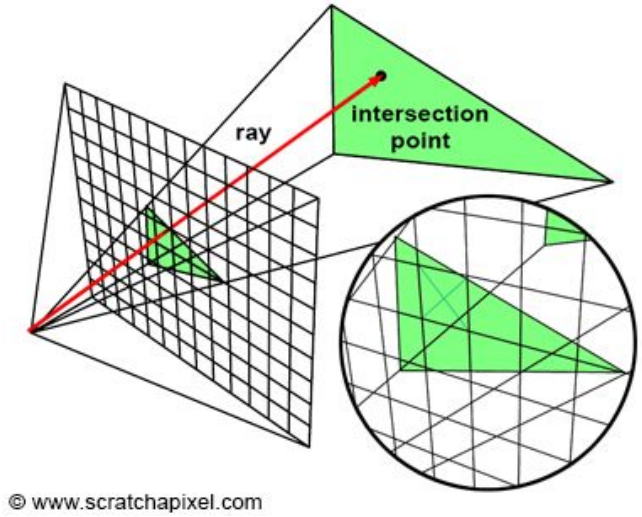
FPGA Ray Tracer

Parker Huntington and Cece Chu

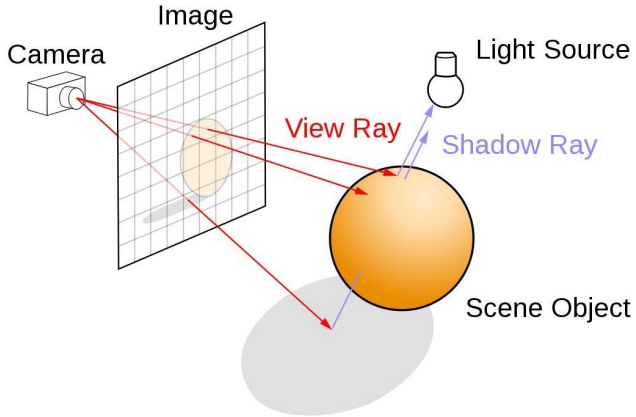
Rendering: Two classes of methods

Rasterization



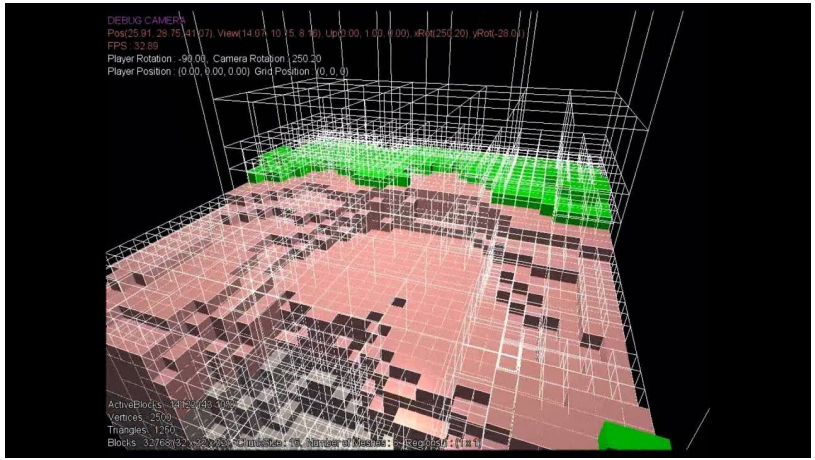
Ray Tracing

- Simulate light
- Reverse the problem -> light comes from camera
- Easier to get photorealism
- More computationally complex



Project Overview

- Take a simplified ray tracer
- Only consider cubes
- Limit the optical effects
- Divide space up using an octree



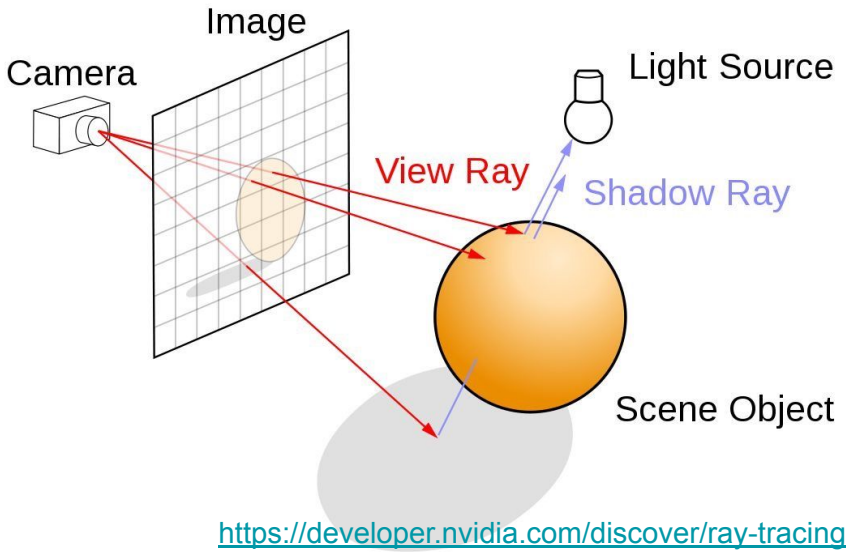
Project Overview

What is Ray Tracing?

- Graphics technique for rendering realistic images
- Shoot rays out from the observer/camera to determine pixel color

Why an FPGA Ray Tracer?

- Many rays in parallel -> hardware Acceleration
- Many opportunities for optimization and expansion

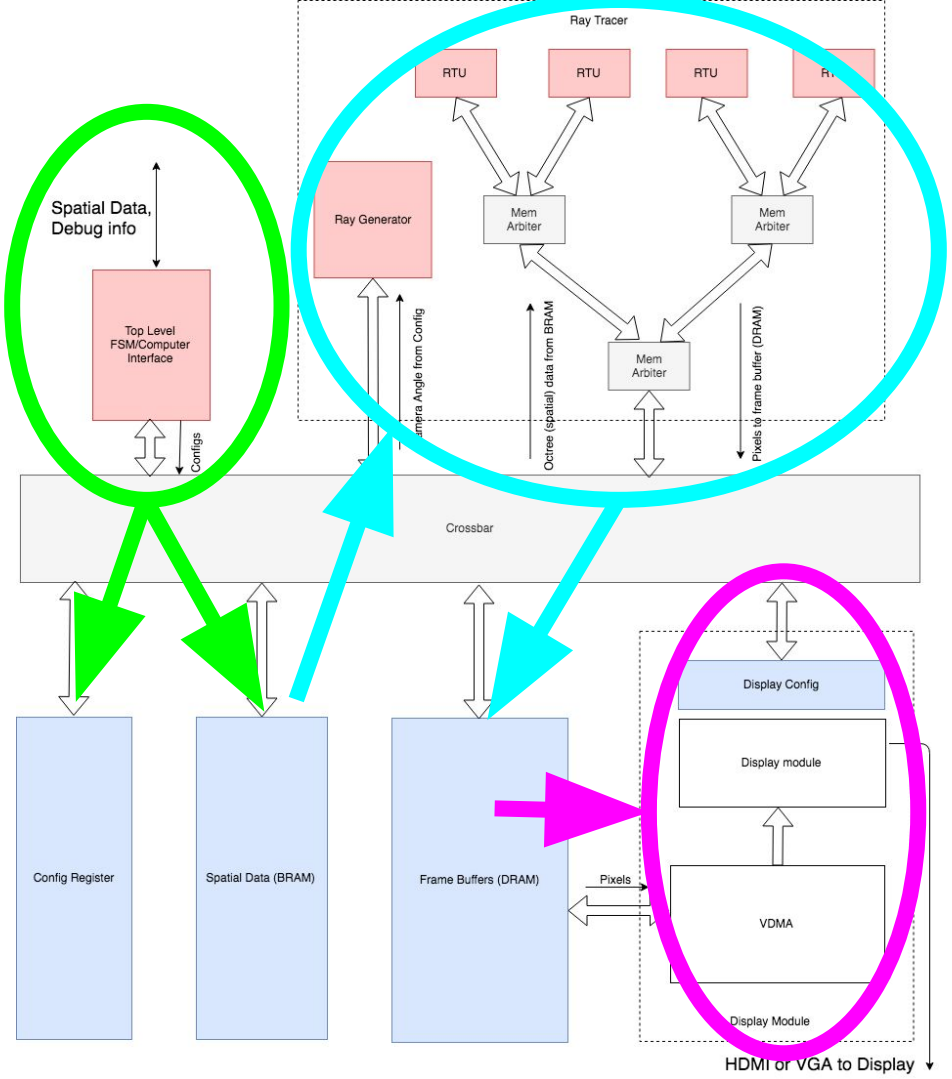


https://developer.nvidia.com/discover/ray-tracing

Block Diagram

Pynq Z2 board

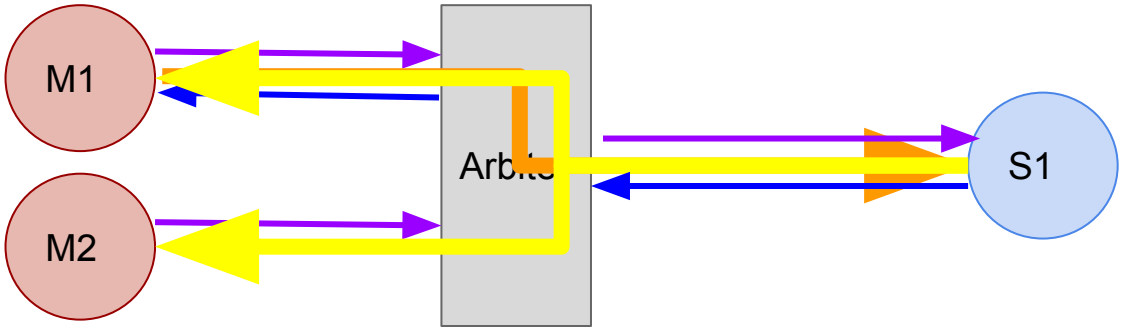
1. Spatial information written from computer interface to BRAM
2. Camera position written from to config port
3. Ray Tracer started, reads spatial data + writes pixels to frame buffer
4. Ray Tracer finished, frame displayed via HDMI



Memory Subsystem (Cece)

Function: Store data and service read/write requests

Protocol:



- Masters (Ray Tracer submodules, computer interface) send read/write requests to slaves (BRAM, DRAM)
- Data transfer when valid signal from master and ready signal from slave = 1
- Requests mediated by arbiters
- For reads, master ID is attached to returned data, all masters on common bus

Memory Subsystem (Cece)

Spatial Information: stored in BRAM

Frame buffer: stored in DRAM, each pixel is 24bit RGB

Config port: registers storing configuration information for ray tracer and display

Crossbar: routes requests to the correct slave depending on address, arbitrates between simultaneous requests

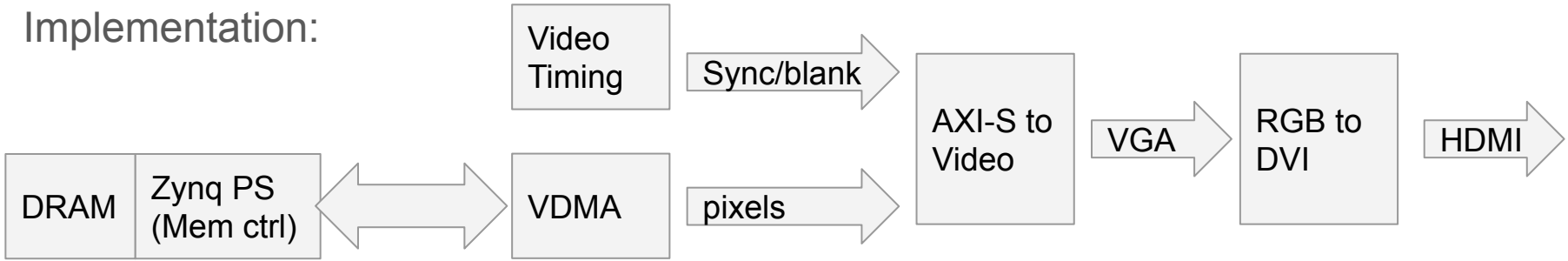
Binary arbiters: Arbitrates requests between two masters (Ray Units)

Challenges: interfacing with AXI, handling master IDs, avoiding bottlenecks

Display Module (Cece)

Function: display pixels from frame buffer on a screen. Goal: 720p HDMI

Implementation:

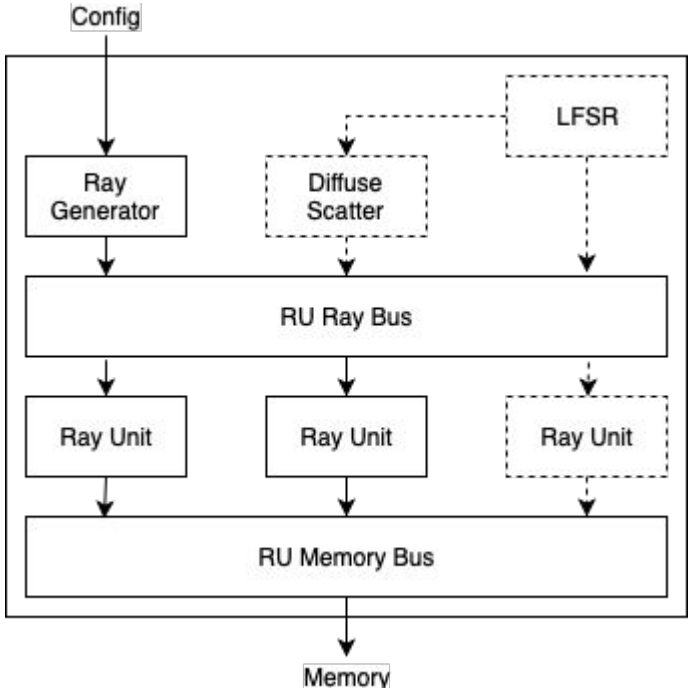


- Stream pixel data from the DRAM frame buffer using a VDMA module
- Add video signals using AXI Stream to Video Out module
- Convert to HDMI output using RGB to DVI module

Challenges: working with AXI, proper configuration/interaction with IPs

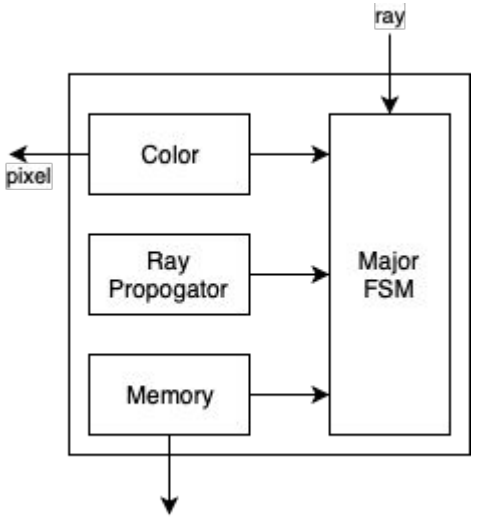
Ray Tracer (Parker)

- Responsible for all ray tracing operations
- Setup up through config
- Generates rays
  - Normalized
- Sends to ray unit
  - Propagates
  - Scatters



Ray Tracer (Parker)

- Color
  - Running color calculation
- Ray Propagator
  - Moves ray out of leaf node
  - Binary search leaf boundaries
- Memory
  - Traverses octree
  - Keeps pointer stack for optimization
  - Gets leaf color info
- Major FSM
  - Coordinates submodules



Computer Interface/Top-level FSM

Function: Transfer external data into system, set camera angle and other configs, control operation of other modules

Implementation:

- AXI module connected to ARM cores
- Bridges PS to custom memory buss

Goals

- Tier 1:
  - Static scene
  - Multiple camera angles
- Tier 2:
  - Diffuse scattering
- Tier 3:
  - Volumetric scattering
- Tier 4:
  - Minecraft

Timeline

Week 1 (Nov 10): Verify/simulate ray tracing algorithm, working display module with frame buffer in DRAM

Week 2 (Nov 17): Single unit ray tracer with working memory hookup and frame generation

Week 3 (Nov 24): Multiple ray units

Week 4 (Dec 1): Implement stretch goals (scattering, animations)

Week 5 (Dec 8): Buffer week/work on report

Week 6 (Dec 11): Finalize report