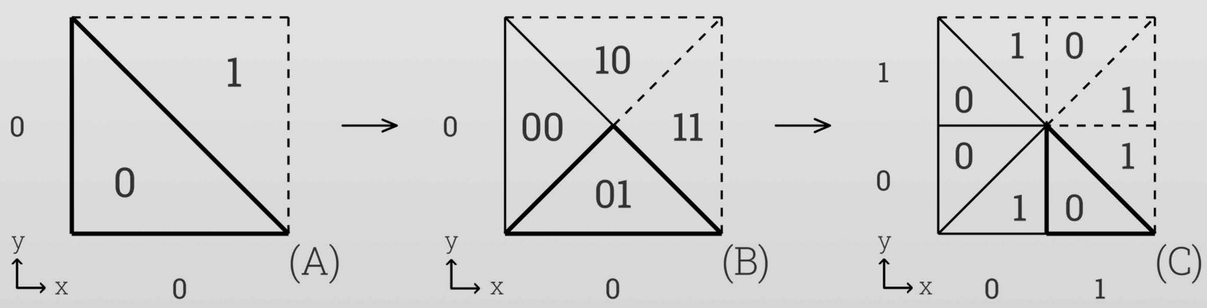
Improved Triangle Encoding for Cached Adaptve Tessellation

Linus Horváth (linus.horvath@gmail.com)
Bernhard Kerbl (kerbl@cg.tuwien.ac.at)
Michael Wimmer (wimmer@cg.tuwien.ac.at)



Triangle encoding using regular grids with progressive resolutions

Introduction

Hardware tessellation is not without its limitations, which have driven developers and researchers to investigate alternative, software-based solutions.

We propose a modification to the adaptive GPU tessellation method presented by Khoury et al., in order to increase its efficiency on current hardware. Instead of generating detail geometry anew in every frame, the authors' original design produces tessellated primitives incrementally by recursively splitting triangles over multiple frames. Resulting tessellated triangles are written to a GPU buffer, hence they must be encoded to economize on-chip memory.

The authors' triangle encoding method requires only two integers (one for the original primitive and one tessellation key) per stored triangle and is derived from the compact representation by Gargantini for linear quadtrees [1], which have recently been ported to the GPU for real-time rendering [2, 3].

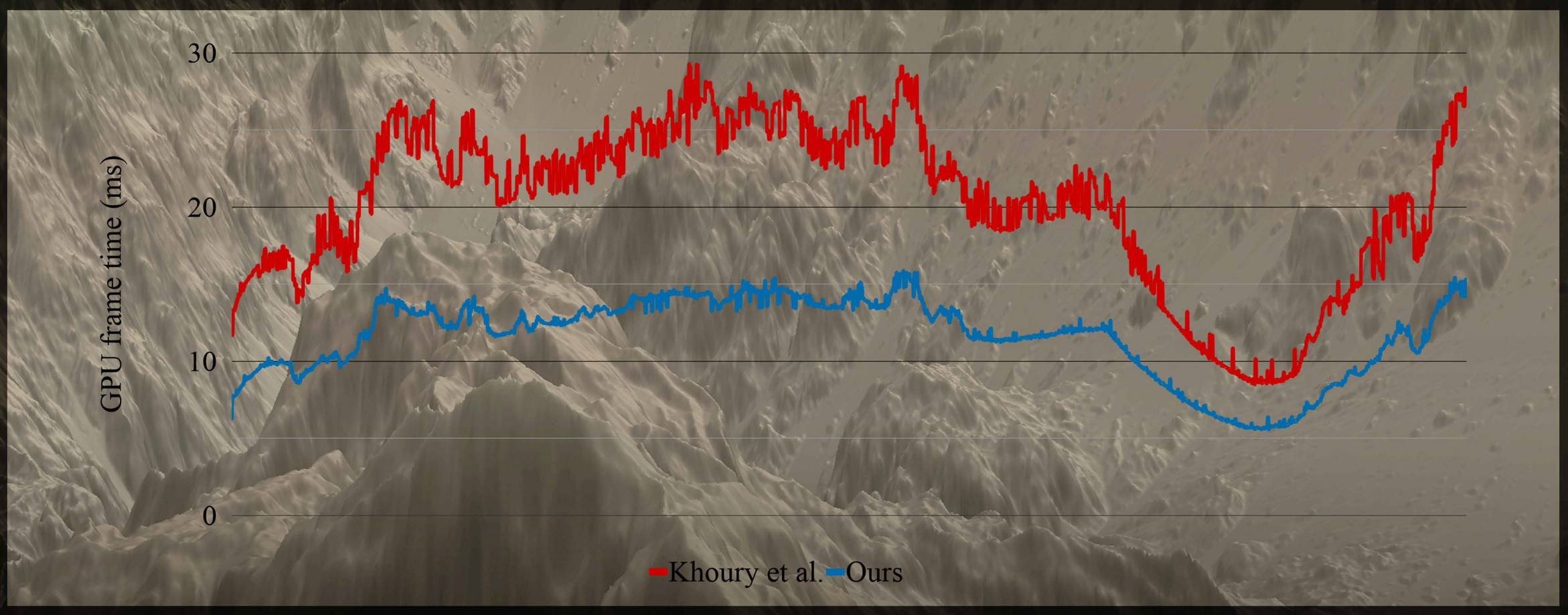
Specifically, the authors propose a key layout wherein each bit represents a transformation of barycentric coordinates. The key is decoded by recursively applying transformations to obtain barycentric coordinates that uniquely identify each subtriangle.

Modifications

We can interpret each triangle as part of a quad in regular grids with different resolutions/states. From the position of the most significant bit (MSB), the state and quad grid resolution can be directly inferred. An even MSB position indicates 2-state, and an odd one indicates 4-state. Subtracting the proper number of region bits and dividing by 2 gives the quad grid resolution for this triangle in *x* and *y*. The quad coordinates are stored *(x,y)* in the corresponding lower bits. The region code interpretation is more involved and depends on both quad coordinates in order to support fully recursive subdivision.

We have evaluated the performance of our new encoding scheme in the original OpenGL framework by Khoury et al. by modifying their implementation of adaptive tessellation via compute shaders. We consider the total frame time using the original and our modified encoding scheme. The chart below compares the rendering performance with our encoding scheme to the unaltered version. Timings were obtained by measuring the total GPU time in each frame over an animated camera path with both methods separately. All timings were recorded at 1080p on an NVIDIA RTX 2070 GPU.

Our constant-time decoding procedure can reduce frame times by up to **40%**, thus reaching real-time performance (<17ms per frame).



[1] Irene Gargantini. An effective way to represent quadtrees. *Commun. ACM*, 25(12):905–910, December 1982.
[2] Jonathan Dupuy, Jean-Claude Iehl, and Pierre Poulin. Quadtrees on the gpu. *GPU Pro: Advanced Rendering Techniques*, 5:211–222, 10 2018.
[3] Wade Brainerd, Tim Foley, Manuel Kraemer, Henry Moreton, and Matthias Nießner. Efficient gpu rendering of subdivision surfaces using adaptive quadtrees. *ACM Trans. Graph.*, 35(4), July 2016.
[4] Jad Khoury, Jonathan Dupuy, and Christophe Riccio. Adaptive gpu tessellation with compute shaders. *GPU Zen: Advanced Rendering Techniques*, 2:3–17, 2019.