NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF RURAL AND SURVEYING ENGINEERING
LABORATORY OF PHOTOGRAMMETRY

3D GeoInfo 2020 & BIM GIS Integration Workshop
7 - 11 September, London

MULTI-THREADED RENDERING FOR CROSS-PLATFORM 3D VISUALIZATION BASED ON VULKAN API

Charalabos Ioannidis, Argyro Maria Boutsi

European Union
European Regional Development Fund
EPAnEK 2014-2020
OPERATIONAL PROGRAMME
COMPETITIVENESS • ENTREPRENEURSHIP • INNOVATION
ΕΣΠΑ 2014-2020
Partnership Agreement 2014 - 2020

PRESENTATION OUTLINE

TOPICS TO DISCUSS

Motivation & Graphics APIs

Aim & Challenges

Methodology

Implementation

Evaluation

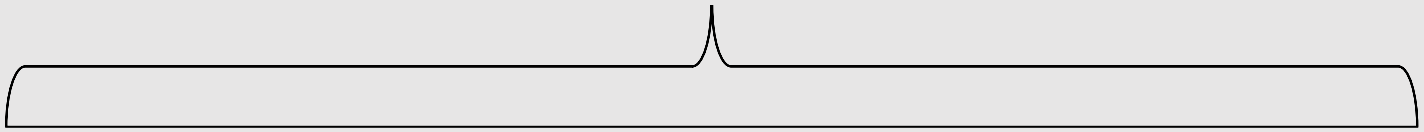
Conclusion

MOTIVATION

FIELD OF PHOTOGRAMMETRY & GEOMATICS

Need of (i) visualization of multi-source & high-dimensional 3D spatial data in a consistent way
(ii) maintenance of visual quality & geometric accuracy
Lack of (iii) dedicated hardware & high-end processing units

Characteristics of software for local rendering

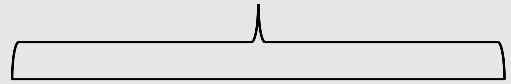


HIGH-PERFORMANCE PORTABILITY EXPLICIT CONTROL

INTRODUCTION

GPU PROGRAMMING & SOFTWARE PIPLINE

Graphics Processing Unit (GPU): Multi-core performance | Parallelism | Programmable functionality



GPGPU - Compute APIs:
CUDA & OpenCL

Graphics APIs

Program - controller of the overall process, user-level operations as creating threads, windows etc.

***Graphics API** - dispatch to the next layer, functions for passing commands in a standardized format*

Communication software layer - interface between the CPU & GPU, invisible to the developer

Application → Abstraction → Device Driver → Hardware

INTRODUCTION

GRAPHICS APIs

TRADITIONAL | **MODERN / LOW-LEVEL**

Cross-platform	Vendor-specific (Microsoft)	Vendor-specific (MacOS, iOS)	Cross-platform
GLSL shading language	HLSL	C++ 14 shading language	SPIR-V format
Driver overhead	Lower CPU overhead & reduced bottlenecks		
Cross-vendor unpredictability	More stable/predictable driver performance		
Memory & error management	Explicit, console-like control (synchronization & memory allocation)		

AIM & CHALLENGES

APP FOR 3D RENDERING & VISUALIZATION

AIM: Cross-platform 3D model viewer with multi-threading support based on modern C++ and Vulkan

- Running on Windows, MacOS & Android
- Suited to graphics hardware compatible with Vulkan's driver
- Rendering a high-resolution textured mesh (OBJ) into an interactive GUI

Challenges:

- Explicit rendering pipeline creation
- Memory allocation & resource (buffer & image) creation
- Synchronization
- Portability across mobile Android platforms

Methodology:

- Multi-threaded command buffer generation with synchronization primitives
- Render passes for adapting to mobile GPU's tiled-rendering

METHODOLOGY

RENDERING 3D GRAPHICS

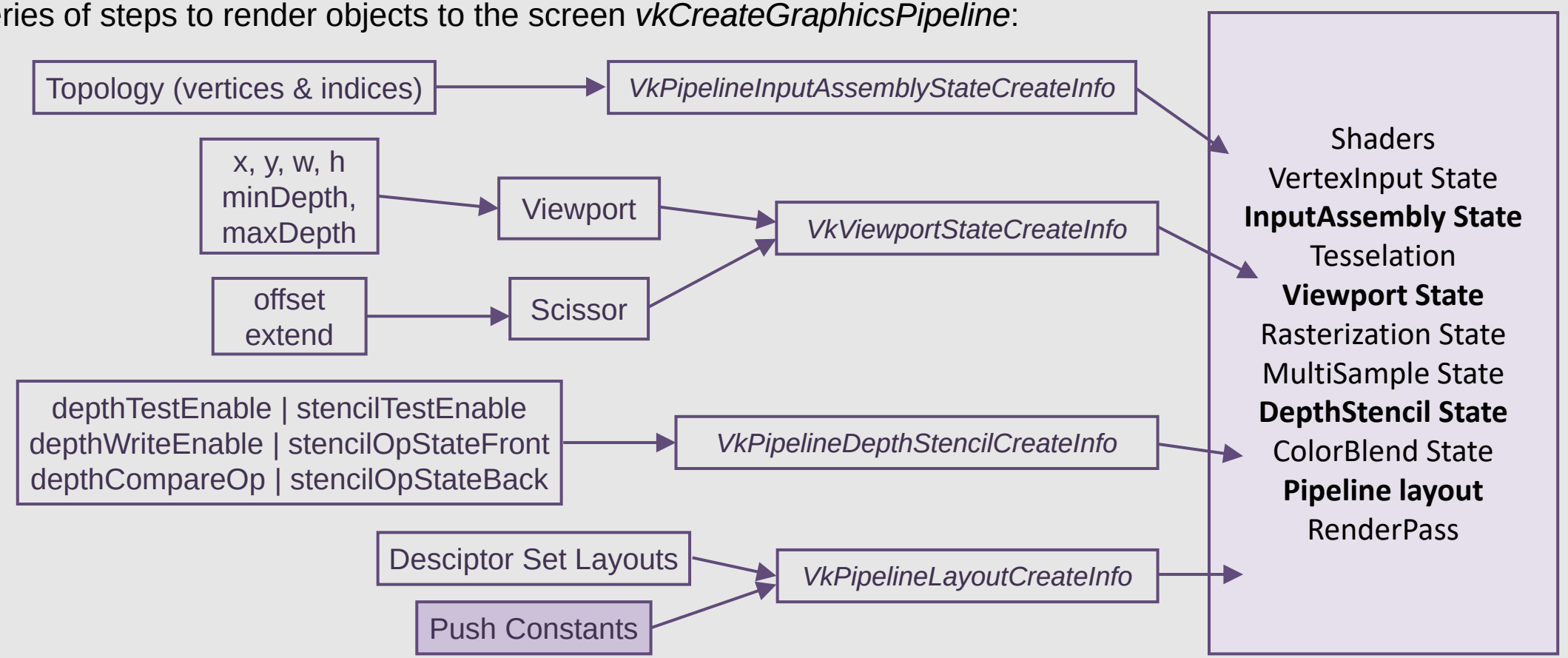
INITIALIZATION

- Create a *VkInstance* & select a supported graphics card (*VkPhysicalDevice*)
- Create a *VkDevice* and *VkQueue* for drawing and presentation
- Create a window, window surface and swap chain
- Wrap the swap chain images into *VkImageView*
- Create a render pass that specifies the render targets and usage
- Create framebuffers for the render pass
- Set up the graphics pipeline (*VkCreateGraphicsPipeline)*
- Allocate and record a command buffer with the draw commands for every possible swap chain image
- Draw frames by acquiring images, submitting the right draw command buffer and returning the images back to the swap chain

METHODOLOGY

GRAPHICS PIPELINE

Series of steps to render objects to the screen *vkCreateGraphicsPipeline*:



METHODOLOGY

RENDERING 3D GRAPHICS

INITIALIZATION

- Create a *VkInstance* & select a supported graphics card (*VkPhysicalDevice*)
- Create a *VkDevice* and *VkQueue* for drawing and presentation
- Create a window, window surface and swap chain
- Wrap the swap chain images into *VkImageView*
- Create a render pass that specifies the render targets and usage
- Create framebuffers for the render pass
- **Set up the graphics pipeline (*VkCreateGraphicsPipeline)***

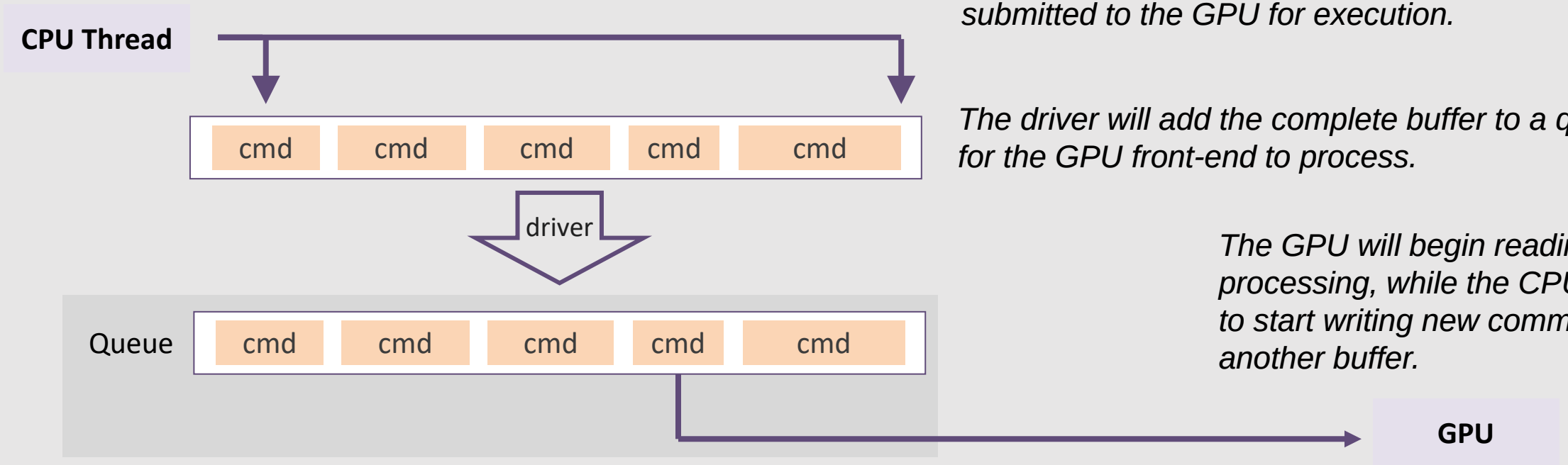
MULTI-THREADING

- Allocate and record a command buffer with the draw commands for every possible swap chain image
- Draw frames by acquiring images, submitting the right draw command buffer and returning the images back to the swap chain

METHODOLOGY

SINGLE-THREADED SUBMISSION

Command Buffers - *VkCommandBuffer:* Recording commands which are later submitted to a device for execution (draw/dispatch, texture uploads, etc)



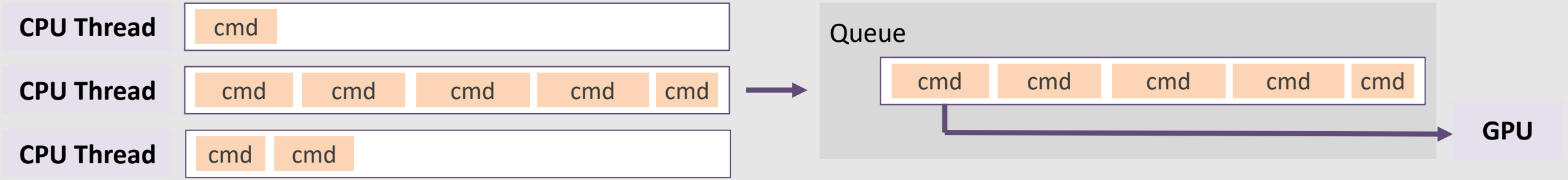
Once the command buffer fills up, it will get submitted to the GPU for execution.

The driver will add the complete buffer to a queue for the GPU front-end to process.

The GPU will begin reading & processing, while the CPU is free to start writing new commands to another buffer.

METHODOLOGY

MULTI-THREADED SUBMISSION & TILE-BASED RENDERING



Developed Multi-threading technique: Two levels of commands buffers (Primary & Secondary)

1. Drawing commands in the main thread through the rendering pipeline - Secondary command buffer recording in worker CPU's thread

2. Report of the completed operations to the primary command buffer

3. Last operation (i) ends the render pass

(ii) reports to the window surface that the frame is ready

(iii) updates the render state

METHODOLOGY

SYNCHRONIZATION & THREAD MANAGEMENT

Scheduling & synchronization of operations submission to the queue: Timeline Semaphore primitive

Role: (i) access of shared resources

(ii) control of submission order

Developed semaphore programming: Integration to the algorithms of queue operations

Rendering image views from the swap chain

Signal of single semaphore for multiple threads in multi-threading submission

METHODOLOGY

TILE-BASED RENDERING FOR MOBILE GPU

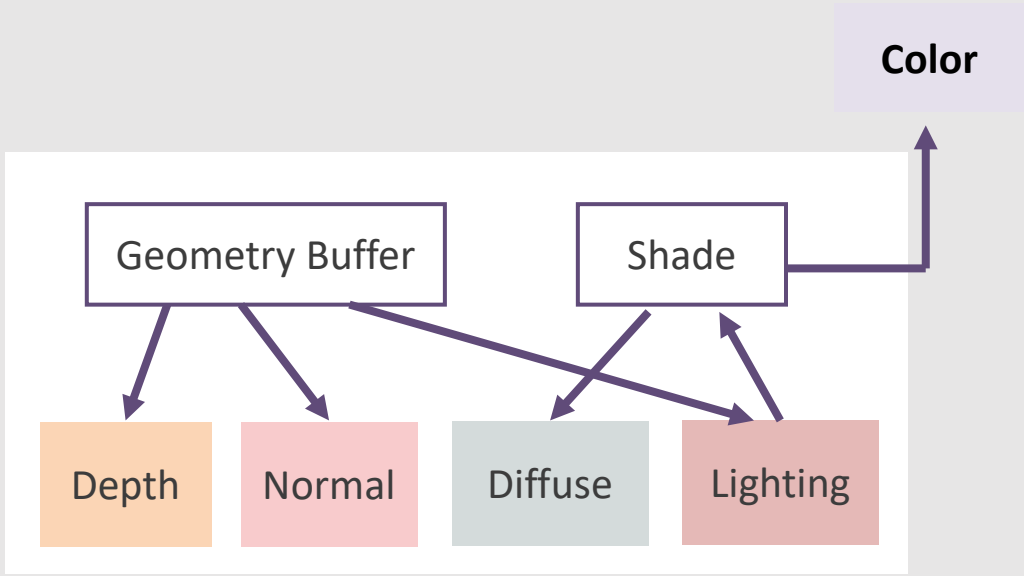
Multi-threading approach in mobile devices with tiled-based rendering:

(i) Recording of drawing commands in the secondary command buffers

(ii) Submission to the same render pass

Developed techniques:

- Multi-render passes for faster tile cache memory
- Merge of render passes on the same chip memory like pixel correspondences and shading

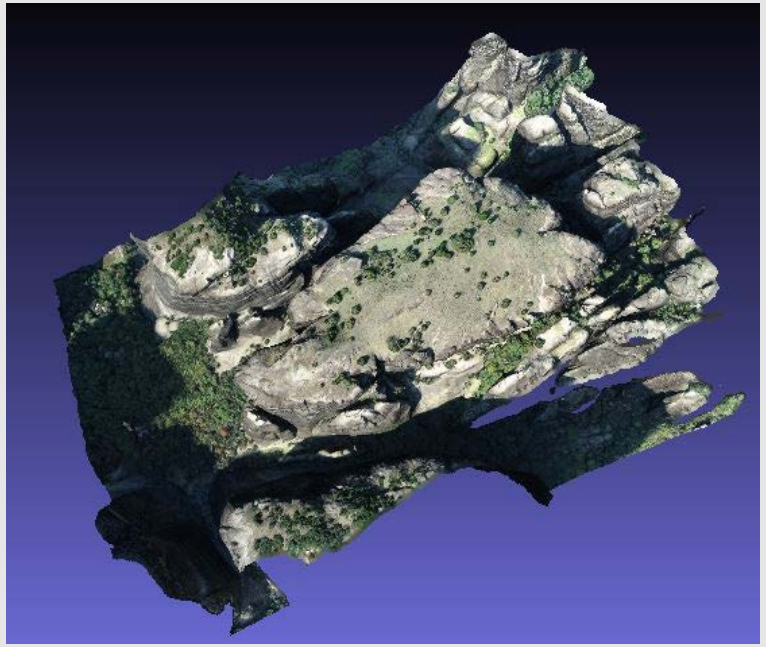


IMPLEMENTATION

CASE STUDY & THIRD-PARTY LIBRARIES

Case study: St. Modestos rock of the UNESCO site of Meteora, Greece

Format	Size	Vertices
OBJ	1 GB	4M



Tools & Libraries:

- Visual Studio IDE (Initial development)
- Vulkan SDK by LunarG
- GLFW: surface and events creation
- assimp: 3D model loading, parsing and storing in the program-specific format

IMPLEMENTATION

MULTI-THREADING PROGRAMMING & MULTI-PLATFORM SUPPORT

Rendering parallelization across four CPU cores & two levels of command buffers

- Primary Command Buffer:
  - (i) Recording of the workload with big state changes
  - (ii) Consuming the drawing calls for the visualization of the 3D model & its image texture
- Secondary Command Buffer: Building & dispatching draw calls within a render pass

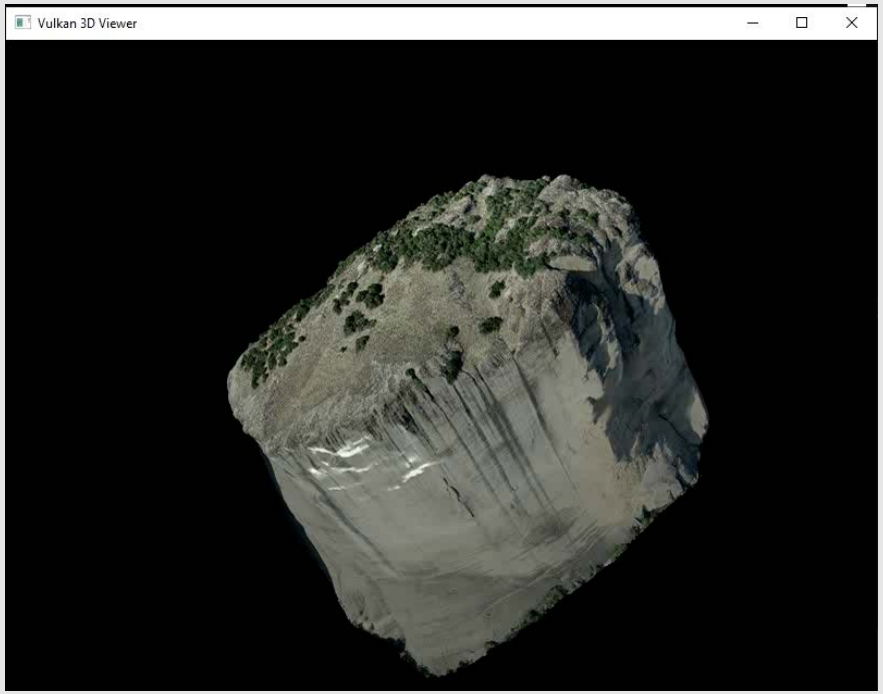
MacOS: MoltenVK runtime library (i) SPIR-V conversion to MSL

(ii) Vulkan mapping to Apple's Metal graphics framework

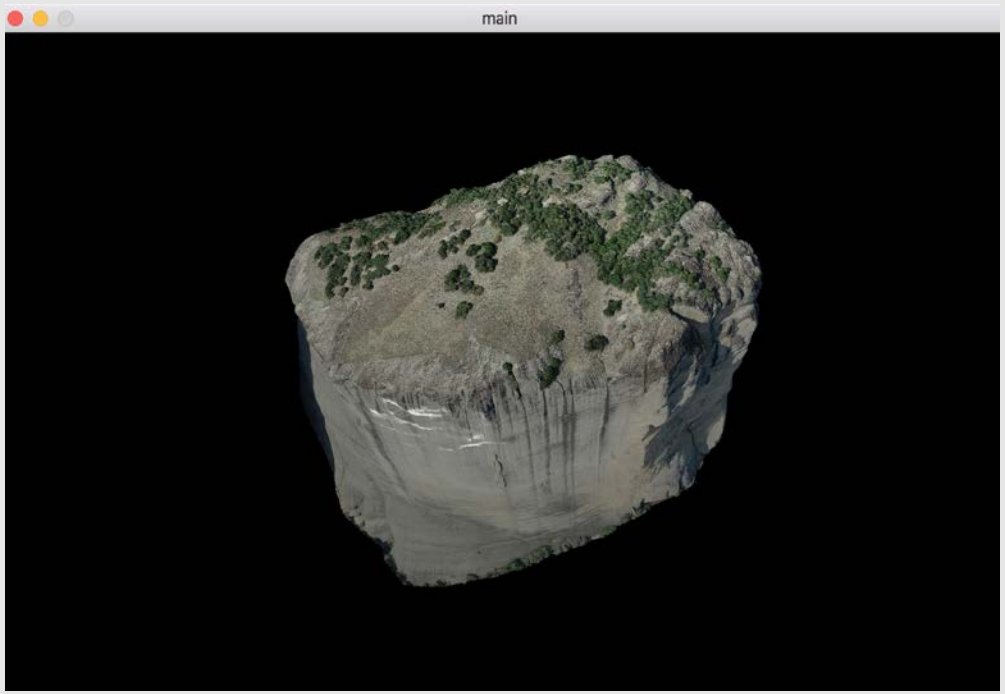
Android:Low-latency memory configuration - Android Studio IDE

IMPLEMENTATION

VISUALIZATION RESULTS



Windows



MacOS

Android

EVALUATION

MULTI-PLATFORM PERFORMANCE ANALYSIS

	Hardware specifications		Test Results	
	GPU/Memory	CPU	**FPS**	**Total CPU usage**
Windows 10	NVIDIA GeForce RTX 2070, 8 GB GDDR6	Windows 10	145	22,57 %
MacOS 10.15.4	AMD Radeon Pro 555X, 4 GB GDDR5	Windows 10	117	29,88 %
Android 9	Qualcomm Adreno 610	Windows 10	52	33,42 %

Aim: Efficacy of the developed rendering techniques & synchronization strategies on mutilple platforms and devices of various capabilities

Number of Cores	**Time (ms)**
Single -threaded	**454.07**
Two	**233.85**
Three	**173.52**
Four	**125.22**

CONCLUSION

OUTREACH & FUTURE WORK

OUTREACH

- Great portability to a multitude of devices & platforms
- High-degree of performance stability
- Adaptation to the implicit tile-based rendering
- Ability to handle large files & attain visual quality

Visualization cases & areas of interest:

Cultural Heritage | 3D cadastral | Urban planning | LiDAR data | 3D scanning products

FUTURE WORK

- Support of more 3D formats
- Ray-casting option for photo-realistic textures & advanced post-processing effects

Thank you for your attention

European Union
European Regional
Development Fund

EPAnEK 2014–2020
OPERATIONAL PROGRAMME
COMPETITIVENESS • ENTREPRENEURSHIP • INNOVATION

ΕΣΠΑ
2014-2020
Partnership Agreement
2014 - 2020

Meteora Project

@project_meteora

https://www.meteora.net.gr/en/contact-us/