Ray Tracing

Matthew Lou (ml4855), Tony Giannini (aug2102), Innokentiy Kaurov (ik2492)

Spring 2026

The core of this project is a hardware accelerator for a simplified ray-tracing renderer. The system as a whole is meant to be a small-scale model of a modern graphics pipeline in which a host machine manages the overall scene and display, while an FPGA performs the expensive per-pixel ray computations in parallel. Instead of rendering entirely in software, the host will send batches of pixel work to the FPGA, the FPGA will compute the results for those pixels, and the host will display the returned image.

The renderer will support a small set of objects: a sphere, a cube, and a pyramid. To keep the design manageable, the scene will contain only one active object at a time, selected using three buttons on the FPGA board. Because the object descriptions are simple, the hardware does not need to store a large model or texture memory; it only needs a small amount of scene data such as object type, object parameters, camera settings, and light position. The design will use fixed-point arithmetic for the ray calculations in order to keep the hardware compact and efficient.

The hardware accelerator will be responsible for computing ray results for many pixels in parallel. The software running on the host machine will be responsible for scene setup, batching pixel requests, sending data to the FPGA, receiving computed results, and displaying the image. The host will also read user-input state from the board so that interaction on the FPGA side affects the rendered output shown on the host machine.

Rendering Goal
The goal is to produce a visible rendered image of a simple 3D object with lighting that responds to user input. The emphasis of the project is not photorealism, but rather demonstrating that the FPGA can accelerate the repeated pixel computations that dominate ray tracing. The first version will focus on a single object, a single movable light source, and simple shading. More advanced visual effects will be considered only if the basic system is working early.

Resolution and Performance
The target will be a modest image resolution that is achievable under the time constraint of the class. The main success criterion is that the FPGA computes batches of pixels fast enough to provide a clear acceleration benefit over a purely software-based version. The displayed image may update progressively as batches return from the FPGA rather than requiring full video-rate rendering. If performance is better than expected, we may increase the resolution or batch size.

Hardware-Software Interface
The host machine and FPGA will communicate through GPIO using a simple batch-based protocol. For each batch, the host will send the information needed to compute a group of pixels, including the current object selection and lighting state. The FPGA will return the computed pixel outputs for that batch. This interface keeps the hardware focused on acceleration rather than full display generation, and it allows the host machine to remain responsible for image assembly and display.

Data Representation
All ray-related calculations on the FPGA will use fixed-point values. This keeps the design simpler and more resource-efficient than a floating-point implementation. Since the project only needs to support a few built-in object types, the amount of data stored on the FPGA will remain small. Rather than loading large geometry files, the system will use compact object parameters for the sphere, cube, and pyramid.

User Interface
A joystick connected to the FPGA will control either the light source position or camera position. The host machine will read this control state from the board and use it when preparing the next rendering batches, so moving the joystick changes the lighting in the displayed image. In addition, three on-board buttons will be used to choose which object is currently rendered: sphere, cube, or pyramid. This gives the project an interactive element while keeping the control logic simple.

System Structure
The project will have three main pieces. First, the host software will manage scene state, package work into batches, communicate with the FPGA, and display the output image. Second, the FPGA hardware will contain the batch-processing logic that performs the ray calculations in parallel. Third, the board-side input logic will capture joystick and button state so that the rendered scene can respond to the user in real time.

Major Tasks

- Define the batch protocol over GPIO, including what data is sent for each request and what results are returned.
- Build a software-only prototype on the host machine that can display the supported objects and manage object and light state.
- Implement the FPGA accelerator for per-pixel ray computation using fixed-point arithmetic.
- Connect the host software to the FPGA so pixel batches can be sent and returned correctly.
- Add joystick input for light movement and button input for object selection.
- Measure performance and compare FPGA-accelerated rendering against a host-only baseline.
- Demonstrate the complete interactive system rendering the three supported objects on the host display.

Expected Outcome

At the end of the project, we expect to have a working interactive renderer in which the host machine displays a ray-traced image while the FPGA performs the heavy pixel computations. The user will be able to switch among a sphere, cube, and pyramid using board buttons and move the light source using a joystick. The final system should clearly show that the FPGA can accelerate this workload by computing many pixel rays in parallel.