DAG-based Rendering Pipelines

Taha Doğan Güneş
Ozyegin University, Istanbul, Turkey
tdgunes@gmail.com

1. Summary

DAG (Directed Acyclic Graph) pipelines for Terasology, is an extensive refactor of the current rendering engine of Terasology. While refactors are primarily considered to make the implemented code much cleaner and efficient, the proposed project includes enabling rendering engine to be able to support many different features that are not easily supported by the current implementation. Renderer extensibility, VR headsets, multiple cameras with different effects, minimizing state changes in OpenGL and reducing render time (by checking possible alternative DAG configuration) are features that will become available through this project.

2. The Project

The current rendering engine is monolithic and the rendering pipeline can only be altered by directly modifying the source code. However, organization of all rendering tasks can be represented as a directed acyclic graph. This representation not only enhances the quality of the code but also enables other developers to add their own effects into the rendering engine without making substantial modifications.

The proposed DAG-based rendering pipeline can include heterogeneous rendering tasks such as multiple rendering of a 3D scene, post-processing effects, shadow mapping etc, all represented by nodes in a DAG. Dynamic changes and real-time reconfiguration will be possible by accessing a node's attributes, adding, removing or replacing nodes.

To give an example of how a DAG-based rendering pipeline will improve the current renderer implementation, let's consider shadow mapping, a technique regularly used in 3D applications. Most basic implementations requires two passes: rendering the scene from the point of view of a light and then a pass to test if a fragment is in the shadow.

Shadow mapping section from current rendering engine: (WorldRendererImpl.java:341):

```
private void renderShadowMap() { // removed logging and checks for demonstration
            graphicState.preRenderSetupSceneShadowMap();
            shadowMapCamera.lookThrough();
```

```
            while (renderQueues.chunksOpaqueShadow.size() > 0)
                renderChunk(renderQueues.chunksOpaqueShadow.poll(),
                            ChunkMesh.RenderPhase.OPAQUE,
                            shadowMapCamera, ChunkRenderMode.SHADOW_MAP);

            for (RenderSystem renderer : systemManager.iterateRenderSubscribers())
                renderer.renderShadows();


            playerCamera.lookThrough(); // not strictly needed: just defensive
programming here.
            graphicState.postRenderCleanupSceneShadowMap();
        }
    }
```

In the current rendering implementation, the relationship between buffers is not clear and is cumbersome for a newcomer to make improvements. The DAG-based rendering pipeline approach clarifies the various rendering steps and their dependencies without exposing the underlying low-level graphics API.

Also, a mod developer might want to render high quality shadows with a ray tracing algorithm. Through the API of DAG-based pipeline, a mod developer will be able to manipulate the active pipeline's DAG and tweak or place his/her own module directly into the rendering process, as a node. The current rendering engine implementation does not allow for this. Aa mod developer is required instead to fork the entire Terasology project and manually change the hard-coded implementation.

3. Goals

a. Main Goals

i. Breaking the current pipeline in rendering tasks

Wrapping the tasks into nodes and having the existing pipeline working with a DAG.

ii. Enabling mod developers to add their own effects into the rendering engine

By adopting and implementing the concept of rendering pipelines as DAGs, it will become possible for mod developers to modify the existing pipelines. For example new effects might be added to a pipeline

by adding a custom node. Or existing effects will be replaced by better algorithms by replacing an existing node with a different implementation.

iii. Allowing multiple pipelines to coexist in the main pipeline.

Makes it possible to implement stereo rendering (dual cameras) for VR support, totally distinct pipelines (rendering a task that is not dependent on main pipeline) with different effects.

iv. Automatic elimination of redundant state changes

Some state changes in OpenGL are computationally expensive. Eliminating them would improve performance of the rendering engine.

v. Node Profiling Tool

Automatically seeing the graph and its bottlenecks would be really beneficial for developers, such as observing the effect of adding a new node. (This tool can also report how much time is spend on a single node.)

vi. Benchmark Report - Performance comparison of the proposed architecture with previous one.

It would be good to have a benchmark suite to automatically generate a performance report for this one.

vii. Estimating the fastest possible DAG configuration whenever alternatives are available

Assuming a DAG made of rendering tasks has alternative configurations leading to the same final image, it will be possible to benchmark each configuration on the fly, eventually selecting the fastest one.

b. Extra Goal

VR headset support (Rift, Vive) - GitHub issue #2111

Given the current state of the engine this goal is dependent on the refactoring detailed in section 3a. The goal here is to take advantage of the new rendering engine architecture and create nodes and pipelines supporting VR headsets, through cross-platform libraries such as OpenVR.

4. Timeline

Before May 23rd: (bonding period)

- Understand current implementation by studying the source code and interacting with the existing crew and mentors.
- Read:
  - Real-Time Rendering by Tomas Akenine-Möller, Eric Haines, and Naty Hoffman
  - Computer Graphics: Principles and Practice by John F. Hughes, Andries van Dam, Morgan McGuire
- Examine competing open source projects for inspiration and knowledge
- Tackle smaller project issues to get a feeling for the project culture and standards.

After May 23rd:

Week #	Date	Tasks

1	23rd May	- Complete a diagram that shows rendering steps of the current rendering engine.
		  - Show connections between dependent/independent steps of the engine.
		  - Identify the steps that can be transformed into “nodes”.
		  - Identify possible node types
		- Create usage scenarios

2	30th May	- Implement a very basic version of the proposed architecture
		  - Discuss a possible rendering API for mod developers with the crew.
		  - Start implementing basic rendering API with simple functionality.

3	6th June	- Implement camera node

4	13th June	- Implement post-processing modules
		  - Tone mapping
		  - Bloom
		  - Blur
		  - Adapt these modules to work with the current renderer by extracting previous implementation

5	20th June	- Wrap already-implemented rendering steps into nodes of a DAG - Shadow mapping

		- World reflections - SSAO and others - Refactor current GLSL implementations, if necessary
6	27th June	- Implement a benchmark tool - Add ability to save replays and load them - Implement an ability to generate benchmark report with given scenarios.
7	4th July	- Implement node profiling tool - Combine benchmark tool with node profiling - Add results of node profiling tool to benchmark tools' report
8	11th July	- Examine the bottlenecks of the current implementation - Search and implement possible optimization techniques
9	18th July	- Refine source code documentation
10	25th July	- Add visual tests - Report performance differences between current and latest version of the rendering engine - Add a sample mod with custom effects (showing the capabilities of the API) - Implement profiling step for finding the fastest available DAG configuration without breaking the given procedure.
11	1st August	If everything goes as planned until this date, without any delays: - Study available VR API's - Implement ability to add multiple cameras into DAG-pipeline - Investigate how this can be used to support VR headsets - Make experiments to find best suitable configuration for VR headset.
12	8th August	- Adapt current VR implementation to DAG - Head Tracking - Movement Tracking (HTC Vive)
13	15th August	- Finalize API and architecture with proper documentation

5. About Me

a. Technical Skills

Amount of programming experience (high to low): Java, C/C++, Objective C, Python, Swift, Javascript, Verilog, F#.

Familiar with OpenGL, Blender, Qt

Editors: Intellij, VIM, Clion, Xcode, Pycharm

Environment: Mac OS X, Ubuntu (sometimes)

Research Experience: Artificial Intelligence (Multi-agent Systems, Trust in Distributed Systems, Internet of Things)

b. Open Source

I have previously worked on a primitive cross-platform open source engine project called "Spy-E" (screenshots). It's written in C++ and based on OpenGL and OpenAL. Besides 3D, I implemented various small open source projects:

VerySimpleCPU simulator: VerySimpleCPU is a kind of RISC CPU that is planned for very small electronic devices. I implemented a user friendly simulator for testing VerySimpleCPU code with Javascript. It can be found in here.

Socivy: A ride-sharing platform that was intended for my university. We had a huge transportation problem in our university, in order to address that we try to build a platform to match car owners with travelers. We had over 700 users which used our web site, iOS and Android app.

For my other project, please check my blog and my Github page. For my professional experience, please check my resume.

c. Academic Background & Education

I am currently attending my 4th year at Ozyegin University, Istanbul, Turkey. My major is BSc. in Computer Science, currently working on Artificial Intelligence, Machine Learning, Trust in Multi-Agent Systems and Internet of Things.

I started to work on my first research project with my academic supervisor, Assoc. Prof. Murat Sensoy, on trust-based fusion of uncertain information from diverse sources, as an undergraduate research assistant. In

order to address the complexity of the evaluation of trust algorithms, I developed a social sensing framework based on Twitter which helped researchers to evaluate their algorithms on the data that this framework provides much easier, instead of dealing with simulated data for evaluation.

At the end of my sophomore year, I was afforded the opportunity to intern in University of Aberdeen, Scotland under supervision of Prof. Timothy J. Norman. There, I improved the framework even more by adding text analysis module and a visualisation framework.

Master level courses that I attended in my university are:

- Semantic Web
- Multicore Programming
- Autonomous Decision Making In Multi-Agent Systems
- Agile Development
- Artificial Intelligence
- Computer Vision

My transcript can be downloaded from here.

d. Summer Plans

I will be primarily spending my summer in Eskişehir, Turkey, but I would love to have the opportunity to travel and meet with collaborators and mentors from Terasology based in Europe. A camp perhaps? I plan to work on this project full time.

6. Additional Notes

Some related links:

- https://github.com/MovingBlocks/Terasology/issues/1741
- http://forum.terasology.org/threads/issue-1741-dag-based-rendering-pipelines.1315/
- https://github.com/MovingBlocks/Terasology/pull/2240#issuecomment-200064959
- https://en.wikipedia.org/wiki/Scene_graph
- https://www.youtube.com/watch?v=nBVYMxyN_FI
- https://developer.apple.com/library/ios/documentation/3DDrawing/Conceptual/OpenGLES_ProgrammingGuide/OpenGLESApplicationDesign/OpenGLESApplicationDesign.html
- http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2012/10/Andersson-Tatarchuk-FrostbiteRenderingArchitecture(GDC07_AMD_Session).pdf