Recursive Tessellation

Hyunjin Lee Yuna Jeong Sungkil Lee
Sungkyunkwan University*

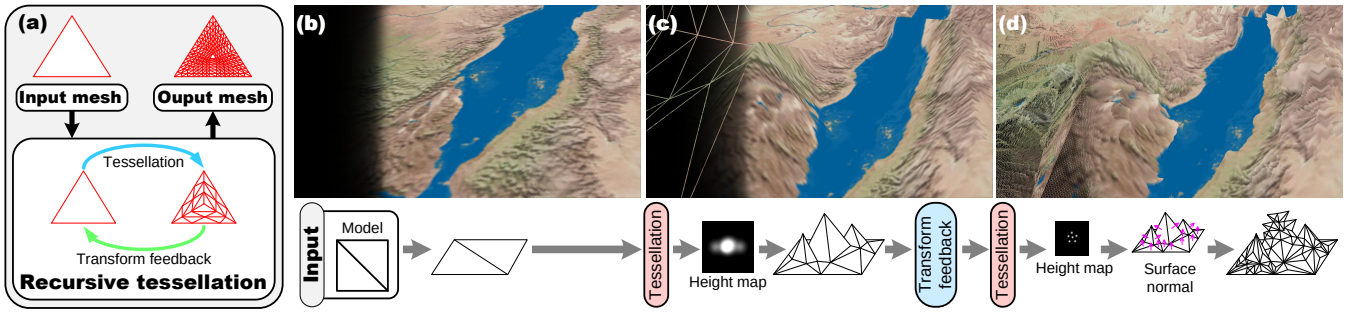

Figure 1: *Overview of recursive tessellation (a). A coarse input mesh (b) is tessellated with initial displacement mapping (c), and is stored into a transform-feedback buffer. Then, the tessellated mesh can be re-tessellated with another displacement map along its new normals (d). In this way, recursive tessellation can represent high details beyond the typical limits of GPU even with multiple levels of displacement mapping.*

1 Introduction

Tessellation shaders, available from the latest graphics processing units (GPUs), are increasingly important in high-fidelity real-time rendering. The tessellation shaders allow us to subdivide a coarse input patch to tiny primitives with subdivision schemes such as Loop subdivision [Loop 1987], Catmull-Clark subdivision [Catmull and Clark 1978], and Phong tessellation [Boubekeur and Alexa 2008]. The tessellated geometries represent smoother surfaces, but also their details can be enhanced with displacement mapping for on-the-fly surface elevation. Each output vertex is relocated based on the height information, which refines the geometric details such as peaks, craters, terrain, and wrinkles.

Whereas the hardware tessellation is sufficiently fast even for high-resolution outputs, the output resolution might not be enough for high-resolution subdivision due to the hardware limits of maximal tessellation factor; in the latest GPUs, the tessellation factor is limited up to 64. Hence, it is often necessary to pre-tessellate the input mesh outside the GPU pipeline (i.e., offline or CPU processing), resulting in less-structured or hard-to-manage systems.

2 Our Approach

In this paper, we present a novel tessellation scheme, which performs the tessellation and transform feedback (or stream-out in DirectX) *recursively*. The transform feedback enables us to store the tessellated mesh without redundant rasterization overhead. As a result, this strategy allows us to (virtually) infinitesimally subdivide a coarse mesh by performing multi-stage tessellation, which can lead to extremely detailed geometric representations. Also, the tessellated mesh can be a cache for on-demand tessellation, which may vary by level of details (LODs), and further serve as a fundamental building block for highly dynamic adaptive LOD management.

Another important benefit of the recursive tessellation is the use of multi-stage displacement mapping, which is not straightforward in a single tessellation instance (e.g., tangential displacement against a base plane or overhanging edges; see Figure 1 for example). With respect to the new normals of the tessellated meshes, we can apply another instance of displacement mapping recursively, which significantly enhances high-frequency details against the initial elevation.

*e-mail: {hyunjinlee, jeongyuna, sungkil}@skku.edu

3 Implementation and Performance

In conventional GPUs, the tessellation shaders are executed as additional stages after vertex-shader stage. The degrees of tessellation are controlled in terms of internal and edge subdivision in the tessellation control shader (TCS), and are passed to the tessellator in which new child primitives are generated. Then, the new primitives are transformed based on the attributes of their parent patches in the tessellation evaluation shader (TES). In this TES stage, the displacement of each new vertex can be applied using the offsets read from an external texture. Finally, the tessellated and displaced primitives are written in a transform-feedback buffer.

This combined operation of tessellation and transform feedback is treated as a single instance at a recursion depth, and thus, can be repeated to produce high-detail geometries. Meanwhile, we can apply displacement mappings at different levels or from source textures. Once the resolution of the buffer reaches the desired degree of details, the recursion stops at the frame. For the next frames, we can either more tessellate it or trace back to its lower-detail parent.

Finally, we report rendering performance of our recursive tessellation, implemented on an Intel i7 machine with an NVIDIA GTX 680 at 1280×720. The framerates reached 2910, 2850, and 2820 fps for meshes with 300K, 500K, and 1M triangles, respectively, which were generally higher than the rendering without tessellation for the same number of triangles.

Acknowledgements. This work was supported by the Basic Science, Mid-career, and Global Frontier (on Human-centered Interaction for Coexistence) R&D programs through NRF grants funded by the Korea Government (MSIP) (Nos. 2011-0014015, 2012R1A2A2A01045719, and 2012M3A6A3055695).

References

BOUBEKEUR, T., AND ALEXA, M. 2008. Phong tessellation. In *ACM Transactions on Graphics (TOG)*, vol. 27, ACM, 141.

CATMULL, E., AND CLARK, J. 1978. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-aided design 10*, 6, 350–355.

LOOP, C. 1987. Smooth subdivision surfaces based on triangles.