with hardware tessellation at a tess factor of* 8*) and memory consumption for the Car model (see Figure 3); our method (fractional and integer sharpness) and previous work (feature adaptive subdivision [NLMD12])*

Our implementation uses DirectX 11 running on an NVIDIA GeForce GTX 480. Figure 2 shows performance and memory consumption of our method (fractional and integer sharpness variant) and previous work (feature adaptive subdivision [NLMD12]) for the Car model (see Figure 3) with different sharpness tags. Rendering is performed using a tess factor of 8. Performance without any sharpness is the same for all methods since the same number of patches are being created by adaptive subdivision. Having set sharpness tags to 1 or 2 previous feature adaptive subdivision is marginally faster due to lower patch setup costs. At a sharpness above 2 our direct evaluation algorithm is faster. With higher sharpness tags, feature adaptive subdivision becomes significantly slower (note the exponential behaviour). In contrast, render time using our method remains almost constant. Render time with our method increases slightly since we still perform adaptive subdivision at extraordinary vertices. The same relation between our method and feature adaptive subdivision can be observed in terms of memory consumption, however, the memory consumption of our method is always less. A visualization of the different subdivision levels is provided by Figure 3 (left ours; right previous work). Also note that our method allows the modification of sharpness tags at runtime.

5.1. Conclusion

We have presented a novel and GPU-friendly method that allows efficient evaluation of semi-smooth creases as defined by the RenderMan specification [Pix05]. Our algorithm keeps render time low and memory I/O small, and thus allows dealing even with high-order sharpness tags. While we have demonstrated how to integrate direct evaluation of semi-smooth creases for hardware tessellation, our method

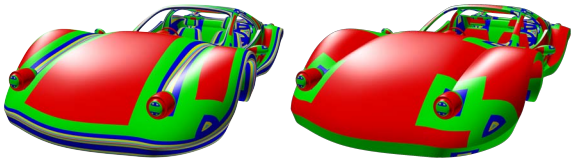

Figure 3: *Car model consisting of* 1519 *patches with* 314 *semi-smooth crease tags (sharpness of* 6*) rendered with feature adaptive subdivision (left) and our method (right). Subdivision levels are indicated by different colors.*

can be also used by offline renderers. For instance, subdivision surfaces with semi-smooth crease tags can be efficiently ray-traced without costly iterative sudivision.

References

[CC78] CATMULL E., CLARK J.: Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-aided design 10*, 6 (1978), 350–355. 1

[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* (1998), ACM, pp. 85–94. 1, 2, 3

[HDD*94] HOPPE H., DEROSE T., DUCHAMP T., HALSTEAD M., JIN H., MCDONALD J., SCHWEITZER J., STUETZLE W.: Piecewise smooth surface reconstruction. *SIGGRAPH '94* (1994), 295–302. 1

[LS08] LOOP C., SCHAEFER S.: Approximating Catmull-Clark subdivision surfaces with bicubic patches. *ACM TOG 27*, 1 (2008), 8:1–8:11. 2

[Nas87] NASRI A.: Polyhedral subdivision methods for free-form surfaces. *ACM TOG 6*, 1 (1987), 29–73. 1

[NLMD12] NIESSNER M., LOOP C., MEYER M., DEROSE T.: Feature adaptive GPU rendering of Catmull-Clark subdivision surfaces. *ACM TOG* (2012). 1, 2, 3, 4

[PEO09] PATNEY A., EBEIDA M. S., OWENS J. D.: Parallel view-dependent tessellation of Catmull-Clark subdivision surfaces. In *HPG '09: Proceedings of the Conference on High Performance Graphics 2009* (New York, NY, USA, 2009), ACM, pp. 99–108. 1

[Pix05] PIXAR ANIMATION STUDIOS: The RenderMan Interface version 3.2.1, 2005. (https://renderman.pixar.com/products/-rispec/index.htm). 1, 4

[SJP05] SHIUE L.-J., JONES I., PETERS J.: A realtime GPU subdivision kernel. *ACM TOG 24*, 3 (2005), 1010–1015. 1

[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* (1998), ACM, pp. 395–404. 1, 2