A Halfedge Refinement Rule for Parallel Catmull-Clark Subdivision
Supplemental Material: GPU Performance Measurements

J. Dupuy and K. Vanhoey

Unity Technologies

This document provides exhaustive performance measurements of open source GPU-based Catmull-Clark subdivision implementations.

Methods. The open source implementations we identified and consider are listed in Table together with the type of mesh features they do or do not handle. OpenSubdiv is the industry standard. We run it using its GLSL backend on Linux. Nießner et al.’s application is Windows-only (it uses DirectX). Patney et al. provides an end-to-end rendering implementation in CUDA, that does not allow to isolate Vertex Point Subdivision. Thus we only execute it for the End-to-End Subdivision scenario. For fair comparison, we modified their code to output a uniform instead of an adaptive subdivision. Mlakar et al. provide an end-to-end parallel GPU implementation using CUDA. To measure timings for the Vertex Point Subdivision scenario, we modified their code to pre-compute topology subdivision and store its results in a table. We then run and time Vertex Point Subdivision only. We report runtimes as measured by their provided CUDA timers, which we modified in one respect: Their count excludes time spent on CUDA memory allocations and free's (because they can in theory be pre-allocated) but also memset operations (which cannot be preallocated) which we included. Our method provides both scenario’s using GLSL shaders as illustrated in the accompanying code.

<table>
<thead>
<tr>
<th>Method</th>
<th>O.S.</th>
<th>Non-quad</th>
<th>Creases</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenSubdiv [Pix13]</td>
<td>Linux</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Nießner et al. [NLMD12]</td>
<td>Windows</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Patney et al. [PEO09]</td>
<td>Windows</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Mlakar et al. [MWS∗20]</td>
<td>Linux</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Ours</td>
<td>Linux</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

Data. We consider a total of 8 meshes with different properties (see sections 1 to 8). The first four meshes have semi-sharp creases, which are only supported by OpenSubdiv and our implementation. The last four consist of two quad-only and boundary-free meshes and two more complex meshes with boundaries and non quads.

Protocol. We performed GPU runtime measurements on each mesh and for six subdivision depths (depths 1 to 6) by all methods capable of handling them. Note that subdividing the T-Rex model (Section 8) down to depth 6 using OpenSubdiv resulted in a GPU out-of-memory, hence its absence in the plots. For each subdivision, we show three plots, akin to those in our main paper: two show the timings for the End-to-End Subdivision and Vertex Point Subdivision scenarios, respectively, and one shows the timings of each of our individual GLSL shaders. Each plot reports the median runtime measured over 50 evaluations and the minimum and maximum runtime as error bars. As explained in the main paper: we made sure all timings include shader/kernel execution time, necessary memset instructions, state changes, and CPU-GPU synchronizations. All measurements were done on an NVIDIA RTX 2080 graphics card and a 4.00GHz Intel Core i7-8086K CPU with 32GiB RAM.

Discussion. Results are all in support of our analysis of Section 6 in the main paper. We additionally note that we observe large variations among runs (see error bars) for the method of Nießner et al.

References

[Pix13] Pixar: Opensubdiv from research to industry adoption. In ACM SIGGRAPH 2013 Courses (New York, NY, USA, 2013), SIGGRAPH ’13, Association for Computing Machinery. 1

submitted to High-Performance Graphics (2021)
1. Rook

1. Rook

End-to-End Subdivision

Vertic Point Subdivision

Our Kernel Timings

submitted to High-Performance Graphics (2021)
2. Bishop

Bishop

(332 non-quads / 24 boundaries)
224 creases

$H_0 = 3,740$
$E_0 = 968$
$V_0 = 917$

$H_4 = 957,440$
$E_4 = 239,360$
$V_4 = 239,555$

submitted to High-Performance Graphics (2021)
3. Car

![Car Image]

<table>
<thead>
<tr>
<th>Depth</th>
<th>$H_0$</th>
<th>$F_0$</th>
<th>$E_0$</th>
<th>$V_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6,300</td>
<td>1,612,800</td>
<td>3,180</td>
<td>314 creases</td>
</tr>
<tr>
<td>4</td>
<td>1,642</td>
<td>403,717</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

submitted to High-Performance Graphics (2021)
4. ArmorGuy

ArmorGuy

.getEnd-to-End Subdivision

End-to-End Subdivision

Vertex Point Subdivision

Our Kernel Timings

submitted to High-Performance Graphics (2021)
5. Bigguy

![Bigguy](image)

\[ S^3 \rightarrow S^1 \]

(all-quads / no boundaries)

- \( H_0 = 5,800 \quad R_0 = 1,484,800 \)
- \( F_0 = 1,450 \quad F_0 = 372,200 \)
- \( E_0 = 2,900 \quad E_0 = 742,400 \)
- \( V_0 = 1,452 \quad V_0 = 371,202 \)

---

End-to-End Subdivision

![Graph](image)

- OpenSubdiv
- [NLMD12]
- [PEO09]
- [MWS20]
- Ours

Vertex Point Subdivision

![Graph](image)

- OpenSubdiv
- [NLMD12]
- [MWS20]
- Ours

Our Kernel Timings

![Graph](image)

- ClearBuffer
- FacePoints
- EdgePoints
- VertexPoints
- Halfedges

submitted to High-Performance Graphics (2021)
6. Monsterfrog

![Monsterfrog Image]

(all-quads / no boundaries)

<table>
<thead>
<tr>
<th>Depth</th>
<th>Faces</th>
<th>Edges</th>
<th>Vertices</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1,368</td>
<td>2,922</td>
<td>308</td>
</tr>
<tr>
<td>4</td>
<td>330,768</td>
<td>661,504</td>
<td>330,768</td>
</tr>
</tbody>
</table>

submitted to High-Performance Graphics (2021)
7. Imrod

Imrod

(3,479 non-quads / 223 boundaries)

\(H_0 = 21.399\)  \(V_0 = 4.630\)  \(E_0 = 10.811\)  \(F_0 = 1.371, 341\)

submitted to *High-Performance Graphics* (2021)
8. T-Rex

(T-Rex)

(S0 / S4)

{(468 non-quads / 594 boundaries)}

- $H_0 = 45.224$
- $H_3 = 11.577, 344$
- $E_0 = 11, 422$
- $E_3 = 2, 894, 336$
- $V_0 = 11, 539$
- $V_3 = 2, 899, 140$

---

**End-to-End Subdivision**

- OpenSubdiv
- [NLMD12]
- [MWS∗20]
- Ours

**Vertex Point Subdivision**

- OpenSubdiv
- [NLMD12]
- [MWS∗20]
- Ours

**Our Kernel Timings**

- ClearBuffer
- FacePoints
- EdgePoints
- VertexPoints
- Halfedges