# Aquarium Code Analysis/Overview

brandon1.jones@intel.com

April 2, 2019

## Intent

In order to showcase meaningful results from our performance optimization work, we must start with a workload that represents an efficient and modern rendering model. The purpose of this document is to identify inefficiencies that occur within the Aquarium workload. By identifying and removing these issues, we can develop a workload that effectively showcases our backend optimization work. This document only refers to the Dawn backend. No analysis of the OpenGL backend is planned at this time.

## Aquarium::init()

This is the primary initialization function for Aquarium. It includes parameter parsing, resource creation, and resource upload.

***context->initGeneralResources():*** This function is empty and does nothing.

***setupModelEnumMap():*** This function places the name of each info within g_sceneInfo into an array.

***loadReource():*** Function name is misspelled. This function does the majority of the initialization work in the functions it calls: *loadModels()* and *loadPlacement().* The function also loads resources for the "skybox". The skybox is a 6-sided background the scene is drawn within.

- ***loadModels()****:* This function iterates through each object in g_sceneInfo and calls *loadModel. loadModel* iterates through .json files to find resources to load. The shader for the model is set with *model->setProgram* and then resource initialization and upload is done with *model->init().* Each model creates uniform buffers, textures, input state, bind group layouts, and render pipelines.
  - ! **Performance Issue:** When TextureDawn::loadTexture() is called, all possible mipmaps are generated on the CPU, then transferred to the GPU. This is the primary bottleneck causing a long startup time. **Solution:** Find a more efficient way to generate mipmaps. I believe the best path would be to upload the full size textures to the GPU, then use a compute shader to generate all the mips.
  - ***GenericModelDawn::init():*** This sets up various background resources, such as the globe, arch, ship, rocks etc. These objects are static throughout the workload and have varying number of instances.
    - ! **Redundant Buffers:** *lightFactorUniforms* are constant between all generic models. It's unnecessary to make more than one copy. **Solution:** Create a uniform buffer that's shared between all generic models.
  - ***FishModelDawn::init():*** This sets up models for each of the various fish types.
    - ! **Performance Issue:** When *fishPersBuffer* is created, it is hardcoded to create an entry for 100,000 fish. This is unnecessarily large. **Solution:** Pass the number of fish per fish model and size the buffer accordingly.

        - ! **Redundant Buffers:** *lightFactorUniforms* are constant between all fish models. It's unnecessary to make more than one copy. **Solution:** Create a uniform buffer that's shared between all fish models.
    - ***InnerModelDawn::init():*** This function sets up the model for the inside of the globe.
    - ***OutsideModelDawn::init():*** This function sets up the model outside the globe, including the skybox.
    - ***SeaweedModelDawn::init():*** This function creates a model for each type of seaweed.
        - ! **Redundant Buffers:** *lightFactorUniforms* are constant between all seaweed models. It's unnecessary to make more than one copy. **Solution:** Create a uniform buffer that's shared between all seaweed models.
- ***loadPlacement():*** This function parses JSON files that contain the world locations for static models and puts them inside a matrix owned by the model.

***calculateFishCount():*** This function calculates the number of fish for each different fish type.

## Aquarium::render()

This is the main render function for Aquarium. A single render pass is created and added to for all components. After all operations have been added to the render pass, it is submitted and the backbuffer is presented.

***updateGlobalUniforms():*** This function calculates common position values in the scene, as well as the FPS. The values are copied into CPU-side buffer *viewUniforms*.

- ! **Performance Issue:** These global *viewUniforms* are later copied into individual buffers for each model and updated on the GPU, even though they do not change between models. **Solution:** Upload these to a single uniform buffer than can be shared by all models once per frame.

***matrix::resetPseudoRandom():*** Sets a random seed for matrixes to use.

***context->preFrame():*** This function gets the next texture from the backbuffer, then creates and begins a render pass for the frame that renders to the backbuffer. This render pass is used throughout the workload and shared between all models.

***drawBackground():*** This function iterates and draws the various background components of the scene using DrawIndexed.

- **GenericModelDawn::draw():** The global render pass is acquired and added to here. Bind groups, vertex & index buffers are set and *DrawIndexed* is used to draw all the background pieces.
    - ! **Performance Issue:** *DrawIndexed* is sometimes called with instances = 1. *DrawIndexed* requires overhead that is not optimal when drawing a single instance in comparison to *Draw*. **Solution:** Add a way to use *Draw* when *instances* is 1.
    - ! **Performance Issue:** *DrawIndexed* is sometimes called with *instances = 0*. Nothing is drawn in this case. **Solution:** Add an early out when *instances* is 0.

***drawFishes():*** The 5 different fish models have a corresponding *fishVertexUniforms*, *viewUniforms* and *fishPersBuffer* updated on the GPU. Each fish model is then drawn using *DrawIndexed*.

- ! **Performance Issue:** *fishVertexUniforms* contains constant data, but it is unnecessarily updated and copied for every fish model for every frame. **Solution:** Initialize this data in a uniform buffer within *FishModelDawn::init()* and do not update the buffer every frame.
- ***FishModel->draw():*** The global render pass is acquired and added to here. Bind groups, vertex & index buffers are set and each fish model is drawn using *DrawIndexed* with a varying count.
    - ! **Performance Issue:** *fishPersBuffer* is updated with a hardcoded length of 100,000, which does not correspond to the actual number of fish rendered. **Solution:** Pass the number of fish for the model as a parameter and use that as the count.

***drawInner():*** *MODELGLOBEINNER* has *viewBuffer* updated and is then drawn.

- **InnerModelDawn::draw():** The global render pass is acquired and added to here. Bind groups, vertex & index buffers are set and *DrawIndexed* is used to draw a single instance.
    - ! **Performance Issue:** The *DrawIndexed* call has hardcoded the number of instances to 1. *DrawIndexed* requires overhead that is not optimal when drawing a single instance in comparison to *Draw*. **Solution:** Revise the function to work with *Draw*.

***drawSeaweed():*** *MODELSEAWEEDA* and *MODELSEAWEEDB* have *viewUniforms* and *timeBuffer* updated on the GPU, then 11 instances for each model are drawn using *DrawIndexed*.

***drawOutside():*** *MODELENVIRONMENTBOX* has *viewBuffer* updated on the GPU and is then drawn.

- **OuterModelDawn::draw():** The global render pass is acquired and added to here. Bind groups, vertex & index buffers are set and *DrawIndexed* is used to draw a single instance.
    - ! **Performance Issue:** The *DrawIndexed* call has hardcoded the number of instances to 1. *DrawIndexed* requires overhead that is not optimal when drawing a single instance in comparison to *Draw*. **Solution:** Revise the function to work with *Draw*.