A Halfedge Refinement Rule for Parallel Loop Subdivision

Supplemental Material: GPU Performance Measurements

Kenneth Vanhoey
Unity Technologies

Jonathan Dupuy
Unity Technologies

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

Methods. While there are several articles on the topic of GPU-based Loop subdivision [HFN+14, MWS+20], none provide open-source code. **OpenSubdiv** is an open-source software. It is the current industry standard for GPU-based subdivision. We run it using its GLSL backend on Linux and time two scenarios. For the modelling scenario, OpenSubdiv relies on tables pre-computed on the CPU [HFN+14]. We thus monitor CPU time for this scenario. For the rendering scenario, we time the vertex point update step using GLSL timers using a `glQueryCounter(...,GL_TIMESTAMP)` routine. **Our** implementation handles borders and semi-sharp creases similarly to OpenSubdiv. In contrast to OpenSubdiv, we compute both scenarios on the GPU using GLSL shaders as illustrated in our accompanying code. We also time it on Linux using a `glQueryCounter(...,GL_TIMESTAMP)` routine.

Data. We consider a total of 8 meshes with different properties (see sections 1 to 8). The first three meshes have no boundaries nor semi-sharp creases. The following two have borders. The final three have borders and semi-sharp creases. Input mesh face counts vary from 634 faces (Knight, Sec. 3) to 33, 912 faces (ArmorGuyT, Sec. 8). We use a 'T' suffix for the name of the input mesh whenever we need to pre-triangulate it. We provide all input OBJ files with our source code.

Protocol. We performed GPU runtime measurements on both OpenSubdiv and our method for each mesh down to seven subdivision levels. For each subdivision, we provide three plots. The first plot provides measurements for end-to-end subdivision, which is relevant for scenarios where the topology of the input mesh is dynamic, e.g., interactive modelling. The second plot provides measurements for vertex point subdivision, which is relevant for scenarios where the topology of the input mesh is static, e.g., interactive rendering of skinned meshes. Finally, the third plot provides measurements for each GPU kernel we implemented. Each plot reports the median runtime measured over 50 evaluations and the minimum and maximum runtime as error bars. All timings include shader/kernel execution time, necessary memset instructions, state changes, and CPU-GPU synchronizations. All measurements were done on an NVIDIA Titan Xp graphics card and a 3.50GHz Intel Core i5-4690K CPU with 24GiB RAM.

Discussion. Our measurements show that our method is roughly as fast as OpenSubdiv in the *Vertex Points Subdivision* scenario, although exhibits less variance. For the *End-to-End Subdivision* scenario however, our method significantly outperforms OpenSubdiv. This is because OpenSubdiv relies on sequential CPU pre-computations in this case, while we perform all computations on the GPU.

OpenSubdiv performance measurements are left missing whenever we get an out-of-memory error on our platform. This shows that it runs out of memory for subdivision depths that our method handles well, hence has a less favorable memory complexity than we do.

Finally, we explained above that we only compare to available open-source implementations. This excludes the method of Mlakar *et al.* [MWS+20] as there is no open source implementation available for Loop subdivision. However, their method has been compared against parallel half-edge refinement for Catmull-Clark subdivision [DV21]. This showed that both methods' timings are of the same order of magnitude, with their method slightly outperforming that of Dupuy and Vanhoey. We believe that we can reasonably expect this to be similar for Loop subdivision too, since the complexities involved are similar.

References

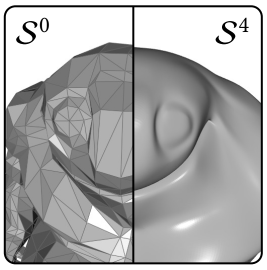
[DV21] Jonathan Dupuy and Kenneth Vanhoey. A Halfedge Refinement Rule for Parallel Catmull-Clark Subdivision. *Computer Graph. Forum*, 40(8), 2021.

[HFN+14] Yun-Cen Huang, Jieqing Feng, Matthias Nießner, Yuanmin Cui, and Baoguang Yang. Feature-adaptive rendering of loop subdivision surfaces on modern gpus. *J. Comput. Sci. Technol.*, 29(6):1014–1025, 2014.

[MWS+20] D. Mlakar, M. Winter, P. Stadlbauer, H.-P. Seidel, M. Steinberger, and R. Zayer. Subdivision-specialized linear algebra kernels for static and dynamic mesh connectivity on the gpu. *Computer Graph. Forum*, 39(2), 2020.

1 Bigguy

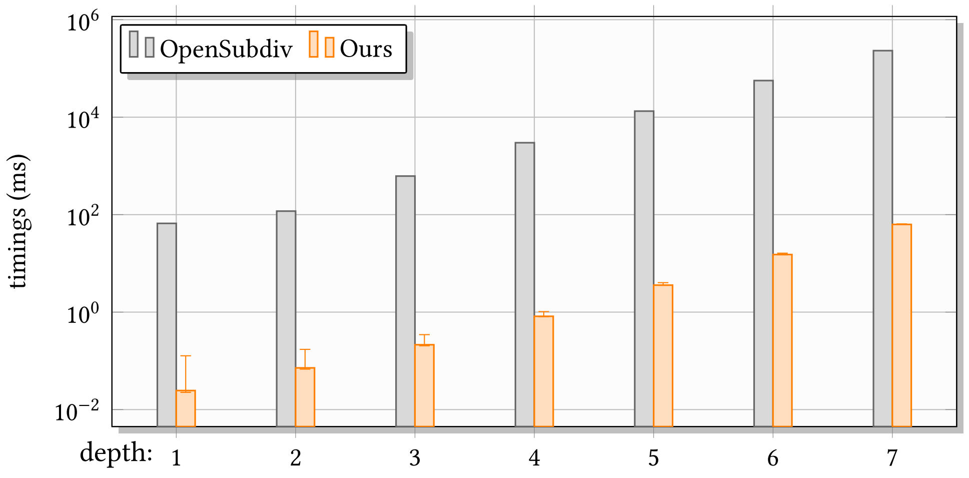
BigguyT



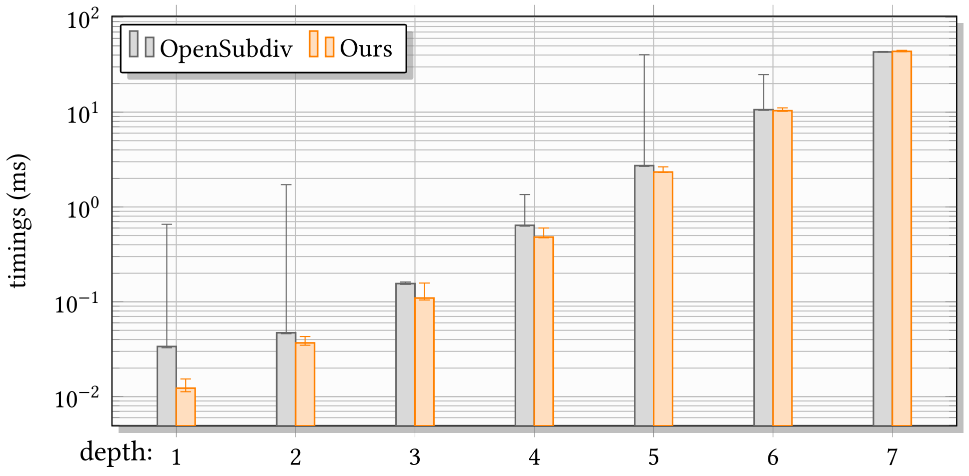
$\begin{pmatrix}\text{0 boundaries}\\ \text{0 creases}\end{pmatrix}$

$H_0 = 8,700$	$H_4 = 2,227,200$
$F_0 = 2,900$	$F_4 = 742,400$
$E_0 = 4,340$	$E_4 = 1,113,600$
$V_0 = 1,452$	$V_4 = 371,202$

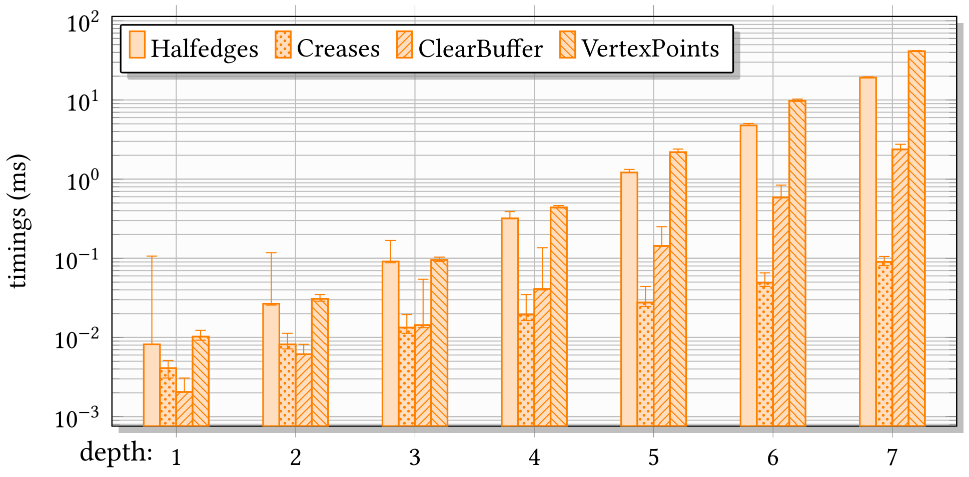
End-to-End Subdivision



Vertex Point Subdivision

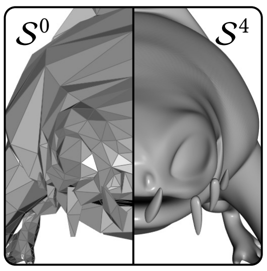


Our Kernel Timings



2 Monsterfrog

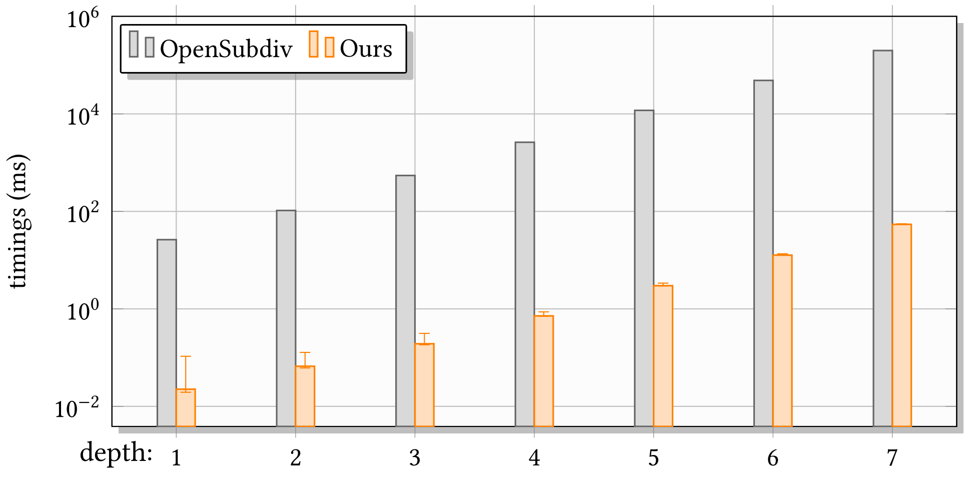
MonsterfrogT



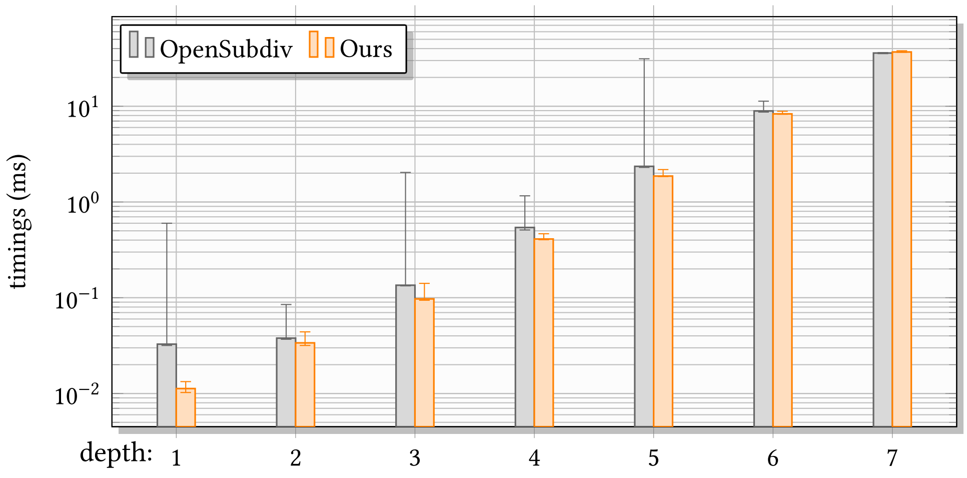
$\begin{pmatrix}\text{0 boundaries}\\ \text{0 creases}\end{pmatrix}$

$H_0 = 7,752$	$H_4 = 1,984,512$
$F_0 = 2,584$	$F_4 = 661,504$
$E_0 = 3,876$	$E_4 = 992,256$
$V_0 = 1,308$	$V_4 = 330,768$

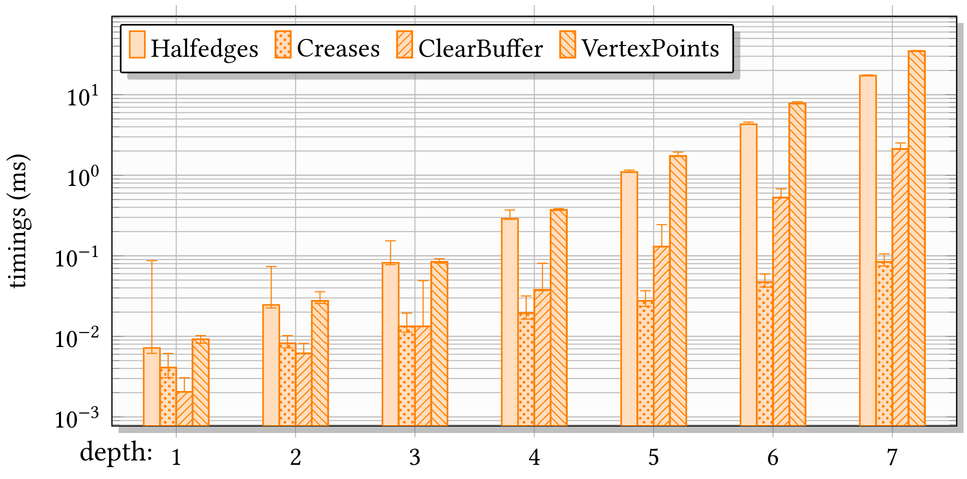
End-to-End Subdivision



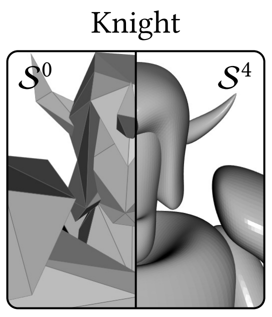
Vertex Point Subdivision



Our Kernel Timings

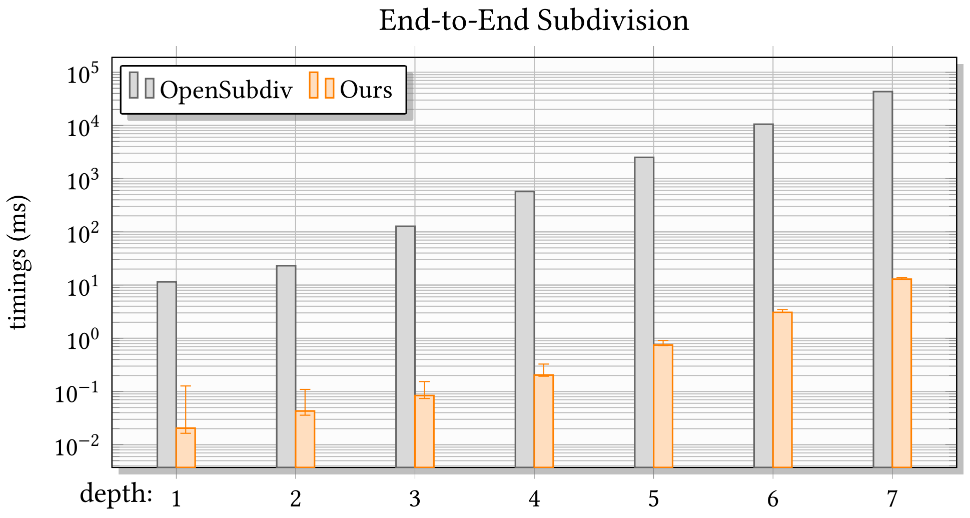


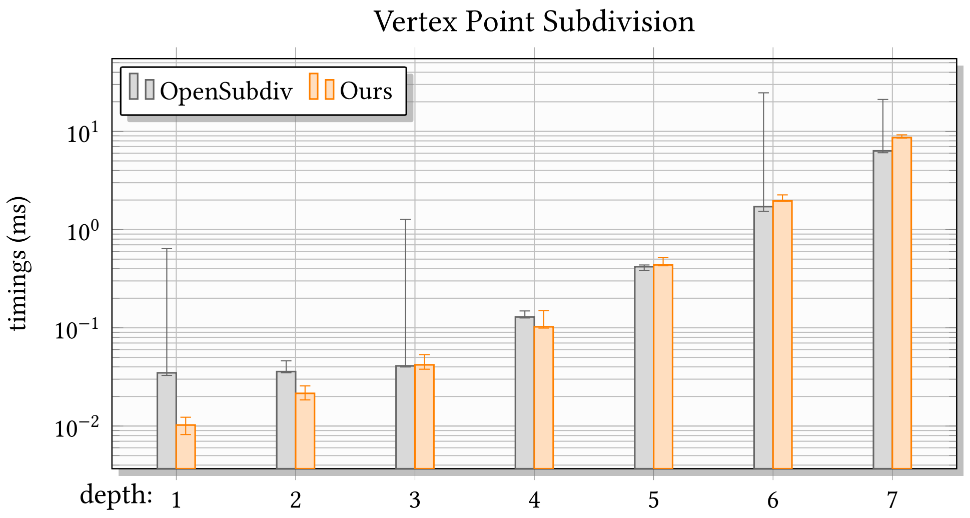
3 Knight

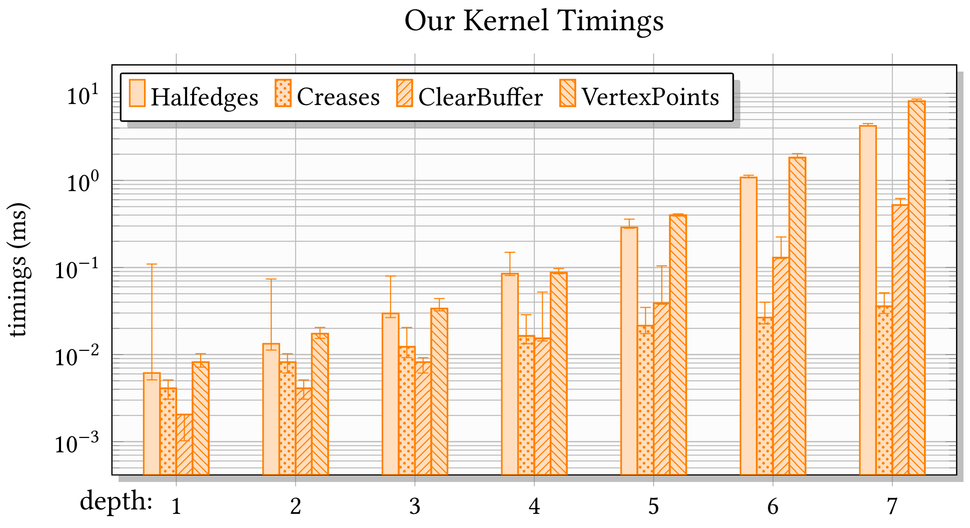


$\begin{pmatrix}\text{0 boundaries}\\ \text{0 creases}\end{pmatrix}$

$H_0 = 1,902$	$H_4 = 486,912$
$F_0 = 634$	$F_4 = 162,304$
$E_0 = 951$	$E_4 = 243,456$
$V_0 = 365$	$V_4 = 81,200$

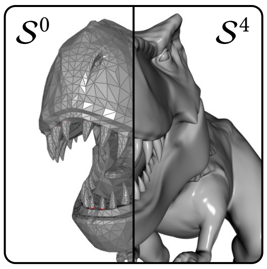






4 T-Rex

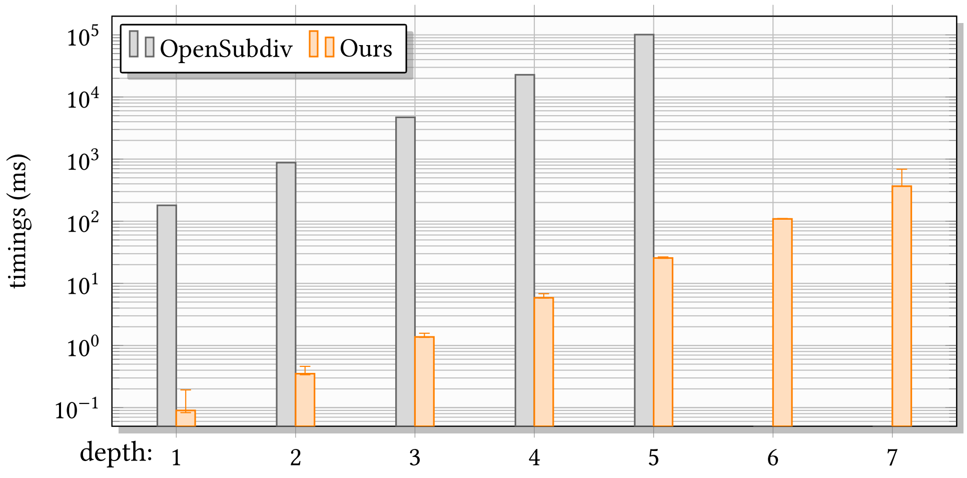
T-rexT



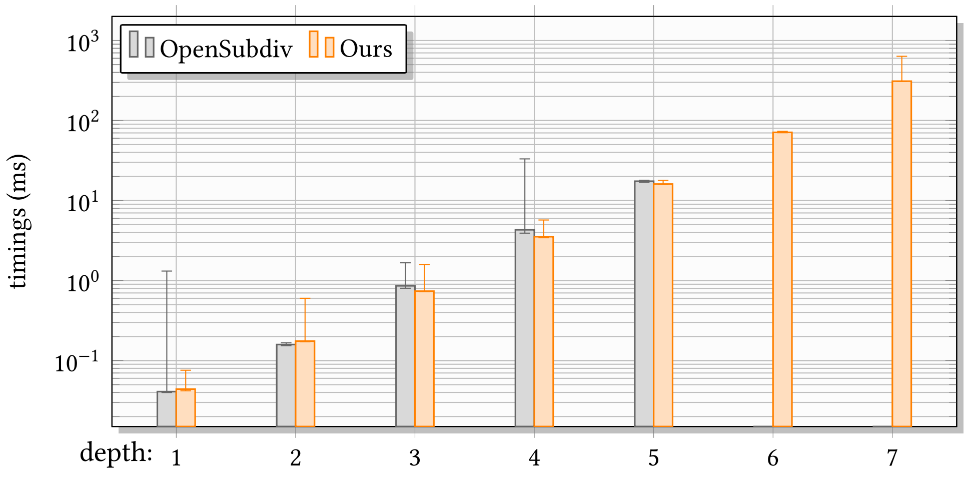
$\begin{pmatrix}\text{594 boundaries}\\ \text{0 creases}\end{pmatrix}$

$H_0 = 67,140$	$H_4 = 17,187,840$
$F_0 = 22,380$	$F_4 = 5,729,280$
$E_0 = 33,867$	$E_4 = 8,598,672$
$V_0 = 11,539$	$V_4 = 2,869,444$

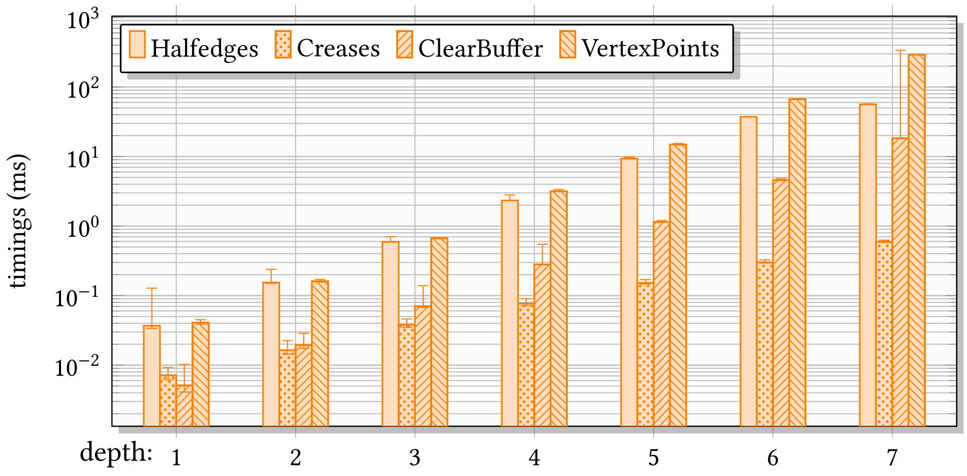
End-to-End Subdivision



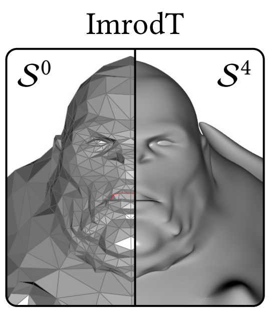
Vertex Point Subdivision



Our Kernel Timings

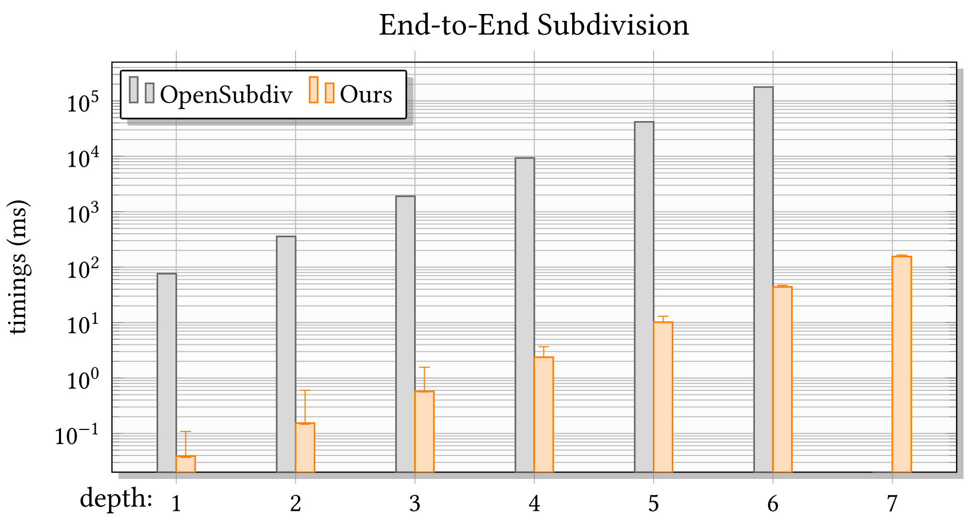


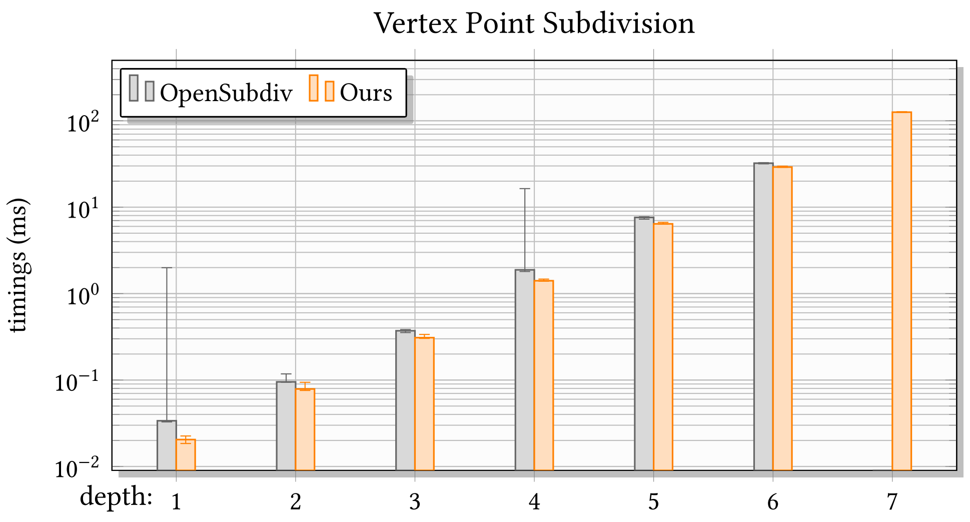
5 imrod

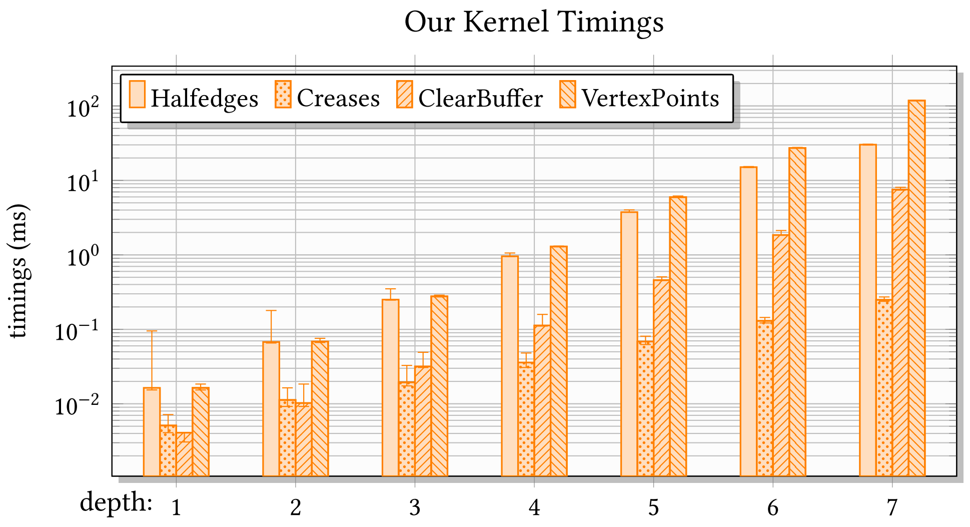


$\begin{pmatrix}\text{223 boundaries}\\ \text{0 creases}\end{pmatrix}$

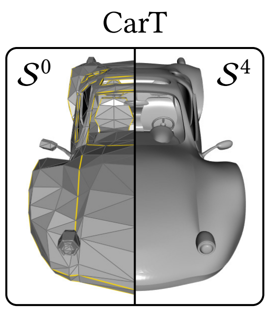
$H_0 = 26,985$	$H_4 = 6,908,160$
$F_0 = 8,995$	$F_4 = 2,302,720$
$E_0 = 13,604$	$E_4 = 3,455,864$
$V_0 = 4,630$	$V_4 = 1,153,165$





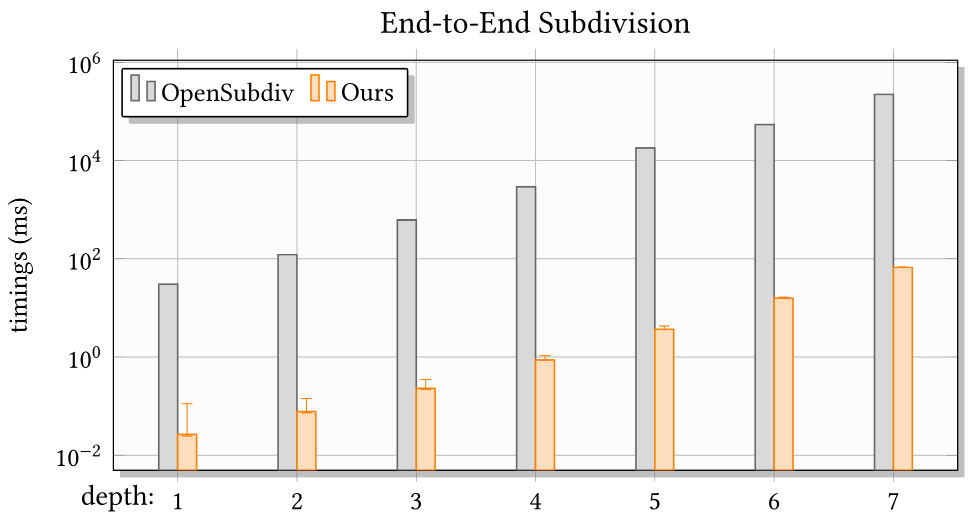


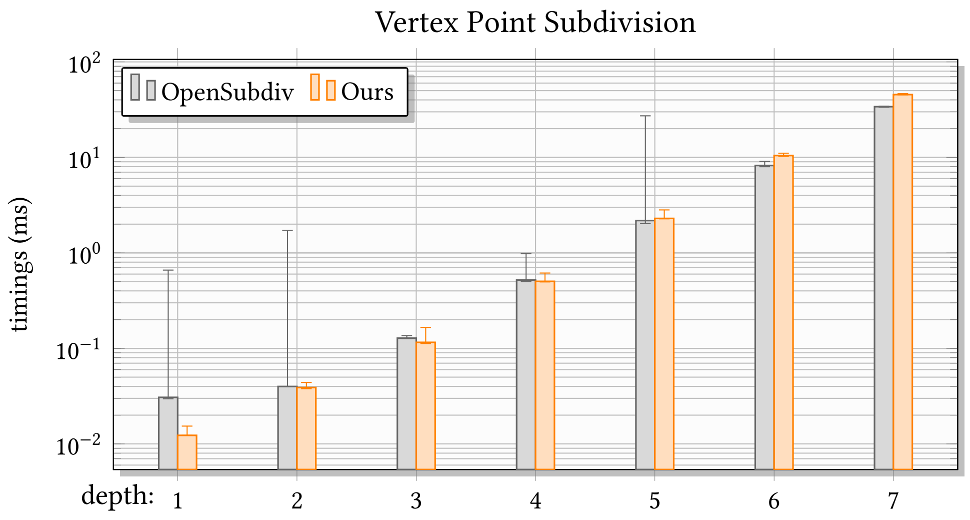
6 Car

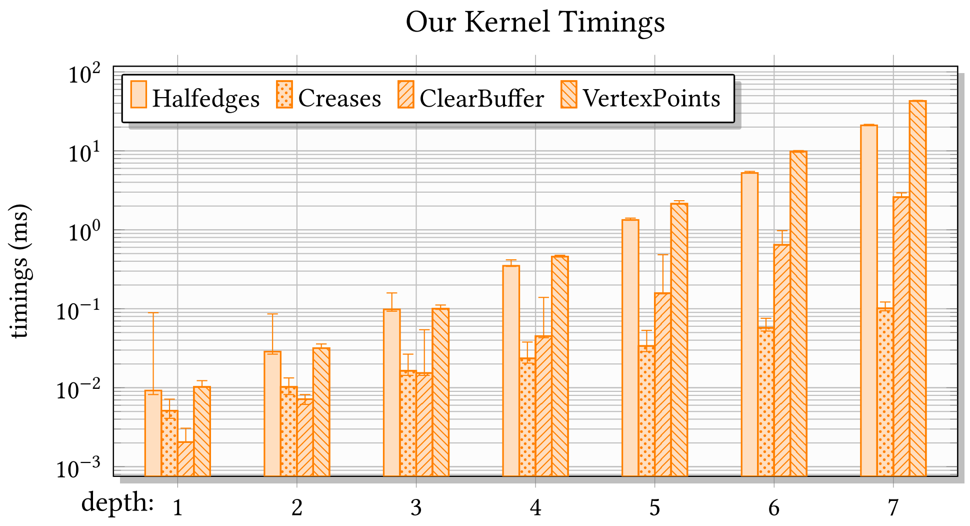


$\begin{pmatrix}\text{60 boundaries}\\ \text{314 creases}\end{pmatrix}$

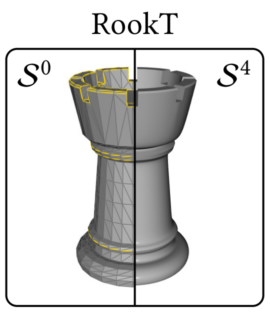
$H_0 = 9,450$	$H_4 = 2,419,200$
$F_0 = 3,150$	$F_4 = 806,400$
$E_0 = 4,755$	$E_4 = 1,210,080$
$V_0 = 1,642$	$V_4 = 403,717$





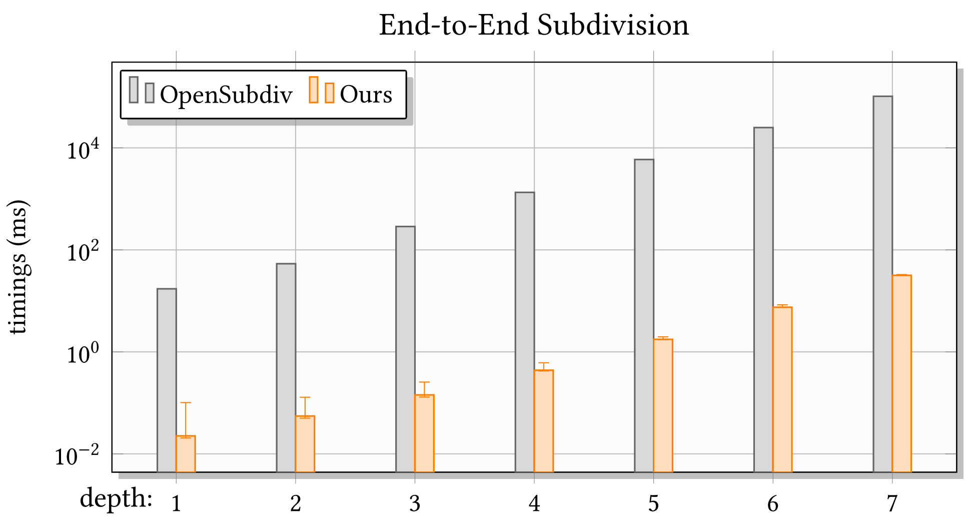


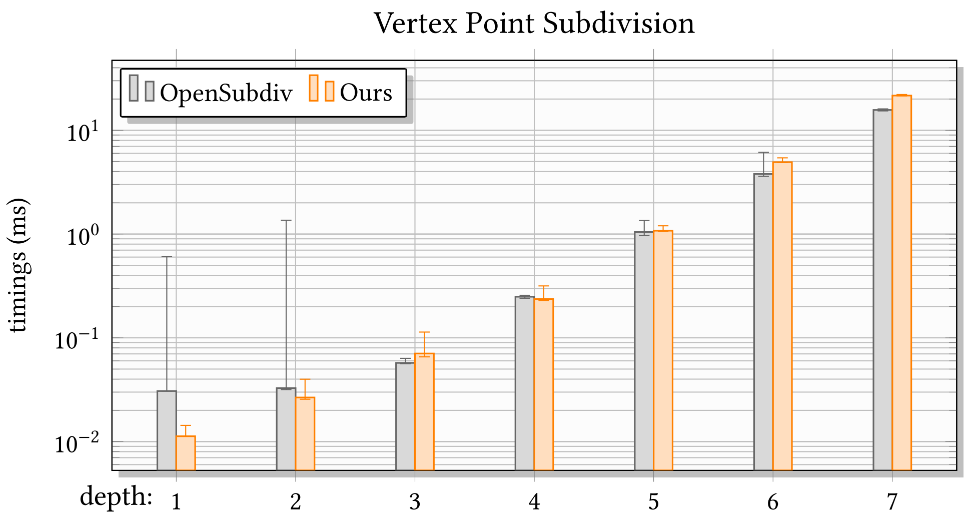
7 Rook

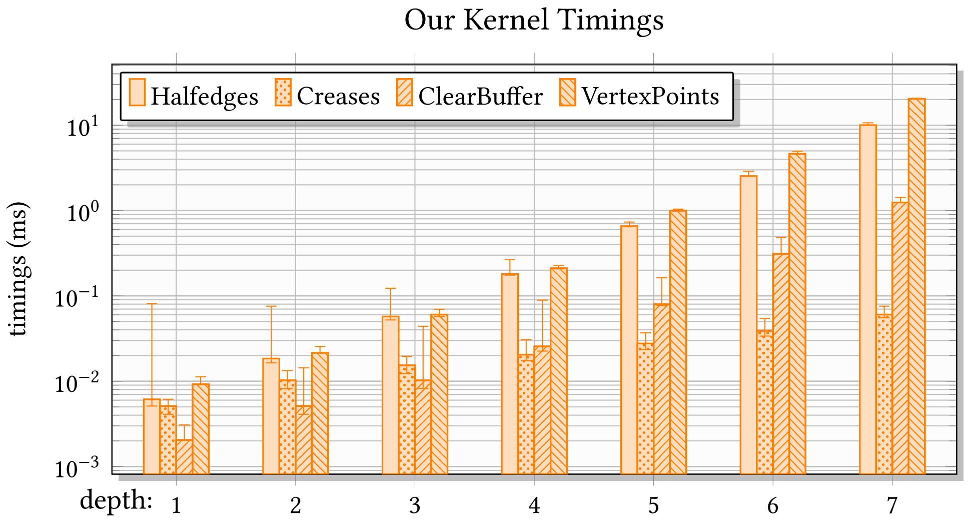


$\begin{pmatrix}\text{24 boundaries}\\ \text{280 creases}\end{pmatrix}$

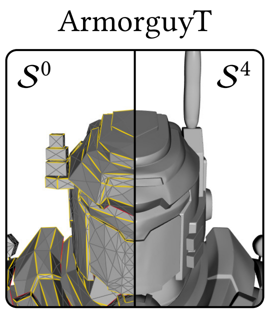
$H_0 = 4,530$	$H_4 = 1,159,680$
$F_0 = 1,510$	$F_4 = 386,560$
$E_0 = 2,277$	$E_4 = 580,032$
$V_0 = 768$	$V_4 = 193,473$





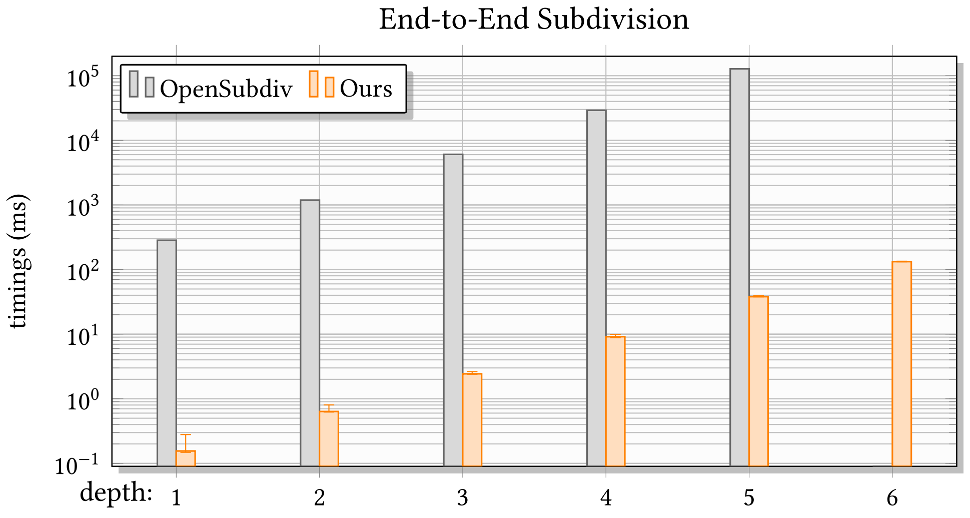


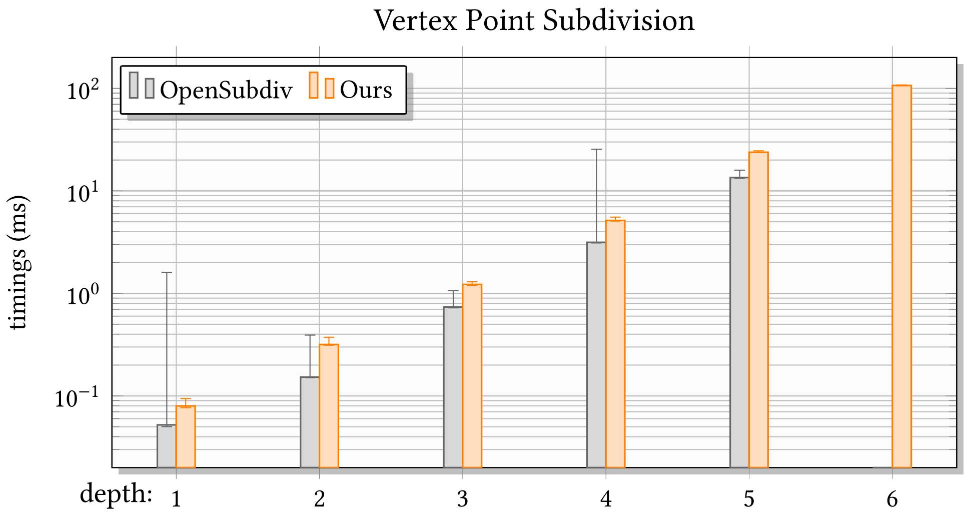
8 ArmorGuy

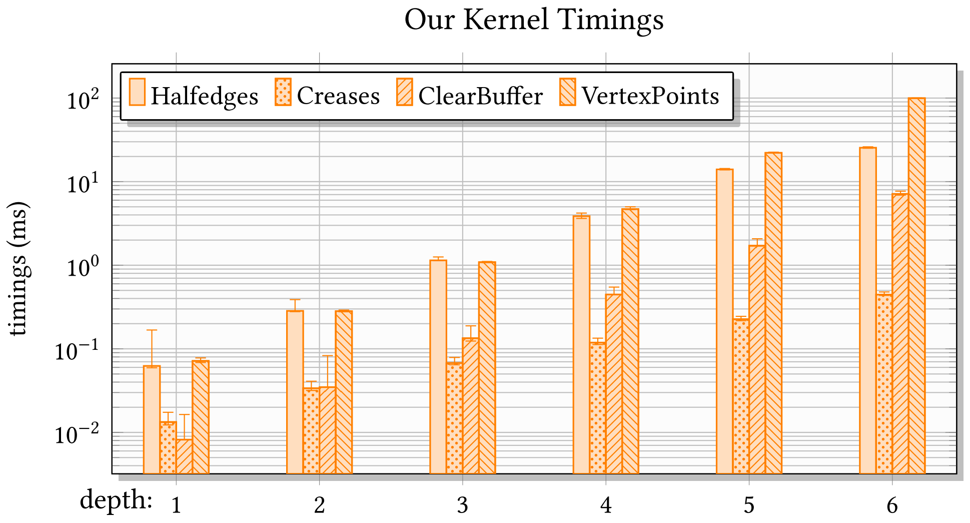


$\begin{pmatrix}\text{2034 boundaries}\\ \text{7101 creases}\end{pmatrix}$

$H_0 = 101,736$	$H_4 = 26,044,416$
$F_0 = 33,912$	$F_4 = 8,681,472$
$E_0 = 51,885$	$E_4 = 13,038,480$
$V_0 = 18,423$	$V_4 = 4,357,458$


End-to-End Subdivision
OpenSubdiv
Ours
timings (ms)
10^{-1}
10^0
10^1
10^2
10^3
10^4
10^5
depth:


Vertex Point Subdivision
OpenSubdiv
Ours
timings (ms)
10^{-1}
10^0
10^1
10^2
depth:


Our Kernel Timings
Halfedges
Creases
ClearBuffer
VertexPoints
timings (ms)
10^{-2}
10^{-1}
10^0
10^1
10^2
depth: