Rendering Point Clouds with Compute Shaders
Markus Schütz, Michael Wimmer, TU Wien

Abstract
Regular point rasterization with glDrawArrays(GL_POINT,...) can be slow due to the overhead of the rendering pipeline. Compute shaders with atomicMin and atomicAdd are often a faster alternative.

Method 1: Compute
A compute shader transforms points to pixel coordinates, and then encodes linear depth and color into a 64 bit integer. With atomicMin, we store the fragments with the lowest depth in a pixel buffer. A second compute shader transfers the pixel buffer into a texture.

Method 2: High-Quality
First, create a depth buffer with method 1. Then, use atomicAdd to sum up and count color values of points at most 1% behind depth buffer. Finally, divide sum of colors by number of fragments to get an average color value of overlapping points in a pixel. Compute shader implementation of Botsch et al. [1].

Results
• Our compute and the classic GL_POINTS method produce the same result
• The basic compute method is up to 2x to 10x faster than GL_POINTS
• The high-quality method is up to 2x to 4x faster than GL_POINTS
• Evaluated for point sizes of 1 pixel
• GL_POINTS still faster for point sizes larger than 2x2 pixels

Dataset: San Simeon, 117M points, courtesy of PG&E
Code: github.com/m-schuetz/compute_rasterizer
Video: bit.ly/2nv48gl

Acknowledgements
The authors wish to thank PLE LMS for the point cloud of Retz, PG&E and OpenTopography for funding and hosting the point cloud of San Simeon (Morro Bay), and the Ludwig Boltzmann Institute for Archaeological Prospection and Virtual Archaeology for the point cloud of the Heidentor.

References / Related Work