VU Entwurf und Programmierung einer Rendering-Engine

Organization

186.166 - WS 2.0

Harald Steinlechner, Georg Haaser, Christian Luksch, Stefan Maierhofer
Organization

● Vorlesung
  ○ Monday, 16:15 (s.t.) - 17:45
  ○ Seminarraum 186, Institut für Computer Graphik und Algorithmen
  ○ ECTS efforts: approx half/half

● Übung
  ○ As a project, implement a module for a rendering engine
  ○ Topics can be chosen by students
Student project

- Extra slides for “Übungsteil”
- Similar to previous years:
  - Rendering and optimization a scene. This includes:
    - Geometry processing (e.g. Terrain generation, Meshes,...) or model loading
    - Acceleration data structure or optimization algorithm
    - Rendering of the scene
Exam

- **Hand in (per email) the project + a written report**
  - Till 2 days before the exam date
  - Written report (2-4 pages)
    - Description of the project
    - Description of the used techniques
    - Analysis of the performance

- **Oral exam**
  - End of January till end of March
    - Email with 2 possible dates to rendEng@vrvis.at
  - Demo of the project
  - Two questions of the lecture content
    - Details not that important, but understanding of the topics.
Contact

Harald Steinlechner

- VRVis Research Center, Donau-City-Straße 11
- rendEng@vrvis.at
- Register in TISS
- When projects/team is fixed: write email with task description to hs@vrvis.at

VO Homepage

- https://www.cg.tuwien.ac.at/courses/RendEng/
The mission of a rendering engine:

- Provide easy to use software components...
- which can be used to solve rendering engine tasks (like a toolbox)

In order to accomplish this, we need:

- Algorithms and Datastructures
- Graphics API & Hardware Insights
- API design
- Domain specific languages (e.g. scene description)
After the lecture you are able to...

- Analyse specific use case for rendering engines
- Structure reusable parts of a rendering engine
- Evaluate techniques and their trade offs including benchmarks
- Apply lighting and global illumination techniques to applications
Content of this LV

- Requirements for the design of rendering engines
- Hardware and Graphics APIs (OpenGL, Direct3D, Vulkan,..)
- Scene Representation (Scene graphs, display lists, command buffers,...)
- Static and Dynamic Data (Incremental Update Techniques)
- Optimizations (Caching, Culling, Level of Detail, Bounding Volume Hierarchies, Just-In-Time Optimization)
- Resource Management
- Domain Specific Languages (HLSL, Spark, FShade, Semantic Scene Graph,..)
- Reusable Components/Design for Rendering Engines
About the LV team & Aardvark

- LV Team is basically the aardvark core development team.
- Active development of the aardvark rendering engine since 2006 with Robert F. Tobler.
- Roberts mission: easy to use but high-performance rendering engine.
- Aardvark - An Advanced Rapid Development Visualization and Rendering Kernel
  - Heavily used in research + industry projects
Pinned repositories

**aardvark.base**
Aardvark is an open-source platform for visual computing, real-time graphics and visualization. This repository is the basis for most platform libraries and provides basic functionality such as dat...

- **C#** 74
- **6**

**aardvark.rendering**
The dependency-aware, high-performance aardvark rendering engine. This repo is part of aardvark - an open-source platform for visual computing, real-time graphics and visualization.

- **F#** 43
- **7**

**aardvark.media**
Functional (ELM style) front-end and UI for aardvark, an open-source platform for visual computing, real-time graphics and visualization.

- **F#** 20
- **7**

**walkthrough**
A walkthrough aardvark platform packages. Additionally to repository specific examples (e.g. aardvark.rendering) this repository shows the interplay of various aardvark platform packages.

- **F#** 10

**template**
project template for aardvark projects with build script for bootstrapping new aardvark projects (including all necessary dependencies).

- **F#** 4
- **3**

**aardvark.docs**
Simple examples combining multiple packages provided by the aardvark platform. Each platform repository comes with separate examples -- here we collect overarching examples using for example aardva...

- **F#** 76
- **4**
Seealso

https://aardvarkians.com/
Some of our projects

Live demo
<table>
<thead>
<tr>
<th>Year</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Till approx 2002</td>
<td>Aart (Obj C)</td>
</tr>
<tr>
<td>Approx. 2002</td>
<td>Ave (C++)</td>
</tr>
<tr>
<td></td>
<td>Traditional Scene Graph</td>
</tr>
<tr>
<td>Approx. 2005</td>
<td>Aardvark (C#)</td>
</tr>
<tr>
<td>Approx. 2008</td>
<td>Aardvark 2008 (C#)</td>
</tr>
<tr>
<td></td>
<td>Semantic Scene Graph [Tobler 2011]</td>
</tr>
<tr>
<td>Approx. 2010</td>
<td>Aardvark 2010</td>
</tr>
<tr>
<td></td>
<td>Lazy Incremental Computation</td>
</tr>
<tr>
<td></td>
<td>For efficient Scene Graph Rendering [Wörister et al. 2013]</td>
</tr>
<tr>
<td>Approx. 2015</td>
<td>Aardvark 2015</td>
</tr>
<tr>
<td></td>
<td>Composable Shaders [Haaser et al. 2014]</td>
</tr>
<tr>
<td></td>
<td>Towards Incremental Computation, Attribute Grammars for Incremental Scene Graph Rendering</td>
</tr>
<tr>
<td>Approx 2016</td>
<td>Fast and flexible!</td>
</tr>
<tr>
<td>aardvark.rendering</td>
<td></td>
</tr>
<tr>
<td>aardvark.base</td>
<td>General purpose incremental Computation, Incremental Rendering VM [Haaser 2015]</td>
</tr>
<tr>
<td>2017</td>
<td>Usability, Remote Rendering, Aardvark in the browser</td>
</tr>
<tr>
<td></td>
<td>Vulkan, ELM architecture, Aardvark goes web</td>
</tr>
</tbody>
</table>
Challenges

- **Size of data-sets**
  - Often requires out of core approaches

- **Dynamic and static geometry**

- **Efficient graphics hardware utilization**

- **Support for special effects**
  - APIs for accessing special hardware features
  - Provide mechanisms to specify for example shaders and post processing

- **Many different application areas: Focus on real-time applications**
  - Terrain, laserscan, reconstructed data, game levels
  - Architecture and planning
  - Light simulation (Global Illumination)
  - Games
  - Interactive Editing applications
Design Space

How to structure a rendering engine

- What interfaces and modules useful
- How to transfer data
- How to manage memory (we have GPU and main memory)
- How to store data in memory (e.g. for efficiency reasons)
- How to optimize, how to make use of multiple CPU cores

Graphics hardware specific questions

- What to compute in shaders
- What to compute on CPU (in what precision?)
What to expect

- Tools/Algorithms/Concepts to implement rendering engines
- Hardware/Graphics API insights
- How to structure rendering engine into modules
  - (Low cost) abstraction techniques
  - Compiler techniques
- Important data-structures in practise
  - k-d-Trees, Octrees
- Performance considerations
  - Optimizations (how to pack buffers etc)
  - Costs of programming language abstractions (e.g. can we afford virtual function calls?)
- How to manage large scenes (performance + memory)
- Approaches for implementing lighting/material systems
What to expect

● Dependencies and incremental computation for rendering engines
  ○ Efficient ways to handle dynamic data
● Scene representation
● Rendering of big scenes
  ○ Terrain-rendering, rendering precision, caches
● Parallelization for rendering engines

● Not content of this LV:
  ○ Graphics programming tutorial
  ○ How to use existing engines
  ○ How to implement concrete tooling (e.g. level editor, material editor)
Timeline

- 21.10.2019 - Scene representation
- 28.10.2019 - Optimization techniques for rendering engines
- 04.11.2019 - Data and rendering engines
- 11.11.2019 - Benchmarking, Representing fully dynamic scenes, Aardvark Tutorial
- 18.11.2019 - Optimization techniques for fully dynamic scenes
- 25.11.2019 - Domain Specific Languages for Rendering Engines, Composable Shaders
- 02.12.2019 - Materials and Lights for Rendering Engines,
- 09.12.2019 - Shading System and Global Illumination
  - Including lightmap packing, instant radiosity, deferred rendering techniques,
- 16.12.2019 - Questions regarding the lecture/project
Questions