VU Entwurf und Programmierung einer Rendering-Engine

Vorbesprechung UE

Goals of the exercise

- Deepen understanding…
- Course contents are wide-ranging
  - Thus, in the exercise students may focus on specific methods or applications
- Task can be chosen freely
  - Next lecture, we will discuss your ideas and fix the topics
- Implementation framework can be chosen by the student

Overview/mode

- Cool project (up to 2 students)
- Details in next slides
- Steps
  - Discuss ideas in lecture
  - Registration: TISS & short proposal
  - Implementation & Benchmarking
  - Hand in final report & source code
  - Abgabegespräch & oral exam

Problem definition

Three parts

1. Scene generation
2. Optimization technique for, either
   a. Rendering: rendering techniques typically involve the implementation of an acceleration data-structure, e.g. for culling **OR**
   b. Interaction: For use cases such as CAD tools or point cloud editing interaction performance (picking!) is crucial, also for this case we typically use acceleration data-structures.
   c. Large Scale Visualization: Large environment handling precision problems possibly including level of detail
3. Rendering

Validation and Documentation

- The implemented technique need to be analyzed in terms of performance
- Hand in a project report via email, two days before your oral exam appointment.

1. Scene Generation

- Use representative scenes to test your project
- For most algorithms, speed-up critically depends on geometric complexity
  - e.g. culling for very small geometries most likely will not result in speed up….
- Often, computationally generated geometry serves as good starting point
  - Sphere, cylinder, cone, hyperboloid, paraboloid,...
    - can be parametrized easily (e.g. by controlling subdivision levels)
  - Generated terrains
  - Randomly generated city
  - Provide mechanism to change parameters such as geometry/scene size
- You can also use real-world scenes or imported scene data, but make sure to test your implementation with
  - either extreme cases
  - or a set of representative test cases

2a. Optimization technique for rendering

- Depending on your use case, choose an appropriate implementation technique
  - For terrain rendering: culling or adaptive subdivision or (any other ideas?)
  - For CAD scenes: culling, gpu culling or optimization techniques for reducing driver overheads
  - For game scenes you might need handle dynamic and static content differently
- Optimizations:
  - Quad-Tree/Oct-tree for simple view-frustum-culling
  - BSP tree for transparent rendering
  - Occlusion culling (2D case would suffice here)
  - Level of detail (e.g. using subdivision surfaces with varying depth)
  - Geometric optimization (e.g. optimizing meshes for cache locality)
  - GPU accelerated computation of draw calls (e.g. compute shader)
  - GPU accelerated culling (e.g. compute shader)
  - Mechanisms for dealing with many many materials…

2b. Optimization technique for interaction

- For interactive applications such as CAD software or point cloud editing software, picking needs to be FAST.
- Optimization data structures could be:
    - Bounding volume hierarchies
    - Kd-Trees
    - Any other ideas...

3. Rendering

….. Of course….

- Your application should provide (simple!) interactive navigation in the scene
    - Don’t put too much effort into that. Setting the camera to interesting viewpoints for example is also sufficient
- What is not important
    - Tuned graphics effects (other lectures handle this well ;))

Benchmarks

- Check your optimization using representative parameters
  - For optimization data-structures most important parameter is: On/off
    - With and without view frustum culling
  - For trees
    - Depth (e.g. how long to subdivide)
    - Kd-Tree parameters
- Provide your benchmarks in the report
- Most important parameters should be be demonstrated at Abgabegespräch
  - Basis for discussion…
- Extra lecture on benchmarking

Don’t use debug builds and attached debuggers when benchmarking!!

First steps

- Up to 2 students per project team
- For teams of two project adapted project size
- We are happy to discuss your ideas before/during/after lectures
- Make sure to talk to the LV team if the project is OK (getting too much effort for such projects is easy ;))

Project proposal / registration

- Project proposal (max 1 page)
  - Short description: one liner
  - Your name/your team
  - Problem statement
  - Planned approach
  - Evaluation/benchmark methodology
- mail with project description and project team to rendEng@vrvis.at till 11.11.2019

Final report/results

- Hand in (via email, 2 days before exam)
- 2 pages per student
- Extend proposal with
    - Actual approach/used algorithms taken (if different from proposal)
    - Benchmark results
    - Discussion explaining results
        - Expected or unexpected results?
- The implementation (source code).
- If you want to show your result on our PC, provide a little description how to start it, so we can test it before the exam.
- Report and project will be discussed as part of the oral exam.
- Email to: rendEng@vrvis.at

Exam and test machine

- Appointments till 27.3.2020 (rendEng@vrvis.at)
- Email me with 3 possible dates/times 2 weeks before
  - Mo-Fr, 10-11:30, 14:00-16:30
- Email me your report+source code 2 days before the exam
- 1 question regarding the project, 2 questions from the lecture topics
- The exam is should be done in 10mins.
- If you don’t want to carry a laptop to the exam, send us instructions on how to run the program
  - We have a PC WIN10 and Arch Linux GTX980, and Radeon RX Vega

Programming environment

- You are free to choose whatever platform you want!!
- Using Aardvark reduces efforts for some tasks
- For other, rather low level tasks a custom application setup is more desirable
    - e.g. when using low level features such as vulkan generated commands…

- For aardvark users (but also project related questions) we provide support via gitter: https://gitter.im/aardvark-platform/RenderingEngineVU

Inspirations for cool projects

- Culling on the GPU using compute shaders to generate
  - Instance buffers (matrices, used for instancing geometry)
  - Indirect buffers
  - Command buffers (vulkan)
- Culling on the CPU
- Multithreaded scene traversal (UE4 style)
- Picking huge scenes
- Low level OpenGL hackery in order to handle huge scenes

  ….
- Rendering scenes with “huge” number of lights
- Accelerating rendering via (GPU?) occlusion queries

Meet us at: https://gitter.im/aardvark-platform/RenderingEngineVU

https://c1.staticflickr.com/9/8589/15821362994_25efd36e58_z.jpg01

Further reading

- **OpenGL Scene Rendering Techniques:** http://on-demand.gputechconf.com/siggraph/2014/presentation/SG4117-OpenGL-Scene-Rendering-Techniques.pdf
- **Siggraph BOF, interesting porting work of game engine developers for vulkan:** https://www.khronos.org/assets/uploads/developers/library/2016-siggraph/3D-BOF-SIGGRAPH_Jul16.pdf
- **Approaching the Zero Driver Overhead (AZDO talk)**, Everitt, Sellers, McDonald, Foley, Siggraph, GDC 2014, https://de.slideshare.net/CassEveritt/approaching-zero-driver-overhead
- OpenGL Efficiency: AZDO overview talk, https://www.khronos.org/assets/uploads/developers/library/2014-gdc/Khronos-OpenGL-Efficiency-GDC-Mar14.pdf
- **Optimizing the Graphics Pipeline with Compute**, Wihlidal (Frostbite) 2016, https://frostbite-wp-prd.s3.amazonaws.com/wp-content/uploads/2016/03/29204330/GDC_2016_Compute.pdf