Foveated Ray-Tracing

FRANK QIN and ANNY KONG

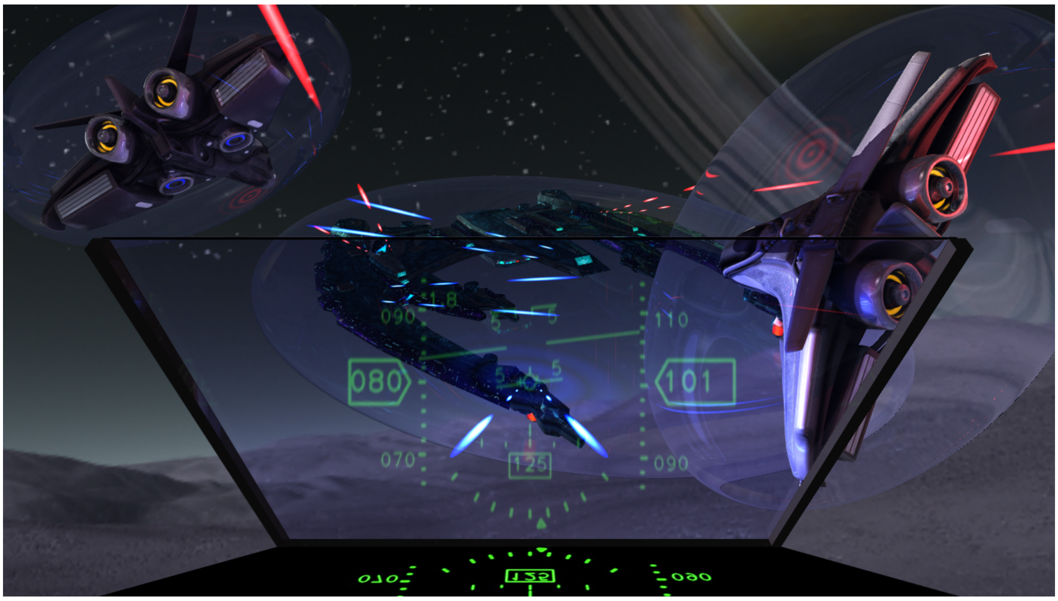

Fig. 1. A ray-traced image rendered by course project from CSE 457 (sp19)

1 ELEVATOR PITCH

We will implement foveated ray-tracing for our homework VR headset. With ray-tracing, the viewer will be able to see more photo-realistic rendering, especially when the scene involves shadows, semi-transparent objects, and reflective objects. The project will be divided into a back-end running on attu server and a front-end running in broswer. The back-end contains the scene and the ray-tracer. The front-end is connected to the headset display and IMUs. The server will perform the CPU intensive ray-tracing and send the rendered image to the client, and the client will display the image on the headset. To increase the performance of the ray-tracer, we will use foveation to reduce the number of rays.

2 EXTENDED OVERVIEW

Our homework VR headset now uses a simple Phong shader to render the scene onto the headset display. The problem with the current rendering process is that it will not render any shadows, reflections, and does not support transparent objects. Although shadow might be implemented using a separate OpenGL rendering pass from the perspective of the light source, reflections and refractions are very hard to implement using OpenGL. By using ray-tracing, we can accurately render shadows, reflections and refractions.

In CSE 457, we have implemented a ray-tracer and can render complex scene (see Figure 1). However, the photo-realistic rendering process is very CPU intensive and required several minutes with 8 threads on my laptop to render one frame. We plan to use two techniques to speed up the

Authors' address: Frank Qin, qzh@cs.washington.edu; Anny Kong, yk57@cs.washington.edu.

rendering process. We will put the ray-tracer on attu, which has 56 CPU cores. We will also use foveation to reduce the number of rays we need for peripheral vision.

In order to render on attu, we need to deploy the ray-tracer and the scene on attu. A laptop connected to headset will communicate with the ray-tracer using websocket. The ray-tracer will set up a websocket server on attu and the laptop will connect to it as a client. The client will perform the foveation calculation and generate a list of rays. The rays will be more concentrated near the center and gradually becomes sparser towards the peripheral. The number of rays can also be controled by the user, so the user can trade resolution for speed. Every time the settings change, the client will send the list of rays to the server. For every frame, the client will collect IMU data to calculate the pose of the viewer. The client will send the pose to the server every frame, and the server will trace each ray with the new viewer pose. The server will then respond with the color of all rays. The client will transform each ray into a vertex on the projection plane with the color returned by the server. The client can then send these vertices to WebGL to interpolate the vertex colors and show them on the display.

2.1 Technical Challenges

This project involves addressing the following key technical challenges.

- We will build on top of the homework projects and replace the WebGL rendering pipeline with ray-tracing.
- We will parallelize the ray-tracer.
- We will construct a system consist of a server and client so that we can take advantage of the large number of CPU cores on attu while also being able to connect to a headset on demo.
- We will implement environment mapping for ray-tracer.
- We will use foveation to reduce the number of rays we need to trace with least loss of quality.
- We will use WebGL geometry transformation to interpolate rays into pixels on the display.

2.2 Key Risks and Mitigations

We identify the following key risks and potential implementation alternatives.

- The speed of ray-tracing may result in low FPS. As a mitigation, we plan to add a UI control for the number of rays so that the user can trade resolution for FPS.
- Attu server we planned to use might be busy as it is shared among all CSE students. An alternative is to use another server from some labs.

3 HARDWARE AND SOFTWARE

This project requires the following hardware.

- [Personal] Laptop: This will be provided by the student team.
- [Personal] attu: This is already accessible by the student team.

This project requires the following software.

- [Personal] VScode: This will be provided by the student team.

4 TEAM RESPONSIBILITIES

This is a 2-person project, with the primary responsibilities being divided as follows.

- **Frank Qin**: Responsible for back-end:
  (1) websocket communication data structures.
  (2) websocket server
  (3) demo scene

(4) ray-tracing
(5) lens correction
(6) environment mapping
(7) (common) final project demo, including an (optional) poster
(8) (common) final project report.

- **Anny Kong**: Responsible for front-end:

(1) websocket client
(2) foveation and ray distribution
(3) UI control for number of rays (resolution)
(4) project rays onto projection plane and connect into triangles
(5) interpolate rays using WebGL
(6) (common) final project demo, including an (optional) poster
(7) (common) final project report.

5 DEVELOPMENT PLAN

We aim to complete this project over three weeks, with the following major milestones.

- **March 3 front-end**: Complete simple ray distribution (uniformly distributed) and triangle connections.
- **March 4 back-end**: Complete environment texture mapping for ray tracing (including lens correction).
- **March 6 front-end**: Complete websocket client.
- **March 6 back-end**: Complete websocket server.
- **March 8 front-end**: Complete interpolating rays using WebGL.
- **March 8 back-end**: Complete scene models and process viewer pose.
- **March 11 front-end**: Complete foveation and ray distribution.
- **March 11 back-end**: Complete multi-threading structure.
- **March 13 front-end**: Complete connecting triangles and interpolation.
- **March 15 back-end**: Complete ray tracing.
- **March 17**: Test render finished object on headset.
- **March 18**: Complete and submit the final report.
- **March 19**: Prepare final project demo, including (optional) poster.
- **March 20**: Participate in the final project demo session.