DOI: 10.1111/cgf.70346

EUROGRAPHICS 2026 / B. Masia and J. Thies
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 45 (2026), Number 2

EBOAT: Error-Bounded Adaptive Tessellation of Singularities for Real-Time Catmull-Clark Subdivision Surfaces Rendering

Yajun Zeng[1], Yang Lu[2], Cong Chen[2], Ruicheng Xiong[1], and Ligang Liu[2,*]

[1] University of Science and Technology of China
[2] Sheyun Technology
[*] Corresponding author

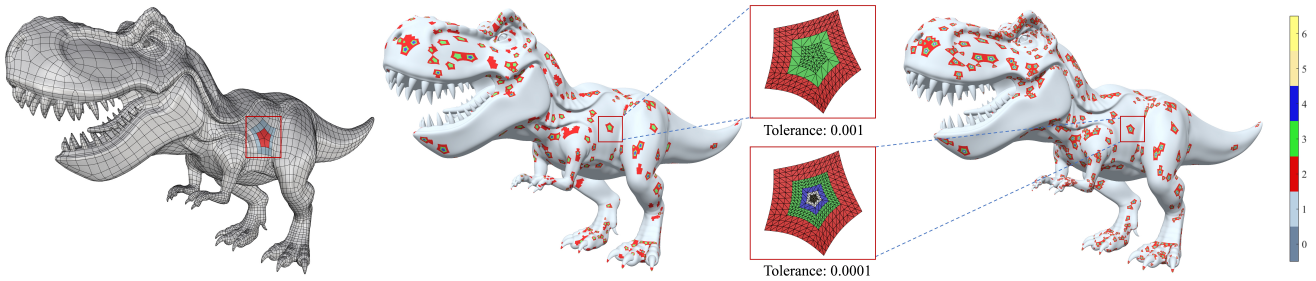


Figure 1: *We propose EBOAT, a novel memory-efficient and error-bounded tessellation method of singularities for Catmull-Clark subdivision surfaces. Singularities in the control mesh are converted into extraordinary rings using the proposed GPU-friendly representation. An extraordinary ring with a valence of 5 is highlighted in the red box within the control mesh rendering (left). At runtime, for each extraordinary ring, we estimate an adaptive subdivision depth and then subdivide the extraordinary ring to this depth using a high-performance feature adaptive subdivision algorithm based on the proposed representation. After subdivision to the estimated depth, the limit surface of the innermost 1-ring is approximated by bi-cubic Bézier patches. Given specified tolerance, this subdivision depth ensures that the error between the limit surface and the Bézier approximation meets a fixed proportion of tolerance, while the tessellation levels ensures that the error between tessellated meshes and Bézier approximation meets the remaining portion of the tolerance. With tolerances of 0.001 and 0.0001 times the minimum length of the bounding box (middle and right), the extraordinary ring is subdivided to different depths, and the resulting 1-ring quads (innermost quads in red boxes) are approximated as bi-cubic Bézier surfaces and then tessellated.*

Abstract

Tessellation is critical for real-time rendering of Catmull-Clark subdivision (CCS) surfaces, with control mesh singularities posing a major challenge. Prior works typically pre-subdivide singular regions to a fixed depth offline to facilitate efficient runtime tessellation, yet this practice fails to meet view-dependent error requirements for high-fidelity rendering—especially at pixel-level accuracy. Hence, we present EBOAT, a novel direct-evaluation tessellation method that integrates memory-efficient feature-adaptive subdivision via a connectivity-elided representation for singularities, along with a two-phase error control strategy and improved subdivision depth estimation built upon it. With a uniform tessellation level of 64, EBOAT achieves a $3.8\times$–$4.7\times$ *speedup over OpenSubdiv, without relying on fixed-depth pre-subdivision. Compared to state-of-the-art breadth-first subdivision, it yields a* $4.2\times$–$7.1\times$ *speedup while providing more accurate sampling. To further demonstrate its effectiveness, we implement a pixel-accurate CCS surface rendering pipeline with EBOAT fully integrated, and provide an in-depth evaluation of rendering quality. Given its advantages in both performance and accuracy, EBOAT offers a promising alternative for real-time CCS surface rendering in industrial applications.*

CCS Concepts

• ***Computing methodologies*** → ***Rendering; Parallel algorithms;***

1. Introduction

Catmull-Clark subdivision (CCS) surfaces [CC78; KOCM23; Pixar25] have been widely adopted for several decades in the domain of film production, industrial design, and game rendering. It is recognized for its ability to facilitate precise and organic modeling, as well as to expedite design processes [Rhino25]. With the increasing demand for higher geometry fidelity and richer images, there is a pressing need for real-time algorithms that can deliver efficient and accurate tessellation for high-quality CCS surface rendering, especially under pixel-level accuracy [XLC*23].

There are two main methods for tessellating a CCS surface: breadth-first subdivision and direct evaluation. The former produces meshes by applying CCS scheme iteratively on the control mesh with simple calculations [MWS*20; DV21; KOCM23], yet suffers from limited adaptivity due to global subdivision. Some methods [HW07b; PW09] can ensure that the refined mesh meets a given error bound relative to the limit surface, but directly combining these methods with global subdivision results in deep subdivision depths and an excessive number of redundant triangles. On the other hand, direct evaluation methods perform direct and accurate sampling on the underlying limit surface, generating tessellated meshes with a closer approximation to the true limit surface compared to the control meshes used in breadth-first subdivision methods. These methods demonstrate satisfactory performance and improved geometric fidelity during runtime, yet they typically require creating subdivision or stencil tables, which rely on subdividing the model to a pre-specified depth in an offline manner [NLMD12; SRK*15; BFK*16]. Despite this limitation, the flexibility and adaptability of direct evaluation make it a preferred choice for real-time, error-bounded rendering of CCS surfaces.

However, the use of subdivision or stencil tables in direct evaluation methods constrains tessellation to a pre-specified subdivision depth, limiting rendering accuracy and geometric fidelity, particularly in view-dependent and pixel-accurate rendering scenarios [XLC*23]. The key challenge causing this issue lies in the intricate process handling singular quads, which include extraordinary vertices [DS78], boundaries [Nas87], or infinitely or semi-sharp creases [HDD*94]. OpenSubdiv, an industry standard implementation of CCS surface evaluation [Pixar25], utilizes Feature Adaptive Subdivision (FAS) to process singularities [NLMD12]. [SRK*15] introduced dynamic FAS to allow different subdivision depths for each singularity. Subsequent research has demonstrated improved performance over FAS by establishing subdivision plans using adaptive quadtrees [BFK*16] or formulating it as algebraic operations on sparse matrices [MWS*20]. Despite efforts to enhance performance, these methods remain limited by a fixed maximal subdivision depth, preventing support for arbitrary precision control. If the pre-subdivision depth fails to meet the required accuracy, costly on-the-fly subdivision becomes necessary, incurring computational expenses that are often orders of magnitude higher than current cost[MWS*20], thereby rendering real-time performance unattainable.

Additionally, some works aim to estimate a tight subdivision depth to control the distance between the limit surface and its linear approximations, such as the control mesh or the limit mesh (obtained by pushing each control vertex to its limit position) [PW09; HDW08]. However, few studies consider introducing an intermediate nonlinear approximation of the limit surface, such as bi-cubic Bézier patches [LSNC09], to manage the overall error. This omission is reasonable, as such intermediate approximation should ultimately be discretized into meshes for rendering, thereby introducing a two-layer error. Nevertheless, we observe that on modern GPUs, carefully exploiting the trade-off between these two layers of error offers significant potential for improving performance.

The key challenges in achieving real-time, error-bounded (especially pixel-accurate) rendering lie in the needs for tight and reliable depth estimation to meet strict view-dependent tolerances, and for high-performance, on-the-fly subdivision algorithms capable of handling singularities at arbitrary depths. To this end, we propose a novel tessellation method, named EBOAT, for CCS surfaces in the direct-evaluation category. It features a memory-efficient feature-adaptive subdivision (FAS) scheme via a connectivity-elided representation (CER) of singularities, along with a two-phase error control strategy and subdivision depth estimation built upon it. In proposed representation, singularity vertices are stored in a ring buffer with connectivity implicitly encoded in vertex indices, allowing on-the-fly reconstruction via lightweight ALU operations and maximizing performance. Additionally, our pipeline decomposes tessellation into singularity processing and triangulation, with errors separately bounded in each phase; exploiting this trade-off further boosts performance. We also provide an in-depth evaluation of rendering quality. Overall, EBOAT provides an efficient solution for adaptive tessellation of CCS surfaces under arbitrary error bounds, achieving improved performance while eliminating the limitations of fixed subdivision depths and insufficient rendering accuracy.

- A GPU-driven workflow enabling real-time, error-bounded rendering of CCS surfaces by addressing the challenges posed by singularities through error-bounded adaptive tessellation that avoids pre-specified depth limits and supports pixel-accurate rendering.
- An on-the-fly CER-based FAS algorithm, combined with a two-phase error control strategy and improved depth estimation built upon it, achieving significant performance improvement via GPU-friendly memory access patterns.

By utilizing CER-based FAS, we achieve a 3.8×–4.7× speedup over OpenSubdiv under typical tessellation levels, without the constraints of pre-specified depths. Additionally, we observe a 4.2×–7.1× speedup compared to state-of-the-art breadth-first subdivision methods, while providing more accurate sampling. We also demonstrate the integration of our tessellation approach with error-bound accuracy and GPU memory efficiency, into a real-time pixel-accurate rendering pipeline, along with evaluation of rendering quality.

2. Related Work

2.1. Efficient Catmull-Clark Subdivision

Significant and ongoing progress has been made in achieving efficient real-time rendering of CCS surfaces using GPUs over the past decade. We separate these approaches into two categories: breadth-first subdivision and direct evaluation.

Breadth-First Subdivision. Breadth-First Subdivision applies subdivision rules in parallel across the entire control mesh at a given subdivision depth [SJP05; PEO09; MWS*20; DV21; VD22; WC23; KOCM23]. The resulting meshes, though deviating from the true limit surface, are then directly used for rendering. Previous works primarily focused on enhancing parallelism and memory efficiency with novel mesh data structures, including look-up table [SJP05], data buffer [PEO09], sparse mesh matrix [MWS*20], halfedge mesh [DV21; VD22], and Edge-Friend [KOCM23]. Currently, these methods offer efficient parallel schemes [KOCM23]. However, because they perform subdivisions globally, they generally do not handle crack-free adaptive subdivisions [DV21; KOCM23].

Direct Evaluation. Direct evaluation involves exact evaluation of the underlying limit surface. [Sta98] presented exact evaluation of the quads with isolated extraordinary vertices by utilizing the eigenstructure of the subdivision matrix. [NLMD12] proposed feature adaptive subdivision (FAS) algorithm, combining direct evaluation on regular patches with recursive subdivision on irregular patches. Exact evaluation on regular patches with a single semi-sharp crease was provided in [NLG12]. Dynamic FAS was further presented to allow different subdivision depths for each singularity [SRK*15]. [BFK*16] pointed out that the FAS process is typically decoupled from the rasterization pipeline as preprocessing due to its high overhead. [BFK*16] employed an adaptive quadtree to create subdivision plans to bypass the real-time FAS process. An alternative approach, formulating the FAS process as algebraic operations on sparse matrices, was presented by [MWS*20]. Our method falls within this category. However, previous methods incur frequent memory transfer between CPU and GPU, or frequent fetching of stencils or connectivity information from GPU device memory. In contrast, our proposed memory-efficient FAS based on CER restricts the memory footprint to the fast GPU shared memory, significantly enhancing performance.

2.2. Error-Bounded Approximation

Alternative approaches to improving tessellation performance directly approximate singularities with simpler surfaces. Representative methods include PN-triangles [VPBM01], quad splitting into four triangles [NYM*08; MNP08], bi-cubic Bézier patches [LS08], perturbed bi-cubic splines [MYP08], and Gregory patches [LSNC09]. While these techniques emphasize rendering efficiency, they largely neglect approximation error bounds, and the resulting meshes may deviate from the intended CCS model, potentially dissatisfying artists [NKF*16]. Some other works estimate a shallow subdivision depth to bound the distance between the limit surface and linear approximations, such as the control mesh [CY06; CCY06; HW07a; HW07b; PW09] or the limit mesh [HDW08]. Our work extends this line of research with a two-phase methodology: first approximating singularities using bi-cubic Bézier patches, then discretizing them into triangles, while ensuring that the final mesh remains within a specified error bound relative to the limit surface.

2.3. Real-time Tessellation

Rather than pre-tessellating surfaces into static meshes, real-time tessellation has increasingly matured due to its on-the-fly geometry generation, which enables view-dependent adaptivity and supports arbitrary precision control [NKF*16]. While hardware tessellation, introduced with the Direct3D 11 API [Tessellation23], provides a versatile and efficient approach for surface tessellation and rendering, a new programmable geometric shading pipeline built on Mesh Shaders (and optionally Task Shaders) was introduced in NVIDIAs Turing architecture to further enhance the flexibility and efficiency of the geometry pipeline [Turing18]. Building upon this, recent research has demonstrated that software-based implementations leveraging the Mesh Shader pipeline can yield superior performance by fully exploiting GPU architectural features, such as shared memory and optimized parallel designs [XLC*23; ZLCL25]. Our work adopts a similar CUDA-to-Mesh-Shader pipeline for real-time tessellation following [XLC*23; ZLCL25].

3. Problem and Overview

3.1. Problem and Methodology

Problem Statement. Our objective is to generate a tessellated mesh of Catmull-Clark Subdivbision (CCS) surfaces such that the distance between the mesh and the limit surface is within a given tolerance ϵ, while also maximizing performance. The tolerance ϵ should be view-dependent for pixel-accurate rendering, which requires that the geometric deviation from the tessellated meshes and the exact limit surface, when projected to screen space, be bounded by at most half a pixel.

Motivation. For regular quads of CCS surfaces, whose limit surfaces are bi-cubic Bspline patches, [XLC*23] proposed an efficient approach using Tensor Cores and image-space crack filling to ensure pixel accuracy. However, regions with singular features remain the primary challenge for real-time, pixel-accurate CCS rendering, due to the complex limit surface geometry, which hinders efficient error-bounded adaptive tessellation under view-dependent tolerances. As shown in Table 1 of the supplementary material, singularities contribute up to 48%–88% of the total rendering time on common models under uniform subdivision by OpenSubdiv [Pixar25]—and this overhead grows significantly under strict pixel-accurate adaptive tessellation. Note that this excludes the cost of stencil table construction. When the pre-subdivision depth fails to meet the required accuracy, on-the-fly subdivision becomes prohibitively expensive, making real-time performance unattainable. This issue is particularly pronounced under view-dependent error control, where zooming in may demand arbitrarily high subdivision depths to satisfy the tolerance. This motivates the development of efficient on-the-fly subdivision capable of reaching arbitrary depths, together with tighter depth estimation guided by view-dependent tolerances.

Our idea. To address the challenges posed by singularities, we proceed along two directions. First, we adopt a memory-efficient representation that implicitly encodes connectivity as vertex offsets, enabling efficient on-the-fly subdivision around singularities. During FAS, connectivity can be reconstructed through lightweight ALU operations, thereby significantly reducing memory access latency—crucial for high-performance GPU algorithms. Second, we introduce a two-phase strategy to ensure error-bounded tessellation. The prescribed tolerance ϵ is split into $\eta\epsilon$ and $(1-\eta)\epsilon$,

where $\eta \in (0,1)$ serves as a trade-off parameter. Improved depth estimation guarantees that the limit surfaces of the innermost 1-ring are approximated by bi-cubic Bézier patches within $\eta\epsilon$, and subsequently tessellated within $(1-\eta)\epsilon$. This two-phase strategy reduces the required subdivision depth by offloading part of the geometric approximation to a more efficient evaluation stage using Tensor Cores, while avoiding overly fragmented patches.

3.2. Overview

Pre-process Given a Catmull-Clark control mesh with common extensions including boundaries [Nas87], hard and semi-sharp creases [HDD*94; DKT98], we first perform a global subdivision to convert it into a quad-only mesh. Apart from regular cases (see Figure 2(a)) and typical singular cases (see Figure 2(b)), other complex cases — though rare on CCS surfaces — are resolved via local recursive subdivision and reduced to the canonical forms shown in Figures 2(a) and 2(b). We demonstrate in Section 2 of the supplementary material how all such cases are handled or transformed in our workflow.

In our workflow, regular quads can be directly converted into bi-cubic Bézier patches and tessellated using a method akin to [XLC*23]. We focus on the more challenging case of quads surrounding singularities, with representative forms shown in Figure 2(b). Each such case includes all quads directly surrounding an isolated extraordinary vertex, as well as their adjacent quads. We refer to this structure as an *extraordinary ring* in the following.

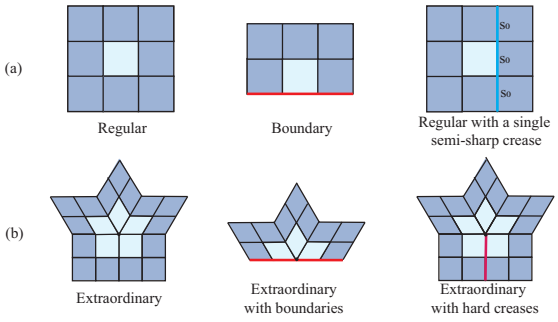


Figure 2: *During preprocessing, we process each case in the CCS control mesh into regular cases by [NLG12] (a) and typical singular cases referred to extraordinary ring (b). Other complex cases are converted into either regular or irregular input. Thick blue lines indicate semi-sharp creases with sharpness s_0 or s_1, while thick red lines represent hard creases or boundaries. Subsequently, we adopt the standard form of an extraordinary ring (first column of (b)) for analysis; other cases, such as those involving a boundary or a hard crease, are handled in the same manner.*

At runtime, the initial step is to determine a tolerance ϵ, either defined by the user or obtained via pixel accuracy [YBP12; XLC*23]. Then the *extraordinary ring* is processed as illustrated in Figure 3 below, following these steps:

1. *Depth Estimation.* In this step, we compute a subdivision depth for each *extraordinary ring* to ensure it can be approximated with Bézier surfaces within the tolerance $\eta\epsilon$, where $\eta \in (0,1)$. (see Section 5).
2. *CER-based FAS* Vertices of the *extraordinary ring* are arranged into connectivity-elided representation (CER), which enables high-performance FAS to the estimated depth. (see Section 4).
3. *Singularity Approximation.* Post FAS, the limit surfaces of resulting 1-ring quads are approximated using bi-cubic Béziers. (see Section 5).
4. *Error-bounded Tessellation.* In this step, the original regular patches and the losslessly generated bi-cubic B-spline from FAS are tessellated under the tolerance ϵ, while the innermost approximated bi-cubic Bézier patches are tessellated within $(1-\eta)\epsilon$. (see Section 6).

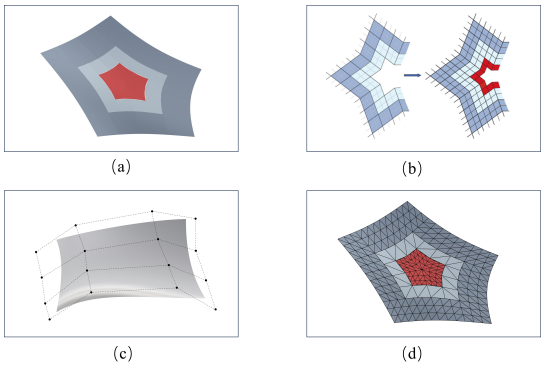


Figure 3: *Singularities Processing. (a) Depth estimation. (b) CER-based FAS. (c) Approximate the limit surfaces of inner-most 1-ring by Bézier. (d) Tessellate the resulting patches within given tolerance ϵ or $(1-\eta)\epsilon$.*

4. CER-based FAS Algorithm

In this section, we demonstrate how our GPU-efficient design enables high-performance on-the-fly FAS from two perspectives: (i) a memory-efficient representation and (ii) an efficient FAS implementation built upon it, followed by an experimental evaluation and discussion of performance.

4.1. Connectivity-elided Representation

4.1.1. Design and Insight

The proposed representation stores vertices of the *extraordinary ring* in a plain, ordered one-dimensional array and places them into a ring buffer, as illustrated in Figure 4. While such an organization is common in prior work, what sets our representation apart is that it is guided by considerations of GPU memory efficiency: rather than storing connectivity explicitly, CER encodes it implicitly in vertex indices, enabling lightweight on-the-fly reconstruction via ALU operations and thereby reducing global memory traffic. Here we use the term "*elide*" to demonstrate that connectivity information is deliberately omitted from memory storage for GPU memory efficiency, and is instead reconstructed on demand at negligible computational cost.

By contrast, previous work has also stored these vertices as

arrays but typically retained connectivity information in various forms (e.g., lookup tables [SJP05] or sparse structures [MWS*20]). Such connectivity data cannot fully fit into the on-chip memory of GPUs, and thus subdivision requires frequent reads and writes to global memory, incurring substantial access overhead. As noted by [DV21], the subdivision process is generally memory-bound, making the reduction of memory footprint and access latency critical for achieving high-performance on-the-fly subdivision.

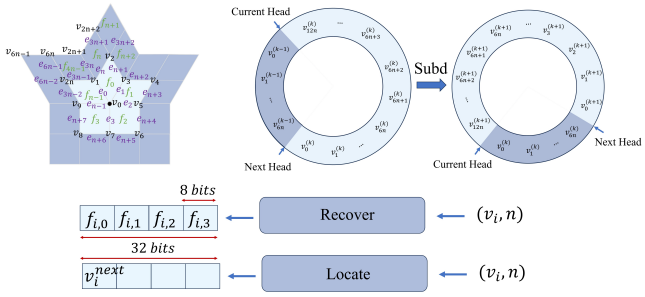


Figure 4: *CER of an extraordinary ring (valence n=5, totaling $6n+1$ vertices). We use a Ring buffer to storage vertices, where $v_0^{(k-1)}, v_1^{(k-1)}, \ldots, v_{6n}^{(k-1)}$ denote the control vertices of an extraordinary ring at depth $k-1$, and $v_0^{(k)}, v_1^{(k)}, \ldots, v_{12n}^{(k)}$ the refined vertices obtained after one subdivision step. At the bottom, we illustrate the basic operations for recovering connectivity or locating output offsets using vertex indices in the ring buffer. These operations are implemented as lightweight ALU instructions and executed on the fly.*

4.1.2. Implementation of CER

For an *extraordinary ring* with valence n, we set the central extraordinary vertex as v_0, and index the surrounding vertices sequentially in counterclockwise order: $v_1, v_2, \ldots, v_{6n}$ (Figure 4).Actually, once n is given, and a fixed ordering of the vertices, edges, and faces of the *extraordinary ring* is established, all connectivity is determined solely by their indices.

At runtime, connectivity information is extracted through *recovering* operations. To support the CCS scheme, five operators comprising simple bit-wise and arithmetic instructions are defined as follows (see Section 3 of supplementary material for detailed definitions):

- $face_v(v_j)$: Return indices of adjacent faces around the vertex v_j, where $j = 0, \ldots, 2n$.
- $face_e(e_j)$: Return indices of two adjacent faces sharing the edge e_j, where $j = 0, 1, ..., 6n-1$.
- $vertex_v(v_j)$: Return indices of vertices linking to the vertex v_j, where $j = 0, \ldots, 2n$.
- $vertex_f(f_j)$: Return indices of vertices on the face f_j,where $j = 0, \ldots, 4n-1$.
- $vertex_e(e_j)$: Return indices of vertices on the edge e_j, where $j = 0, 1, \ldots, 6n-1$.

It is worth noting that for an *extraordinary ring*, the number of valid vertices in each subdivision step is at most $12n+1$. Hence, when the valence satisfies $n \leq 21$, the indices can typically be represented with 8 bits. Except for the adjacency of the central extraordinary vertex (which is trivial), all other adjacency information can therefore be stored within four 8-bit values, i.e., a single `uint32_t`. For rare cases with $n > 21$, where the valence is particularly large, we can handle them separately by using 16-bit indices. With this scheme, all operations have well-structured inputs and outputs, and can be computed efficiently using bitwise and arithmetic instructions.

4.2. CER-based FAS

Considering the locality characteristic of the subdivision rule, in our implementation, each *extraordinary ring* is assigned to a warp of GPU, which executes its complete FAS process to the desired depth, facilitating a reduction in the total number of load operations on GPU device memory.

Furthermore, we formulate the output locations of newly generated vertices to ensure that: 1) all the vertices are stored in a plain array; 2) all vertices involved in the next subdivision are positioned at the beginning of the array, while maintaining the same relative topological order as in the previous level. This approach avoids memory accesses associated with vertex reordering between consecutive subdivision iterations, thereby maintaining a minimal memory footprint throughout the entire FAS process. Based on the CCS rules, we define three output *locating* operators:

- $locate_v(v_j)$ return the offset of new vertex $v_j, j = 0, 1, \ldots, 2n$ after refining.
- $locate_e(e_j)$ return the offset of the new edge point generated from edge $e_j, j = 0, 1, \ldots, 6n-1$.
- $locate_f(f_j)$ return the offset of new face point generated from face $f_j, j = 0, 1, \ldots, 4n-1$.

4.2.1. Implementation of CER-based FAS

In fact, the vertex update rules used in our FAS implementation are identical to those in [NLMD12]. The key difference lies in how data are organized and accessed during the subdivision process.

Specifically, we implement our memory-efficient FAS directly in a ring buffer, as shown in Figure 4. This buffer is allocated in GPU shared memory and is capable of accommodating both the CER and the refined vertices generated during subdivision. Since explicit connectivity information is not stored, the adjacency relationships required for vertex, edge, and face point computations are reconstructed on the fly using the defined *recovering* operations. After a vertex is computed, its relative location in the ring buffer is determined using the corresponding *locating* operations, which compute output offsets deterministically based on vertex indices and the valence of the ring. This design avoids any table lookups or pointer-based connectivity traversal, enabling efficient on-the-fly subdivision with minimal memory overhead.

Before executing FAS, the CER data is loaded into shared memory, and the head offset of ring-buffer is initialized to zero. Subsequently, subdivision is repeatedly performed until the specified depth is reached, where in each subdivision, regular quads are converted into bi-cubic B-spline surfaces and written into GPU device memory, and the head offset of the ring-buffer is advanced by

$6n+1$. Lastly, the limit surfaces of resulting 1-ring quads are approximated as Bézier surfaces and written to GPU device memory.

4.3. Discussion

Table 1 presents a comparative analysis of the time consumption (in ms) of our CER-based FAS against FAS proposed by [NLMD12] and SLAK FAS (recursive) [MWS*20] when subdividing 100K *extraordinary rings* to depths of 4, 5, and 6. The reported time consumption for FAS in [NLMD12] includes only the GPU subdivision time performed via Direct3D11 compute shaders, excluding the preprocessing time required to construct the subdivision tables. The timing for SLAK FAS (recursive) is based on the implementation that recursively subdivides around singularities, as described in the open-source code of Section 5.2 of [MWS*20]. [MWS*20] also proposed FAS by a single sparse matrix-vector multiplication (spMV). However, since no open-source code for this spMV approach is available, we do not provide a direct time comparison here. The experimental results for this spMV approach shown in [MWS*20] indicate a 1.7× speedup over FAS [NLMD12].

Among the aforementioned methods, both SLAK FAS (recursive) [MWS*20] and our approach do not rely on a pre-specified subdivision depth. In contrast, the methods presented in [NLMD12] and SLAK spMv [MWS*20] depend on a fixed subdivision depth to construct subdivision tables or matrices. Experimental results demonstrate that our method achieves 7×–14× speedup over FAS by [NLMD12], and greater speedup over SLAK FAS (recursive) [MWS*20]. The significant improvement on performance can be attributed to our GPU-efficient processing pattern, including our memory-efficient representation and its capability to effectively extract connectivity through the decoding and output addressing operator to reduce memory access.

Valences	3	5	6	7	8	9	10
FAS	2.67	3.75	4.61	5.32	5.59	8.62	9.37
SLAK (Recursive)	14.07	17.96	19.63	21.37	22.80	25.03	26.84
CER-based FAS	0.38	0.42	0.55	0.54	0.61	0.70	0.77

(a) Depth = 4

Valences	3	5	6	7	8	9	10
FAS	3.54	4.78	5.41	6.18	6.71	12.66	13.72
SLAK (Recursive)	15.52	20.97	22.85	24.88	26.77	29.69	31.88
CER-based FAS	0.48	0.57	0.63	0.68	0.75	0.92	0.94

(b) Depth = 5

Valences	3	5	6	7	8	9	10
FAS	4.11	5.72	6.32	7.01	7.77	13.97	15.41
SLAK (Recursive)	17.87	23.32	25.59	28.63	30.96	33.43	36.98
CER-based FAS	0.55	0.60	0.73	0.79	1.04	1.17	1.14

(c) Depth = 6

Table 1: *Time consumption (in ms) of our CER-based FAS, FAS by [NLMD12], and SLAK FAS (Recursive) [MWS*20] under various combinations of valence and subdivision depth. In all methods, 100K extraordinary rings around extraordinary vertices are recursively subdivided, with each subdivision iteration directly out-putting 12n + 1 refined vertices to GPU device memory.*

5. Error-bounded Tessellation

In this section, we present our method for error-bounded tessellation. The main challenge is to control the deviation between the tessellated mesh and the true limit surface near singularities, where the limit surface is defined by an infinite sequence of recursively refined B-spline rings.

Unlike previous methods that estimate subdivision depths to directly bound the distance between the limit surface and the tessellated mesh [HDW08; PW09], we introduce an intermediate nonlinear approximation surface (bi-cubic Bézier in our implementation), that mediates error control between the limit surface and the final tessellated mesh. This naturally introduces two layers of error, which might conventionally be regarded as undesirable. Nevertheless, we claim that, on modern GPU architectures, carefully exploiting the trade-off between these two error layers yields significant performance gains.

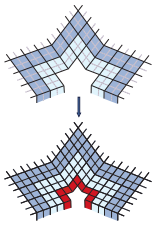
Here we must point out that neither the concept of two-phase evaluation [Sta98] nor the use of bi-cubic Bézier approximation [LSNC09] around singularities is new. Prior approaches typically subdivide to a fixed depth and then apply Bézier or Gregory patches without rigorous error control.

Our key insight is that the two-phase formulation can be systematically exploited on modern GPUs, since the two phases affect efficiency in different ways. By introducing bi-cubic Bézier approximation in the first phase, we significantly reduce the subdivision depth, thereby lowering the number of generated patches and avoiding excessive fragmentation. In the second phase, these Bézier patches are evaluated with high efficiency by leveraging Tensor Core acceleration. This synergy between reduced subdivision complexity and accelerated patch evaluation underpins the high performance of our error-bounded tessellation framework.

5.1. Two-Phase Error Control

In this subsection, we present the details of our two-phase error control strategy.

As outlined in Section 3, our approach handles *extraordinary rings* by using FAS and bi-cubic Bézier approximation. Based on the fact that recursive subdivisions around singularities (namely, extraordinary vertex here) can gradually fill the n-sided holes as shown in the inset figure on the right. After m subdivision steps, the innermost n-sided holes are filled with a series of Bézier surfaces. We reformulate the problem of finding the error-bounded tessellation of a given CCS surface as follows:



Given a user-specified tolerance ϵ, we decompose it into two parts: $\eta\epsilon$ and $(1-\eta)\epsilon$, where $\eta \in (0,1)$ is a weight coefficient. The first phase seeks the minimal subdivision depth m which control the approximation error between the limit surface x around singularities and its bi-cubic Bézier approximation $\hat{x}^m$, ensuring

$$dist(x,\hat{x}^m) < \eta\epsilon. \quad (1)$$

The approximation of the innermost 1-ring is then constructed following the method of [LSNC09].

In the second phase, we find a triangular tessellation l^m of these bi-cubic Bézier patches $\hat{x}^m$ such that

$$dist(l^m, \hat{x}^m) < (1-\eta)\epsilon. \tag{2}$$

Thus, the tessellation l^m of CCS surfaces surrounding singularity satisfies:

$$dist(l^m, x) \leq dist(l^m, \hat{x}^m) + dist(\hat{x}^m, x) < \epsilon. \tag{3}$$

Pixel-accurate implementation. For pixel-accurate rendering, the geometric deviation from the tessellated meshes and exact limit surface is required to be bounded by at most half a pixel when projected to screen space. Our implementation adopts the error control strategy of [XLC*23]: the screen-space tolerance is projected back into object space, and error control is performed following [FMM86; ZS00]. It is worth noting that leveraging Subdividable Linear Efficient Function Enclosures (SLEFEs) [YBP12] can provide tighter error control. Although [XLC*23] pointed out that this method may be theoretically problematic for rational surfaces, we observe that its formulation appears valid for CCS surfaces, which are non-rational. Nevertheless, to maintain compatibility with potential alternatives for singularity approximation, such as Gregory patches, our current implementation follows the route of [XLC*23], while leaving the integration of [YBP12] as future work.

Weight Coefficient. Since the error bound is decomposed by a weight coefficient η, its choice directly influences both subdivision and tessellation. A larger η reduces subdivision depth but increases tessellation levels on Bézier patches, while a smaller η has the opposite effect. Hence, η must be chosen to balance subdivision and tessellation performance. As shown in Figure 5, we evaluate tessellation time under different values of η using $\eta = 0$ (i.e., the single-phase strategy, the limit surfaces of inner-most 1-ring are directly approximated by triangle pairs) as baseline. The results indicate that values of η around 0.5 achieves the best balance, yielding a 16%–44% speedup over the single-phase strategy.

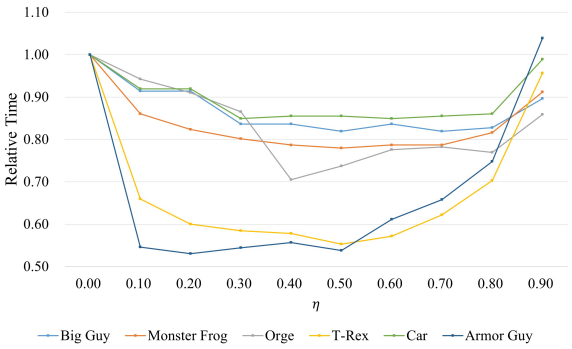


Figure 5: *The relative tessellation time for singularities on test models under different weight coefficients η, using $\eta = 0$ (i.e., the single-phase strategy) as the baseline.*

5.2. Depth Estimation

In this subsection, we focus on establishing a tight depth estimation for the first-phase error control in Eq. (1), whereas the second-phase error control is relatively straightforward and follows [FMM86; ZS00].

To keep the presentation concise and avoid overwhelming notation and theoretical details that may obscure the main idea, we focus here on the core intuition behind our depth estimation approach. The complete derivations, notations, and theoretical proofs are provided in Section 4 of the supplementary material.

Our method is inspired by [PW09], which achieves tight depth estimation by controlling the distance between the limit surface and the control mesh. We extend this idea from linear control-mesh approximation to non-linear approximation surfaces (bi-cubic Béziers). The tightness of their estimation primarily stems from reparameterization via the inverse of the characteristic map χ defined in [Rei95]. Under this mapping, the convergence of the geometric distance with respect to subdivision depth m is generally proportional to the sub-sub-dominant eigenvalue [PR08; PW09]. However, computing with χ^{-1} is cumbersome, as it requires inverting polynomial mappings [BZ04; PR04]. Although [Rei95] elegantly exploited structural properties of the control mesh to bypass this issue, extending such techniques to more general approximation surfaces remains problematic.

To solve this problem, instead of attempting to derive an explicit form of $\phi = \chi^{-1}$, we propose an alternative strategy that extends Bézier approximation while retaining the same convergence rate. We prove that, as long as the distance defined by ϕ is well-defined, the difficulty of computing χ^{-1} can be circumvented by precomputing a series of coefficients offline, which are then used to enable efficient error estimation at runtime.

The final formula for depth estimation in our method is given by

$$dist(x, \hat{x}_m) \leq c_m \|Q_0\|_\infty, \tag{4}$$

where $Q_0 = \{q_0, q_1, \ldots, q_{L-1}\}, L = 6n+1$ denotes the vertices of the *extraordinary ring* with valence n. These vertices are assumed to have been transformed into a local coordinate system centered at central point q_0. Besides, the coefficient $c_m = \sum_{l=0}^{L-1} \delta_{l,m}$ represents the sum of a series of precomputed values that depend only on the valence n and the subdivision depth m. And $\|\cdot\|_\infty$ is the ℓ_∞ norm, then $\|Q_0\|_\infty = \max_{l=0,1,\ldots,L-1} \|q_l\|_\infty$.

We compare our depth estimation method with existing approaches, as shown in Figure 6. The results demonstrate that our method achieves the tightest depth estimation while incurring low computational cost. This improvement arises from generalizing the results of [PR08; PW09] to bi-cubic Bézier surfaces, rather than relying on the control mesh or the limit mesh, thereby ensuring consistently fast convergence while providing a closer approximation to the limit surface.

6. Implementation and Results

We implement EBOAT using CUDA, and integrate it with a Vulkan forward rendering pipeline based on Mesh Shaders under the Blinn-Phong reflection model at a resolution of 1920×1080. All tests are

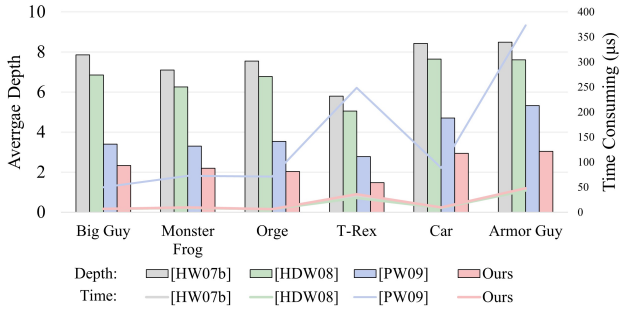


Figure 6: *Average estimated depths and estimation time consumption (in µs) of [HW07b], [HDW08], [PW09] and our method. The target tolerance is set to 0.005 of the minimal length of the bounding box of models.*

conducted on an Nvidia RTX 5070Ti desktop. For the final rendering comparisons, we additionally evaluate our method on a desktop with an NVIDIA RTX 4070 Super. The implementations of the compared algorithms are all based on the latest release of the authors' open source code.

Models setting. All test models have been utilized in previous research, with permissions granted by their owners for use in this work. The basic information of these models is provided in Table 2 of the supplementary material. To avoid data bias due to small model sizes, Big Guy, Monster Frog, Orge, Car are duplicated 10 times, while T-Rex, Armor Guy are duplicated 5 times.

6.1. Experimental Results

We first test the tessellation time spent on singularities by Opensubdiv viewer [Pixar25] across different models, experimental results are given in Table 1 of the supplementary material. The results indicate that the tessellation time for singularities accounts for 48% to 88% of the total tessellation time, underscoring the significance of efficient tessellation on singularities. As previously mentioned, direct evaluation techniques such as OpenSubdiv require extensive preprocessing to achieve optimal performance during runtime. This preprocessing cost can be several orders of magnitude higher than the actual tessellation time, thereby limiting the maximum subdivision depth and, consequently, the rendering accuracy. In pixel-accurate rendering scenarios, which require view-dependent error control, such as when the user zooms in, the pre-specified maximum subdivision depth may become insufficient to satisfy the tolerance requirement. This, in turn, can lead to a significant increase in computational cost near singularities.

Table 2 presents the time costs of various methods for performing tessellation using a uniform tessellation level p (i.e., sampling $(p+1)\times(p+1)$ vertices). For patches with singularities, we utilize CER-based FAS for tessellation to a depth of $m = \lceil \log_2(p) \rceil$, where each bi-cubic Bspline surface generated at depths $0 < d \leq m$ is tessellated with a level of $max(p/2^d, 1)$. We compare our method against existing breadth-first subdivision methods [MWS*20; DV21; KOCM23] and OpenSubdiv[Pixar25]. Here, tessellation time refers to the computation spent tessellating the model prior to rasterization, excluding preprocessing overhead (e.g., stencil table construction in OpenSubdiv [Pixar25]) to ensure fairness. A full comparison and analysis of rendering time are provided in Figures 7 and 8.

Experimental results in Table 2 show that our method is consistently the fastest across all test cases. Specifically, our method outperforms OpenSubdiv by 3.8×–4.7×, and outperforms [KOCM23] by 4.2×–7.1× when $p = 64$. At lower tessellation levels ($p = 16$ and $p = 32$), the improvements are smaller—2.0×–3.2× over OpenSubdiv and 1.2×–4.4× over [KOCM23]—which is expected. Our memory-efficient design exhibits its advantages most clearly under higher GPU pressure, such as at large tessellation levels or with more complex data.

Figures 7 and 8 present detailed performance data and breakdowns for uniform tessellation levels $p = 16, 32, 64$ as well as for adaptive tessellation under pixel-accurate rendering, as described in Section 3. With uniform tessellation levels of $p = 16, 32, 64$, our method achieves a 2.6×–3.7× speedup in rendering time compared to OpenSubdiv. The rendering performance results are consistent across the two devices. It is worth noting, however, that the full potential of EBOAT is realized in error-bounded tessellation scenarios—such as pixel-accurate rendering—since the two-phase error control is not enabled under uniform tessellation. In these scenarios, tessellation costs are significantly reduced compared to uniform tessellation, while high rendering quality and geometric fidelity are preserved. The breakdown in Figure 8 further shows that tessellation on singularities is no longer a bottleneck for high-quality rendering, particularly under view-dependent tolerances or pixel-accurate settings.

	Big Guy_x10	Monster Frog_x10	Orge_x10	T-Rex_x5	Car_x10	ArmorGuy_x5
Opensubdiv	2.79	2.90	4.67	10.45	3.29	11.35
SLAK	5.36	5.38	5.70	9.42	5.94	8.51
HalfEdgeSubd	2.21	2.02	2.13	4.71	2.23	6.52
Edge-Friend	1.68	1.59	1.57	6.42	1.89	5.89
Ours	1.32	1.43	1.51	3.83	1.50	3.53

(a) Tessellation level = 16

	Big Guy_x10	Monster Frog_x10	Orge_x10	T-Rex_x5	Car_x10	ArmorGuy_x5
Opensubdiv	5.31	5.52	8.23	20.74	6.26	20.03
SLAK	9.06	8.52	8.59	24.79	10.20	20.79
HalfEdgeSubd	8.92	7.83	8.10	30.14	9.84	26.25
Edge-Friend	5.46	5.76	6.02	24.64	7.11	21.14
Ours	2.01	2.12	2.79	7.13	2.30	6.29

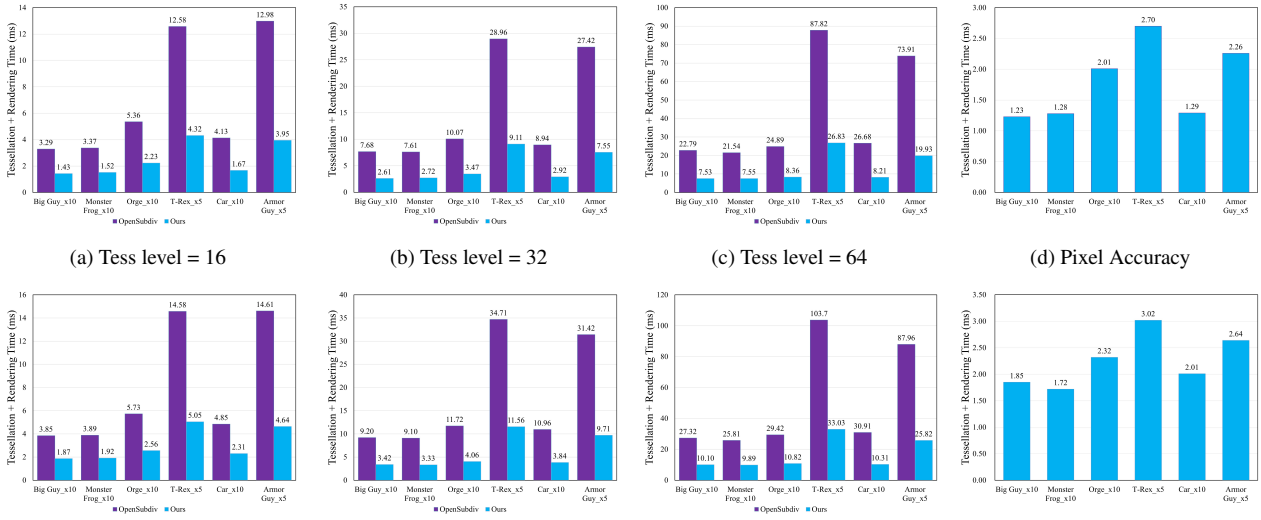
(b) Tessellation level = 32

	Big Guy_x10	Monster Frog_x10	Orge_x10	T-Rex_x5	Car_x10	ArmorGuy_x5
Opensubdiv	14.4	13.94	17.21	55.54	16.65	48.06
SLAK	22.96	20.89	21.43	87.75	28.9	68.66
HalfEdgeSubd	32.91	29.72	30.57	110.9	36.92	92.75
Edge-Friend	20.59	18.5	18.61	82.48	22.23	66.7
Ours	3.59	3.4	4.44	11.66	4.14	10.11

(c) Tessellation level = 64

Table 2: *Time (in ms) spent on tessellation with a uniform tessellation level p: (a) $p = 16$, (b) $p = 32$, and (c) $p = 64$. For our method, the reported performance includes all processing time prior to rasterization. For comparison, we also report the performance of OpenSubdiv [Pixar25], SLAK [MWS*20], HalfEdgeSubd [DV21], and Edge-Friend [KOCM23] under the exact same configuration.*

The corresponding average triangle count per base face, along with the average and maximum subdivision depths for different



(a) Tess level = 16 (b) Tess level = 32 (c) Tess level = 64 (d) Pixel Accuracy

(e) Tess level = 16 (f) Tess level = 32 (g) Tess level = 64 (h) Pixel Accuracy

Figure 7: *Average frame times (ms) for rendering various models under different tessellation modes, including uniform tessellation (levels 16, 32, and 64) and pixel-accurate adaptive tessellation, evaluated on two graphics cards: RTX 5070 Ti (first row) and RTX 4070 Super (second row). The reported frame times comprise two components: CUDA-based adaptive tessellation and Vulkan mesh-shader Blinn-Phong shading.*

	Big Guy	Monster Frog	Orge	T-Rex	Car	Armor Guy
Avg. Tri Cnt	158.62	147.06	252.29	38.84	171.43	89.13
Avg. Depth	4.32	3.53	4.42	2.67	4.21	2.84
Max. Depth	6	6	7	6	8	7

Table 3: The average triangle count per base face, along with the average and maximum subdivision depths for different models during pixel-accurate rendering. The reported data is the average over 100 views. Each model was positioned at the center of the screen, covering as many pixels as possible without triggering frustum culling. Specifically, each model was rotated around both the x- and y-axes at equal intervals, with 10 sampled angles per axis, resulting in 100 test views per model.

models during pixel-accurate rendering, are reported in Table 3. On average, the triangle count per base face remains relatively small—roughly equivalent to uniform tessellation with levels 4–10, and the average subdivision depth around singularities ranges from 2.6 to 4.3. However, depending on the viewing angle, the maximum depth can reach up to 8.

Actually for breadth-first subdivision methods, when the subdivision depth exceeds 7, the resulting refined mesh already becomes extremely large. When zooming in, arbitrarily high depths may be required to satisfy the prescribed tolerance, making pre-subdivision-based approaches such as OpenSubdiv difficult to apply in practice.

6.2. Discussions

In this subsection, we provide an in-depth discussion of our method, including validation of error control, analysis of watertightness and crack filling, as well as issues related to smooth shading.

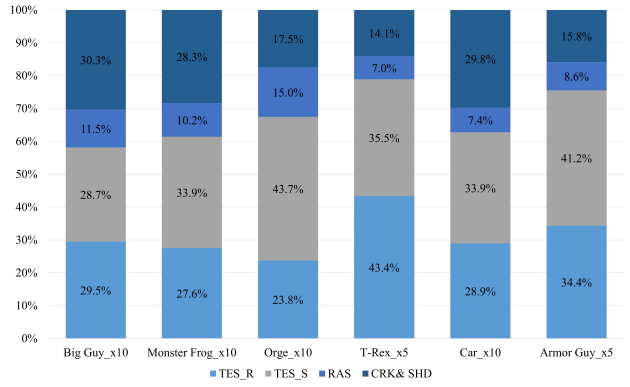
6.2.1. Error Control

We first validate whether our method achieves strict error control. Figure 9 presents the Hausdorff distance between the tessellated mesh produced by our method under a prescribed tolerance ϵ and a ground-truth reference mesh (a refined mesh at depth 7) on the Big Guy model. The Hausdorff distance is computed using MeshLab. With a tolerance of $\epsilon = 0.01$ (Figure 9a), the maximum distance reaches 0.0067, whereas tightening the tolerance to $\epsilon = 0.001$ (Figure 9b) reduces the maximum distance to 0.00078. This comparison demonstrates that our error-bounded tessellation faithfully respects the prescribed tolerance: lowering ϵ consistently results in finer tessellation and reduced approximation error.

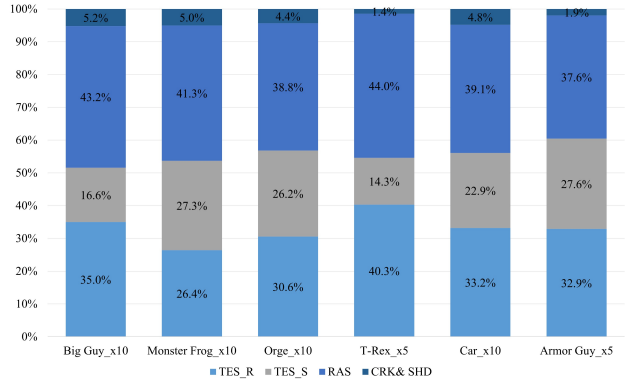
6.2.2. Memory Efficiency and Pipeline

In this work, we adopt a CUDA-to-Mesh-Shader pipeline, where CUDA is utilized for singularity processing and Mesh Shaders execute the surface tessellation. As demonstrated in [XLC*23], this workflow offers superior performance compared to fixed-function hardware tessellation, enabling greater flexibility for parallel GPU design and enhancing memory efficiency—a critical factor for memory-bound processes like tessellation—which explains the observed performance gains.

Notably, our pipeline differs from the workflow in [XLC*23] as surfaces are tessellated on-the-fly directly within the Mesh Shader, rather than being processed by a separate CUDA kernel. Consequently, the tessellation stage only requires loading the base surface



(a) Pixel-accurate rendering



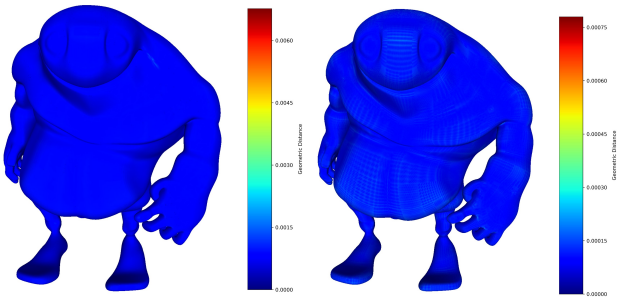
(b) Uniform rendering with level 64

Figure 8: *The breakdown of the frame time for each stage of the models, with pixel-accurate rendering (a) and uniform render-ing with level $p = 64$ (b). It shows the percentage of the total frame time allocated to each stage, including tessellation for reg-ular patches (TES_R), tessellation for singularities (TES_S), rasterization (RAS), shading (SHD) and crack-filling (CRK). We then rotated the models around both the x- and y-axes at equal intervals, capturing data at 10 angles for each axis, resulting in 100 test views per model.*

geometry, eliminating the need to pre-compute and store meshlets in global memory before passing them through the rendering pipeline. From the perspective of global memory access, this strategy avoids redundant read-and-write operations of fully tessellated primitives and removes the requirement for pre-allocating large storage buffers for tessellated geometry. Similar to the advantages over the stencil-based methods in OpenSubdiv, our memory-efficient pipeline significantly minimizes the memory footprint and traffic, which is also a primary driver behind the observed performance improvements.

6.2.3. Watertightness

We point out that, following [XLC*23], we combine pixel-accurate tessellation with image-space crack filling to guarantee crack-free rendering (i.e., no pixel dropout, as proved in Section 3 of the supplementary material of [XLC*23]), at the cost of generating non-watertight tessellated meshes.



(a) tolerance $\epsilon = 0.01$ (b) tolerance $\epsilon = 0.001$

Figure 9: *Error maps under different tolerances. The Hausdorff distance is color-mapped onto the surface, with warmer colors indicating larger deviation. (a) With a global tolerance of $\epsilon = 0.01$, the maximal Hausdorff distance is 0.006770. (b) With a global tolerance of $\epsilon = 0.001$, the maximal Hausdorff distance is reduced to 0.000780.*

While ensuring watertight tessellation is ideal for CCS surfaces, implementing this within the current Mesh-Shader pipeline requires substantially more intricate system design (e.g., regarding task granularity, meshlet sizing, and primitive generation), and most importantly, limits achievable performance. Consequently, this remains a challenging task and is left as future work.

Since our primary objective is real-time, error-bounded (pixel-accurate) rendering of CCS surfaces—rather than producing watertight intermediate meshes–this strategy provides greater flexibility in parallel GPU design and leads to improved performance, as demonstrated by the experimental results presented earlier.

6.2.4. Crack filling

Cracks in screen space arise as a consequence of generating non-watertight meshes. As illustrated in Figure 10, such cracks fall into two categories. In the first case, cracks directly expose the background, manifesting as visible pixel dropouts. In the second case, cracks reveal geometry that should have been occluded, leading to incorrectly shaded pixels.

Fortunately, due to pixel-accurate tessellation, the size of each crack is bounded by a single pixel. To address this, [XLC*23] proposed a crack-filling algorithm that detects and repairs such cracks in screen space. The basic idea is to classify a pixel p as isolated, and thus likely part of a crack (Figure 10, right) if most of its neighboring patch IDs differ from that of p. In such cases, the pixel is filled using the color of the neighboring sample with the smallest depth.

However, we point out that this method may incorrectly detect many pixels as cracks and fill them using neighboring pixels, which leads to unsmooth shading. This issue is particularly evident for CCS surfaces, where recursive subdivision can produce patches of extremely small size. In such cases, many pixels may be naturally isolated and thus mistakenly classified as cracks by [XLC*23].

This observation also motivates our adoption of a two-phase error-control strategy, which reduces subdivision depth and prevents the generation of overly fragmented patches.

We illustrate this problem in Figure 11 on the pixel-accurate rendering of the T-Rex model. When closely examining the region highlighted by the black dashed box, the black solid box on the left side shows patch IDs visualized in color, revealing a dense distribution of very small patches. In the blue box on the right side, pixels detected as cracks by the algorithm are shown in blue and red, where numerous pixels are incorrectly classified as cracks, resulting in visibly unsmooth shading.

To address this issue, we argue that the isolation criterion based solely on patch IDs is insufficient, and therefore introduce a second criterion: the occlusion criterion. For dropout pixels, we adopt the same strategy as [XLC*23]. For other cases, however, we exploit the minimum and maximum depths of the triangle corresponding to each pixel to assist in crack detection and filling.

To clearly describe the adopted occlusion criterion, we denote by t_p the triangle primitive that generates the pixel p. If there exists neighboring pixel q, whose triangle t_q lies entirely in front of t_p in depth, we consider p as increasingly likely to be occluded. We count the number of such neighbors as $occul\_cnt$. If $occul\_cnt \geq 3$, we regard pixel p as highly likely to be occluded by foreground geometry.

Hence, in our algorithm, a pixel (not dropout pixel, case 2) is finally classified as a crack only when both conditions are satisfied: (i) it is detected as isolated based on the patch-ID criterion, and (ii) it satisfies $occul\_cnt \geq 3$ under the depth test. This combined strategy greatly reduces the probability of false positives. Then, during the filling stage, we only use the color of a neighboring pixel q whose associated triangle lies entirely in front of that of p in depth.

We do not need to worry about missing cracks. For case 1 (dropout pixels), we continue to rely on the isolation criterion. For case 2, however, we combine both the isolation and occlusion criteria for detection and repair, which greatly alleviates the problem of incorrect shading in the original approach. Improved results highlighted in the red box of Figure 11 demonstrate significantly smoother shading.

It should be noted that the above results are shown as locally enlarged renderings. When directly zooming in, as illustrated in the green box of Figure 11, both methods produce smooth shading. This is because, once sufficiently zoomed in, the sampling rate adaptively increases under pixel-accurate tessellation, and the color differences between neighboring pixels become negligibly small.

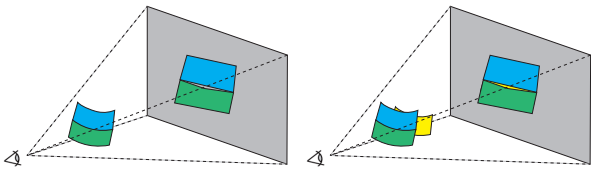


Figure 10: *Non-watertight meshes may introduce cracks in screen space.*

Actually, the issue of unsmooth shading can also be alleviated

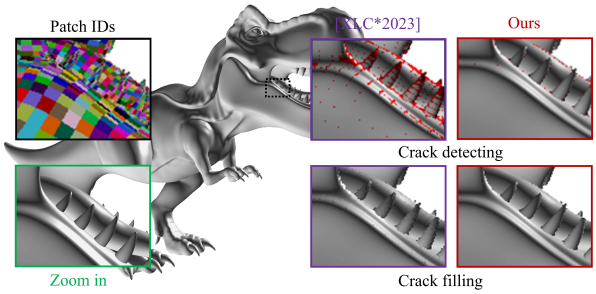


Figure 11: *Pixel-accurate rendering on the T-Rex model with two different crack-filling algorithms. We examine the region highlighted by the black dashed box. On the left, the black solid box (top-left) shows patch IDs visualized with distinct colors, while the green box illustrates the dynamic sampling rate under zoom-in. On the right, we present enlarged views of the rendered image in the same region: the first row highlights cracks detected in the rendering results (in red or blue color, representing two kinds of crack detected as in Figure 10), and the second row shows the corresponding results after crack filling.*

by adding normal control. As shown in Figure 12, the green box illustrates how controlling the size of the normal cone at boundaries increases the sampling rate and improves rendering quality. The red box shows that our crack-filling approach still produce good results.

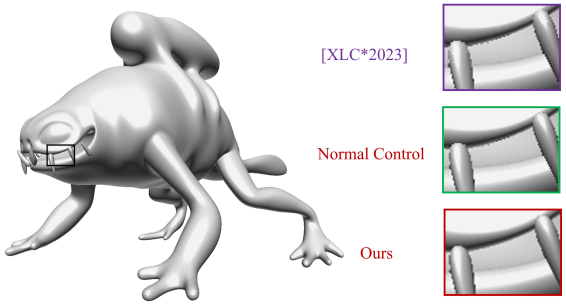


Figure 12: *Pixel-accurate rendering on the Monster Frog model. In the blue box, the crack-filling method of [XLC*23] may lead to unsmooth shading. In the green box, adding normal control alleviates this issue. In the red box, our crack-filling approach produces results.*

7. Conclusions

In this paper, we present EBOAT, a novel real-time error-bounded tessellation method for CCS surfaces, aiming to address the challenges posed by singularities. Rather than subdividing the model to a pre-specified depth to optimize runtime tessellation, causing limited rendering accuracy and geometry fidelity, our approach focuses on real-time adaptive, error-bounded tessellation for CCS surfaces, allowing for arbitrary tessellation levels or depths. Our primary contributions lie in two core components of EBOAT. First, we propose a GPU-driven workflow for real-time, error-bounded rendering of CCS surfaces via adaptive tessellation, which avoids reliance

on pre-specified subdivision depths and supports pixel-accurate rendering. Second, we introduce an efficient, on-the-fly CER-based FAS algorithm, enhanced by a two-phase error control strategy and a tailored depth estimation scheme. This design enables substantial performance gains through memory-efficient processing and GPU-friendly access patterns. Experimental results demonstrate that our FAS algorithm significantly outperforms existing methods in performance. Furthermore, our depth estimation achieves minimal subdivision depth with negligible overhead, enabling efficient error-bounded tessellation near singularities through a two-phase error control strategy. As a result, substantial improvements are observed in both tessellation and overall rendering performance.

Under error-bounded tessellation scenarios, where pixel accuracy is used as the error tolerance, the maximum subdivision depth can reach up to 8 across different views—even in our tests without deliberate zooming, but simply through rotation. This is unfavorable for breadth-first subdivision methods lacking adaptivity, as they produce excessively large refined meshes with many regions heavily over-tessellated. For prior direct-evaluation methods, whose performance relies on pre-subdivision at a specified depth, strict error control may not be satisfied under certain views; otherwise, costly on-the-fly subdivision must be performed, which makes real-time applications infeasible.

Furthermore, pixel-accurate sampling ensures geometric fidelity to the limit surface in screen space, maintaining high rendering quality. EBOAT is especially well-suited for integration into rendering scenarios such as model-in-edit rendering in the viewport of modeling software, and high-poly model rendering in video games.

Limitations & Future work. Although EBOAT achieves substantial improvements on performance in adaptive tessellation of CCS surfaces, several limitations remain that we tend to address in future work.

Our current implementation does not support hierarchical edits [FB88] and Chaikin rules [Cha74; DKT98]. However, given that our current implementation already accommodates adaptive subdivision and semi-sharp or hard creases, these extensions can be implemented with minimal effort.

Driven by the goal of error-bounded real-time rendering, EBOAT ensures visually crack-free results by combining pixel-accurate tessellation with an image-space crack-filling approach similar to [XLC*23]. We also conducted an analysis of the quality of the generated renderings. However, as discussed in Section 6.2.4, the tessellated meshes are not as watertight as those produced by [NLMD12]. Achieving high-performance, watertight, pixel-accurate CCS rendering within a mesh shader pipeline remains a challenging task, which we leave for future work.

Acknowledgements

We gratefully acknowledge Jonathan Dupuy for providing the Catmull-Clark Subdivision surface models. We also thank the anonymous reviewers for their constructive comments and valuable suggestions. This work is supported by the National Key R&D Program of China (2022YFB3303400) and the National Natural Science Foundation of China (62025207).

References

[BFK*16] BRAINERD, WADE, FOLEY, TIM, KRAEMER, MANUEL, et al. "Efficient GPU rendering of subdivision surfaces using adaptive quadtrees". *ACM Transactions on Graphics (TOG)* 35.4 (2016), 1–12 2, 3.

[BZ04] BOIER-MARTIN, IOANA and ZORIN, DENIS. "Differentiable parameterization of Catmull-Clark subdivision surfaces". *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. 2004, 155–164 7.

[CC78] CATMULL, EDWIN and CLARK, JAMES. "Recursively generated B-spline surfaces on arbitrary topological meshes". *Computer-Aided Design* 10.6 (1978), 350–355 2.

[CCY06] CHENG, FUHUA, CHEN, GANG, and YONG, JUN-HAI. "Subdivision depth computation for extra-ordinary Catmull-Clark subdivision surface patches". *Computer Graphics International Conference*. Springer. 2006, 404–416 3.

[Cha74] CHAIKIN, GEORGE MERRILL. "An algorithm for high-speed curve generation". *Computer graphics and image processing* 3.4 (1974), 346–349 12.

[CY06] CHENG, FUHUA and YONG, JUN-HAI. "Subdivision depth computation for Catmull-Clark subdivision surfaces". *Computer-Aided Design and Applications* 3.1-4 (2006), 485–494 3.

[DKT98] DEROSE, TONY, KASS, MICHAEL, and TRUONG, TIEN. "Subdivision surfaces in character animation". SIGGRAPH '98. New York, NY, USA: Association for Computing Machinery, 1998, 85–94. ISBN: 0897919998. DOI: 10.1145/280814.280826. URL: https://doi.org/10.1145/280814.280826 4, 12.

[DS78] DOO, D and SABIN, M. "Behaviour of recursive division surfaces near extraordinary points". *Computer-Aided Design* 10.6 (1978), 356–360 2.

[DV21] DUPUY, JONATHAN and VANHOEY, KENNETH. "A Halfedge Refinement Rule for Parallel Catmull-Clark Subdivision". *Computer Graphics Forum*. Vol. 40. 8. Wiley Online Library. 2021, 57–70 2, 3, 5, 8.

[FB88] FORSEY, DAVID R and BARTELS, RICHARD H. "Hierarchical B-spline refinement". *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*. 1988, 205–212 12.

[FMM86] FILIP, DANIEL, MAGEDSON, ROBERT, and MARKOT, ROBERT. "Surface algorithms using bounds on derivatives". *Computer Aided Geometric Design* 3.4 (1986), 295–311 7.

[HDD*94] HOPPE, HUGUES, DEROSE, TONY, DUCHAMP, TOM, et al. "Piecewise smooth surface reconstruction". *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. 1994, 295–302 2, 4.

[HDW08] HUANG, ZHANGJIN, DENG, JIANSONG, and WANG, GUOPING. "A bound on the approximation of a Catmull–Clark subdivision surface by its limit mesh". *Computer Aided Geometric Design* 25.7 (2008), 457–469 2, 3, 6, 8.

[HW07a] HUANG, ZHANGJIN and WANG, GUOPING. "Distance between a Catmull-Clark subdivision surface and its limit mesh". *Proceedings of the 2007 ACM symposium on Solid and physical modeling*. 2007, 233–240 3.

[HW07b] HUANG, ZHANGJIN and WANG, GUOPING. "Improved error estimate for extraordinary Catmull–Clark subdivision surface patches". *The Visual Computer* 23 (2007), 1005–1014 2, 3, 8.

[KOCM23] KUTH, BASTIAN, OBERBERGER, MAX, CHAJDAS, MATTHÄUS, and MEYER, QUIRIN. "Edge-Friend: Fast and Deterministic Catmull-Clark Subdivision Surfaces". *Computer Graphics Forum*. Vol. 42. 8. Wiley Online Library. 2023, e14863 2, 3, 8.

[LS08] LOOP, CHARLES and SCHAEFER, SCOTT. "Approximating Catmull-Clark subdivision surfaces with bicubic patches". *ACM Transactions on Graphics (TOG)* 27.1 (2008), 1–11 3.

[LSNC09] LOOP, CHARLES, SCHAEFER, SCOTT, NI, TIANYUN, and CASTANO, IGNACIO. "Approximating subdivision surfaces with gregory patches for hardware tessellation". *ACM SIGGRAPH Asia 2009 papers*. 2009, 1–9 2, 3, 6, 7.

[MNP08] MYLES, ASHISH, NI, TIANYUN, and PETERS, JÖRG. "Fast parallel construction of smooth surfaces from meshes with tri/quad/pent facets". *Computer Graphics Forum*. Vol. 27. 5. Wiley Online Library. 2008, 1365–1372 3.

[MWS*20] MLAKAR, DANIEL, WINTER, MARTIN, STADLBAUER, PASCAL, et al. "Subdivision-specialized linear algebra kernels for static and dynamic mesh connectivity on the gpu". *Computer Graphics Forum*. Vol. 39. 2. Wiley Online Library. 2020, 335–349 2, 3, 5, 6, 8.

[MYP08] MYLES, ASHISH, YEO, YOUNG IN, and PETERS, JÖRG. "Gpu conversion of quad meshes to smooth surfaces". *Proceedings of the 2008 ACM symposium on Solid and physical modeling*. 2008, 321–326 3.

[Nas87] NASRI, AHMAD H. "Polyhedral subdivision methods for free-form surfaces". *ACM Transactions on Graphics (TOG)* 6.1 (1987), 29–73 2, 4.

[NKF*16] NIESSNER, MATTHIAS, KEINERT, BENJAMIN, FISHER, MATTHEW, et al. "Real-time rendering techniques with hardware tessellation". *Computer Graphics Forum*. Vol. 35. 1. 2016, 113–137 3.

[NLG12] NIESSNER, MATTHIAS, LOOP, CHARLES T, and GREINER, GÜNTHER. "Efficient Evaluation of Semi-Smooth Creases in Catmull-Clark Subdivision Surfaces." *Eurographics (Short Papers)* 41 (2012), 44 3, 4.

[NLMD12] NIESSNER, MATTHIAS, LOOP, CHARLES, MEYER, MARK, and DEROSE, TONY. "Feature-adaptive GPU rendering of Catmull-Clark subdivision surfaces". *ACM Transactions on Graphics (TOG)* 31.1 (2012), 1–11 2, 3, 5, 6, 12.

[NYM*08] NI, TIANYUN, YEO, Y, MYLES, ASHISH, et al. "GPU smoothing of quad meshes". *2008 IEEE International Conference on Shape Modeling and Applications*. IEEE. 2008, 3–9 3.

[PEO09] PATNEY, ANJUL, EBEIDA, MOHAMED S, and OWENS, JOHN D. "Parallel view-dependent tessellation of Catmull-Clark subdivision surfaces". *Proceedings of the conference on high performance graphics 2009*. 2009, 99–108 3.

[Pixar25] *OpenSubdiv 3.7 User Documentation*. 2025. URL: https://graphics.pixar.com/opensubdiv/docs/intro.html (visited on 01/11/2025) 2, 3, 8.

[PR04] PETERS, JÖRG and REIF, ULRICH. "Shape characterization of subdivision surfacesbasic principles". *Computer Aided Geometric Design* 21.6 (2004), 585–599 7.

[PR08] PETERS, JÖRG and REIF, ULRICH. *Subdivision surfaces*. Springer, 2008 7.

[PW09] PETERS, JÖRG and WU, XIAOBIN. "The distance of a subdivision surface to its control polyhedron". *Journal of Approximation Theory* 161.2 (2009), 491–507 2, 3, 6–8.

[Rei95] REIF, ULRICH. "A unified approach to subdivision algorithms near extraordinary vertices". *Computer Aided Geometric Design* 12.2 (1995), 153–174 7.

[Rhino25] *Rhino 8 - SubD (Subdivision Surface Modeling)*. 2025. URL: https://www.rhino3d.com/en/features/subd/ (visited on 01/11/2025) 2.

[SJP05] SHIUE, LE-JENG, JONES, IAN, and PETERS, JÖRG. "A realtime GPU subdivision kernel". *ACM Transactions on Graphics (TOG)* 24.3 (2005), 1010–1015 3, 5.

[SRK*15] SCHÄFER, HENRY, RAAB, JENS, KEINERT, BENJAMIN, et al. "Dynamic feature-adaptive subdivision". *Proceedings of the 19th symposium on interactive 3d graphics and games*. 2015, 31–38 2, 3.

[Sta98] STAM, JOS. "Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values". *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '98. New York, NY, USA: Association for Computing Machinery, 1998, 395–404. ISBN: 0897919998. DOI: 10.1145/280814.280945. URL: https://doi.org/10.1145/280814.280945 3, 6.

[Tessellation23] MICROSOFT CORPORATION. *Tessellation Stages*. Accessed: 2026-01-23. Oct. 2023. URL: https://learn.microsoft.com/en-us/windows/win32/direct3d11/direct3d-11-advanced-stages-tessellation 3.

[Turing18] KUBISCH, CHRISTOPH. *Introduction to Turing Mesh Shaders*. Accessed: 2026-01-23. Sept. 2018. URL: https://developer.nvidia.com/blog/introduction-turing-mesh-shaders/ 3.

[VD22] VANHOEY, KENNETH and DUPUY, JONATHAN. "A Halfedge Refinement Rule for Parallel Loop Subdivision." *Eurographics (Short Papers)*. 2022, 41–44 3.

[VPBM01] VLACHOS, ALEX, PETERS, JÖRG, BOYD, CHAS, and MITCHELL, JASON L. "Curved PN triangles". *Proceedings of the 2001 symposium on Interactive 3D graphics*. 2001, 159–166 3.

[WC23] WANG, KECHUN and CHEN, RENJIE. "Parallel Loop Subdivision with Sparse Adjacency Matrix". (2023) 3.

[XLC*23] XIONG, RUICHENG, LU, YANG, CHEN, CONG, et al. "ETER: Elastic Tessellation for Real-Time Pixel-Accurate Rendering of Large-Scale NURBS Models". *ACM Transactions on Graphics (TOG)* 42.4 (2023), 1–13 2–4, 7, 9–12.

[YBP12] YEO, YOUNG IN, BIN, LIHAN, and PETERS, JÖRG. "Efficient pixel-accurate rendering of curved surfaces". *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. 2012, 165–174 4, 7.

[ZLCL25] ZENG, YAJUN, LU, YANG, CHEN, CONG, and LIU, LIGANG. "WATER: Watertight Tessellation for Real-Time Pixel-Accurate Rendering of Large-Scale Surfaces". 44.6 (Dec. 2025). ISSN: 0730-0301. DOI: 10.1145/3763317. URL: https://doi.org/10.1145/3763317 3.

[ZS00] ZHENG, JIANMIN and SEDERBERG, THOMAS W. "Estimating tessellation parameter intervals for rational curves and surfaces". *ACM Transactions on Graphics (TOG)* 19.1 (2000), 56–77 7.