http://cg.tuwien.ac.at
mschuetz@cg.tuwien.ac.at
wimmer@cg.tuwien.ac.at

Progressive Real-Time Rendering of Unprocessed Point Clouds

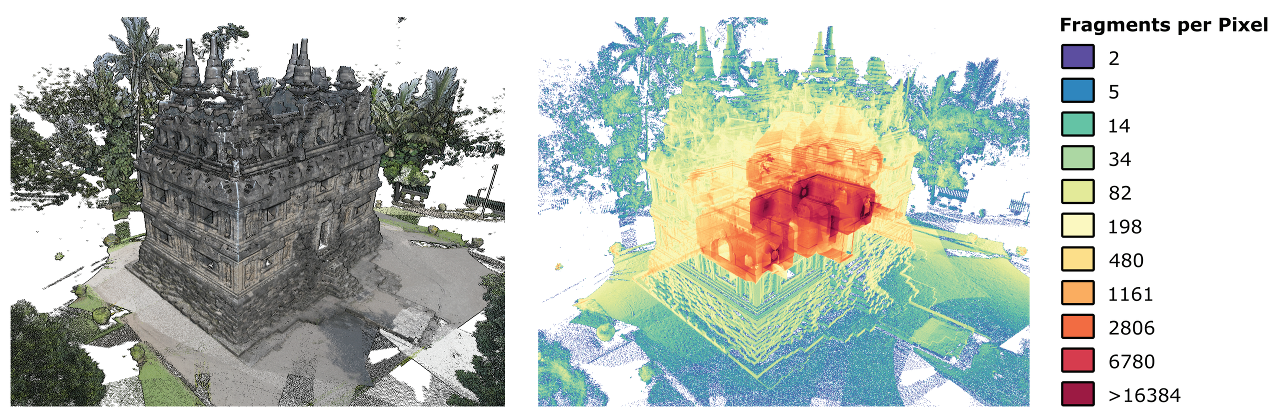
Markus Schuetz, Michael Wimmer
TU Wien

GENERATIONS SIGGRAPH2018

Problem

Rendering tens of millions of points in real time usually requires high-end GPUs or the use of spatial acceleration structures. However, even high-end GPUs are limited in how many points they can draw in real time and generating acceleration structures requires a preprocessing step.

Two major bottlenecks of point cloud rendering are the significantly larger amount of vertices that are necessary to achieve a similar level of detail as textured polygon models, and the large amount of overlapping fragments that are generated from said vertices.



(Candi Sari, 250M points, many of them projected onto the same pixels)

Our Approach ...

- distributes the workload of rendering a single, large blob of points over multiple frames, without the need to generate acceleration structures in advance.
- reuses details that were already drawn in previous frames and progresses uniformly towards the finished result, typically in less than a second.
- is designed to work while points are being loaded or scanned so that users can immediately see results.
- uses a single, randomly shuffled array of points as its data structure. Shuffling happens incrementally while points are loaded.
- allows users to explore any point cloud that fits into GPU memory in real time.

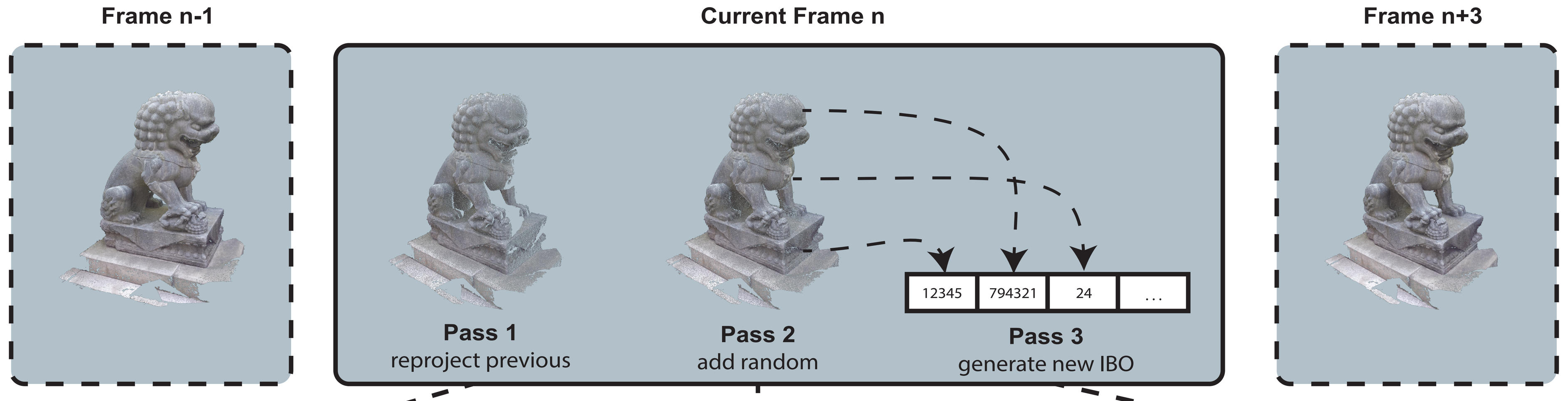
Related Work

- Futterlieb et al. developed a method that accumulates detail when the camera is still and creates a new vertex buffer from visible points in discrete intervalls, in order to preserve the accumulated details when the camera moves again [1]. Our method differs in that we create an index buffer every frame, instead of a vertex buffer in discrete intervalls.
- Similar to our approach, Ponto et al. reprojects every frame to the next, but they add nodes of a hierarchical structure, instead [2]. As such, it converges faster but in non-uniform way, and it requires a hierarchical structure.

Method

Our method consists of two parts, the progressive rendering and the incremental generation of a shuffled vertex buffer object.

Progressive Rendering



Render only points that were visible in the previous frame. This limits the number of points drawn to the number of pixels, which heavily reduces workload.

This pass uses glDrawElementsIndirect because the index buffer object (IBO) and the draw parameters were generated directly on the GPU by pass 3 of the previous frame.

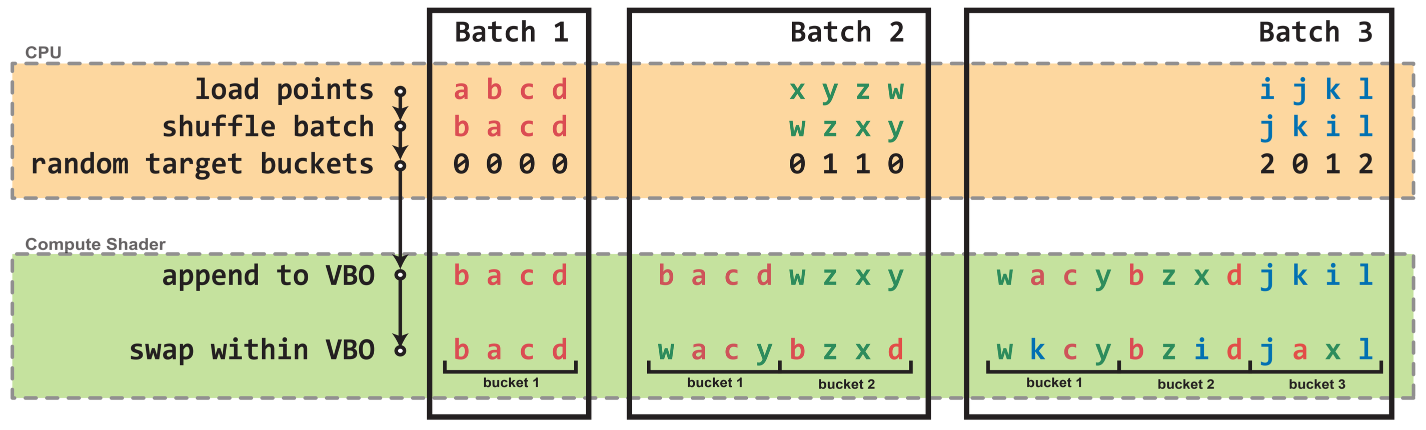
Add a random selection of points to fill gaps that appear after transformations. A random selection results in a more uniform progression to the final result. In each frame, at most (*numPixels + numAdded*) points are drawn.

Random selections are obtained by drawing subranges of a shuffled vertex buffer object (VBO).

Generate an index buffer from all currently visible points, which is used in Pass 1 of the next frame to render only what is visible now.

Pass 1 and 2 also write the point indices to a hidden color attachment. Pass 3 is implemented as a compute shader that executes for each pixel and writes the respective point indices to the new index buffer.

Incrementally building a shuffled VBO



We use an incrementally shuffled VBO in order to efficiently pick random points during rendering, even while points are being loaded. This is done by shuffling each new batch of points, appending it as a new bucket to the VBO, and finally swapping the newly appended points with other points in random buckets. Swapping between buckets is necessary because shuffling just the batch preserves locality within that batch. Race conditions are avoided because each point of the appended bucket is swapped with a point at the same relative position of a random target bucket.

This method maintains a sufficiently randomly shuffled array, without having to reshuffle all previously loaded points in each step. Instead, only *#batchSize* points are shuffled and swapped with each batch.

Results

We benchmarked three point clouds on three GPUs and compared the rendering performance of our progressive approach to the performance of a bruteforce (all points every frame) approach. For the bruteforce timings, we used the unshuffled VBO since it renders faster that way.

To exploit higher performance of some GPUs, we add 3M points per frame on the 1080 GTX, 2M on the 1060 GTX, and 1M on the 940 MX. Higher values increases rendering times but reduces the number of frames until convergence.

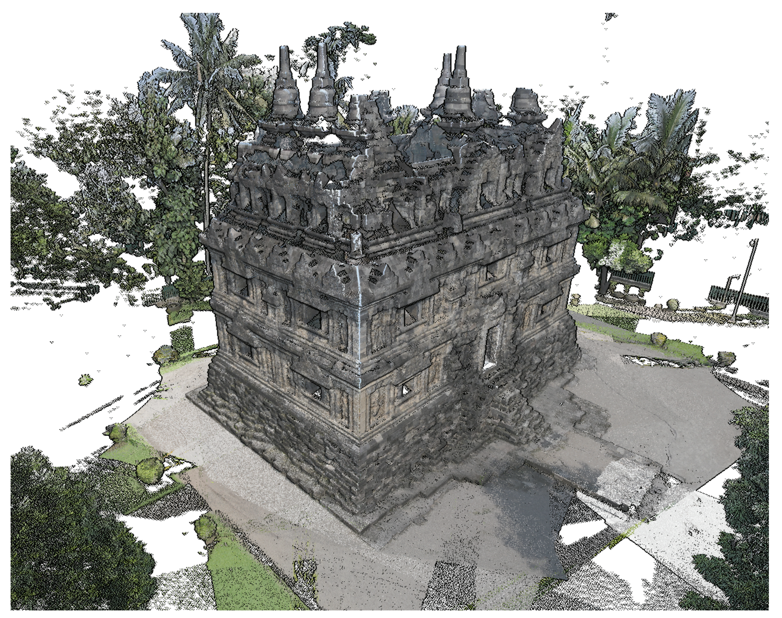
Code samples available at:
https://github.com/m-schuetz/siggraph2018

Heidentor, 26M points

GPU	bruteforce	ours	#add	converges in
1080 GTX	8.483ms	2.154ms	3M	9 frames / 0.02s
1060 GTX	13.554ms	3.414ms	2M	13 frames / 0.05s
940 MX	37.311ms	11.281ms	1M	26 frames / 0.30s

Retz, 120M points

GPU	bruteforce	ours	#add	converges in
1080 GTX	46.289ms	2.892ms	3M	40 frames / 0.12s
1060 GTX	59.736ms	5.642ms	2M	60 frames / 0.34s
940 MX		<not enough GPU memory>		



Candi Sari, 250M points

GPU	bruteforce	ours	#add	converges in
1080 GTX	98.459ms	2.744ms	3M	83 frames / 0.23 s
1060 GTX		<not enough GPU memory>		
940 MX		<not enough GPU memory>		

References

[1] Jörg Futterlieb, Christian Teutsch, and Dirk Berndt. 2016. Smooth visualization of large point clouds. IADIS International Journal on Computer Science and Information

[2] K. Ponto, R. Tredinnick, and G. Casper. 2017. Simulating the experience of home environments. In 2017 International Conference on Virtual Rehabilitation (ICVR). 1–9. https://doi.org/10.1109/ICVR.2017.8007521

Acknowledgements

We would like to thank the following institutions for providing the respective data sets:
- Heidentor: Ludwig Boltzmann Institute for Archaeological Prospection and Virtual Archaeology
- Retz courtesy of *RIEGL* Laser Measurement Systems
- Candi Sari courtesy of TU Wien, Institute of History of Art, Building Archaeology and Restoration