Software Design and Architecture

Team: Engineering Simulations 4
Contributors: Cameron Paul Crutcher, Robert Orlin Jacobson, Levi Z James, Alexander Yang Rhoads
Class: CS46X

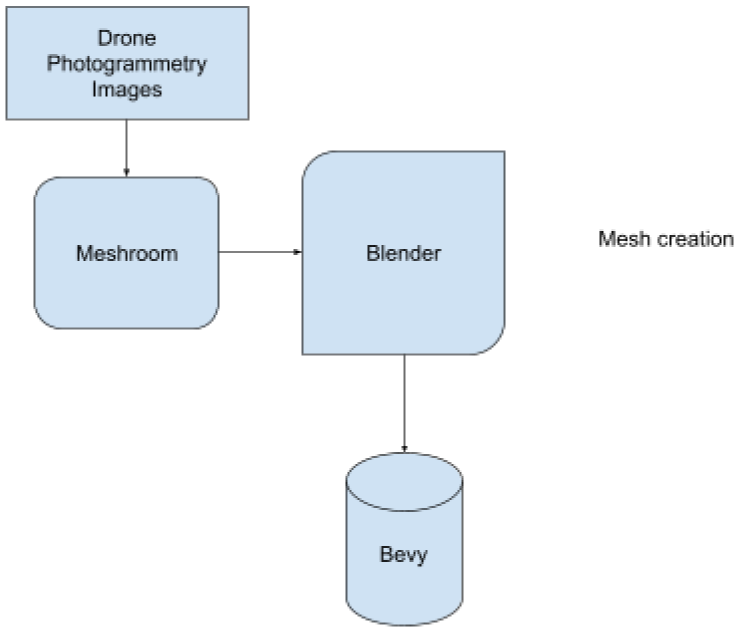
Introduction

This document outlines the software design and architecture for a simple car simulation featuring a robust terrain generation component to supply an interactive environment using the Graphics Library Transmission Format (glTF[3]) in the Bevy game engine. A properly designed implementation ensures a scalable, maintainable, and reliable way to create terrain using photogrammetry[5].

Architectural Goals and Principles

- Ease of integration for any simulation running on the Bevy Game Engine
- Modularity
- Separation of Concerns
- High performance
- Reliability
- Stability
- Maintainability

System Overview



Images captured by a drone are then imported into Meshroom. Meshroom allows for users to stitch these images together and create a 3D mesh[4]. This mesh is imported into Blender for conversion to the glTF file format for use in Bevy. The glTF asset is loaded into the Bevy simulation to generate the simulation terrain. Remaining components handle the car simulation and its physical interactivity with the environment.

Architectural Patterns

A microservices architecture will be used for the project. Each component of the simulation will have a standalone Rust developer file dedicated to it which will define its capabilities within the scope of the project. The photogrammetry pipeline will utilize microservices such as Meshroom and Blender to complete image processing for mesh creation to use in terrain generation.

Component Descriptions

- Meshroom: Meshroom allows a user to import images and create a mesh from the images.
- Blender: Blender is required to clean up the Meshroom mesh, as well as export it in the glTF format.

- Bevy: Bevy is the game engine that the car simulation is run on. It allows for objects and the cameras that provide the user's perspective.
- Rust developer files:
    - Terrain: Loads the generated mesh and textures[6] from the glTF file into the simulation environment.
    - Camera: Creates a camera object to traverse the simulation.
    - Car: Creates a car object to attach the camera and controls for the simulation.
    - Controller: Defines the simulation controls with support for keyboard/mouse and gamepad options.
    - Physics: Extends physics capabilities to the simulation for proper interactions between the car and terrain.

Data Management

Files will be locally stored. Subfolders will be ordered as components of the program, like a subfolder for the car, the physics engine, and the terrain.

Interface Definitions

- Indications of game state.
    - If the simulation is paused.
    - The frames per second.
- Statistics for the vehicle.
    - Speed of car.
    - Force of gravity on the car.
    - Rate of acceleration.
    - Force of collision.
- Meshroom API
- Blender API

Considerations

Security

There are no glaringly obvious security issues, since the program will be entirely offline for the initial versions, but special attention should be given to how the implementation of access to user files works.

Performance

The generated terrain should not drop the frame rate below 30 FPS[2]. Our method should work for any glTF file, not just the ones we use for our sample program. The generated terrain should follow all of the physics implemented into the game by the simulation.

Maintenance and Support

The project partner will be assuming the role of project owner for future implementation by subsequent software development teams. Their skills and platform are unrelated to the current project, however future revisioning of the software may be handled by contracting the work out. Feedback and support channels are slated for in-person meetings as this software may not be deployed outside a local environment.

Deployment Strategy

The stretch goal plans for deployment is through web infrastructure and is therefore not currently planned. Current strategy indicates all work on terrain generation is completed prior to user interaction in a locally deployed environment.

Testing Strategy

Unit tests and feature tests can be handled locally through the Bevy engine and Rust cargo library to generate a local application. Demos will be done remotely for the project partner upon major releases.

Glossary

[1] Collision: The effect of two or more objects within a program being affected when they touch. These effects can take multiple forms, like objects not moving when touching (like a person walking into a wall), or repelling (like bumper cars).

[2] FPS: Frames per second, as in the rate of frames drawn on screen per second. This is essentially directly proportional to the fluidity of an animation. In the realm of 3D games and simulations, this has consequences for how it feels to control something. A higher frame rate corresponds to a larger window of response to input, which improves the feeling of control.

[3] glTF: Graphics Library Transmission Format (A data type)

[4]Mesh: Refers to the 3D model itself, without textures. A collection of polygons to create a larger object.

[5] Photogrammetry: The technique of extracting 3D information from a series of photos, for the sake of making a model directly based
on said photos. The model's textures will be taken from the photos themselves.

[6]Textures: Image files that are used to apply surface detail to 3D models.