High performance city rendering in Vulkan

Alex Zhang
Fraunhofer Singapore
alex.zhang@fraunhofer.sg

Kan Chen
Fraunhofer Singapore
chen.kan@fraunhofer.sg

Henry Johan
Nanyang Technological University
Fraunhofer IDM@NTU
henry.johan@fraunhofer.sg

Marius Erdt
Nanyang Technological University
Fraunhofer Singapore
marius.erdt@fraunhofer.sg

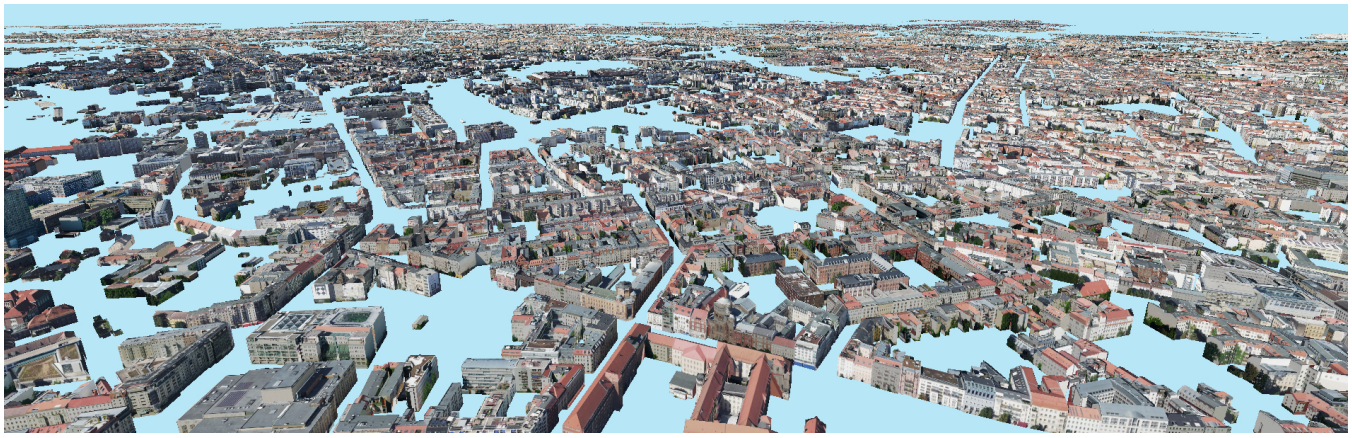

Figure 1: Rendering a portion of the Berlin city dataset[1] with 1.48M draw calls at 30FPS on a GTX 1080Ti. After CPU and GPU culling, 88K textures were streamed and rendered concurrently. Berlin city dataset used with permission.

ABSTRACT

City scale scenes often contain large amounts of geometry and texture that cannot altogether fit on GPU memory. Our ongoing work seek to minimise texture memory usage by streaming only view-relevant textures and to improve rendering performance using parallel opportunities offered by Vulkan, the latest generation of graphics API. Our result presents a high performance rendering of a city with streaming textures after CPU and GPU culling.

CCS CONCEPTS

• **Computing methodologies** → **Rendering**;

KEYWORDS

Massive rendering, texture streaming, Vulkan

ACM Reference Format:
Alex Zhang, Kan Chen, Henry Johan, and Marius Erdt. 2018. High performance city rendering in Vulkan. In *Proceedings of SA '18 Posters.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3283289.3283342

SA '18 Posters, December 04-07, 2018, Tokyo, Japan

ACM ISBN 978-1-4503-6063-0/18/12.
https://doi.org/10.1145/3283289.3283342

1 INTRODUCTION

Real-time rendering of massive cities is a desirable feature in the Geographic Information System (GIS) domain, offering many applications for city planning and information portals. However, large textured cities are computationally expensive to render due to CPU overheads in draw calls and memory constraints. For the Berlin city dataset[1], 3D models are heavily fragmented with one or more meshes for each building's face, one texture per mesh. This large data leads to an inflated count of more than five million textures with a total size of 441GB in raw RGBA format.

Recent work in web-based virtual maps [El Haje et al. 2016; Prandi et al. 2015] addresses large 3D data through progressive streaming of a smaller subset. These 3D contents are spatially partitioned into tiles and transmitted to client-side based on each tile's expected visibility within a camera frustum. Within the gaming industry, authors have developed virtual textures [Chen 2016; Obert et al. 2012] to overcome storage and performance constraints when using a large number of textures. Their methods designate entire scene textures (including mipmaps) as a set of tiles in a virtual address space, with only a small relevant subset mapped to the device's address space for rendering.

On a system level, the Vulkan graphics API aims to improve CPU performance through efficient batch updates for drawing and state changes. It also offers CPU parallelism for rendering and

[1] https://www.businesslocationcenter.de/en/downloadportal

resource transfer, which we exploit for texture streaming. Our current implementation performs broad phase culling on the CPU and narrow phase culling on the GPU before textures are selected for loading and transfer. To minimise colour discrepancies between loaded and yet to be loaded mesh textures, we propose to use a precomputed city colour map to apply an initial colour to all meshes.

2 OUR APPROACH

To minimise device memory usage, only meshes that are visible to the camera will have their textures loaded. This is determined using a broad and a narrow phase of geometry culling methods. The broad phase CPU culling first spatially partitions the scene using a quadtree, and then performs in parallel a frustum and AABB intersection test on each partition's mesh-fitted bounding box. Meshes in visible partitions are then submitted through the rendering pipeline for narrow phase culling, which includes backface, frustum, and occlusion culling with early-z test [Kubisch and Tavenrath 2014]. The fragment shader simply outputs visible mesh IDs into a storage buffer, which in turn is accessed on CPU-side to load mesh textures.

In the next frame, textures are streamed concurrently from disk to device memory using the mesh ID list. This is done in batches by first loading texture images onto host (CPU) memory before transferring to the device. This batch processing has a predefined timeout (1 second) to prevent waiting on a large number of textures to load or unload, and a short timeout ensures loaded textures stay relevant to the changing camera view. To efficiently unload textures and free up memory, meshes with successfully loaded textures are stored in an unordered map, which can be unloaded when either a texture count or memory size threshold is reached.

Overall, our proposed rendering system has three concurrently operating parts. The first is rendering, the second is recording draw commands and the third is streaming texture images. Many issues arise in recording new drawing commands as Vulkan disallows updating texture descriptor sets already used in a command buffer submitted for drawing. To ensure valid synchronisation, two command buffers, each given a unique descriptor set, are swapped between each other, one to record new drawing commands while the other is submitted for rendering. In our implementation, command buffers are swapped as soon as the other is recorded so that texture changes are rapidly updated. This separation of processes allows rendering to be performed as fast as possible without being blocked by command recording or texture streaming.

Another feature of our system attempts to reduce mesh colour discrepancies for black meshes with yet-to-load textures. We use a precomputed city colour map that gives each mesh a relevant colour picked from the highest mipmap level of its intended texture, with each texel coordinate mapped to a mesh ID. This gives the scene a congruent appearance with little abrupt 'popups'.

3 RESULTS AND DISCUSSION

Our proposed system is able to render and stream textures of a Berlin city model containing about 5M meshes (13.5M triangles) and 5M textures in total. Frame rates and textures streamed per second is proportional to the number of draw calls. At 1920x1080 resolution with textures mostly loaded, a view submitting 302K, 723K, 5.04M draw calls runs at 119FPS, 57FPS, 12FPS respectively.

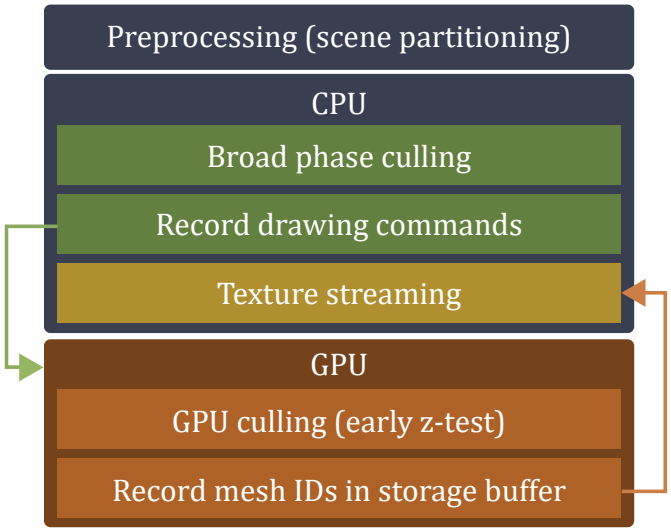

Figure 2: Overview of the rendering system. First, broad phase culling finds camera-visible quad-tree partitions, then submits those meshes for drawing. The fragment shader with early-z test outputs visible mesh IDs into an array, and those mesh's texture are subsequently streamed. These processes run at different rates with no visible stuttering.

Memory costs are fairly reasonable as loaded textures relates precisely to displayed fragments. A view that loads 390K textures (7.7% of the scene's total textures) requires 2.85GB (27%) of device memory. These textures are loaded at half resolution and precompressed using Block Compression (BC3) without sacrificing much image quality, and provides a 94% reduction in memory size (441GB down to 13.7GB). In the future, textures will be loaded more optimally at various mipmap levels based on a distance metric.

It is noted that meshes submitted for rendering after broad phase culling is not the most optimal for rendering performance as only about 10-20% of these meshes are visible after GPU culling. Also for a scene of such scale, many far meshes are sub-pixel in size and not rasterised. Future work in this area seek to cull about 50% more meshes on the CPU using clustered culling to further reduce GPU time and also to perform indirect drawing.

ACKNOWLEDGMENTS

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its International Research Centres in Singapore Funding Initiative.

REFERENCES

Ka Chen. 2016. Adaptive Virtual Textures. *GPU Pro 7: Advanced Rendering Techniques* (2016), 131.

Noura El Haje, Jean-Pierre Jessel, Véronique Gaildrat, and Cédric Sanza. 2016. 3D cities rendering and visualisation: a web-based solution. Eurographics.

C Kubisch and M Tavenrath. 2014. Opengl 4.4 scene rendering techniques. *NVIDIA Corporation* 5 (2014).

Juraj Obert, JMP van Waveren, and Graham Sellers. 2012. Virtual texturing in software and hardware. In *ACM SIGGRAPH 2012 Courses*. ACM, 5.

F Prandi, F Devigili, M Soave, U Di Staso, and R De Amicis. 2015. 3D web visualization of huge CityGML models. *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences* 40 (2015).