Make Your Own Ray Tracing GPU with FPGA

COSCUP 2023

Owen Wu
fallingcat@gmail.com

Agenda

- Self-Intro
- Session Overview
- How to Start
  - HDL
  - EDA
  - FPGA
- Think Differently
- Ray Tracing
- HomebrewGPU projrect

Self-Intro

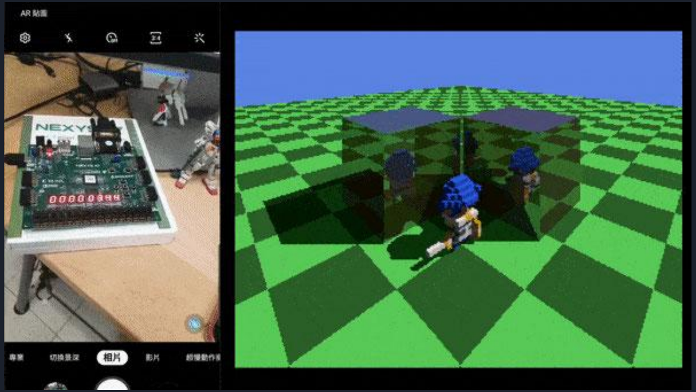
Self-Intro

- Game developer
  - Game engine development
  - PC
  - Console
  - Mobile
- GPU software engineer
  - Optimization from the perspective of hardware
  - AMD
  - Arm
- https://tinyurl.com/owenwu

Session Overview

Session Overview

- This session is for software engineer
- This session is **NOT** for hardware engineer
- Basic intro for the beginner
- Making a chip is very easy and cheap nowaday
    - 6 months from zero to a workable GPU
- Turn you algorithm into hardware, think differently
- Many open sourced projects on GitHub



How to Start

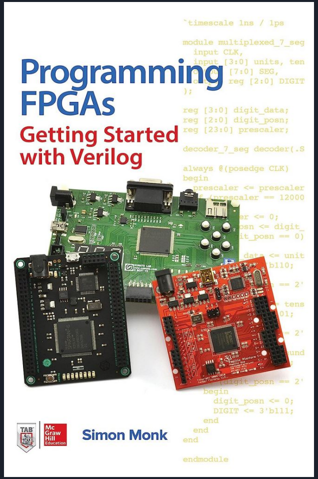
How to Start

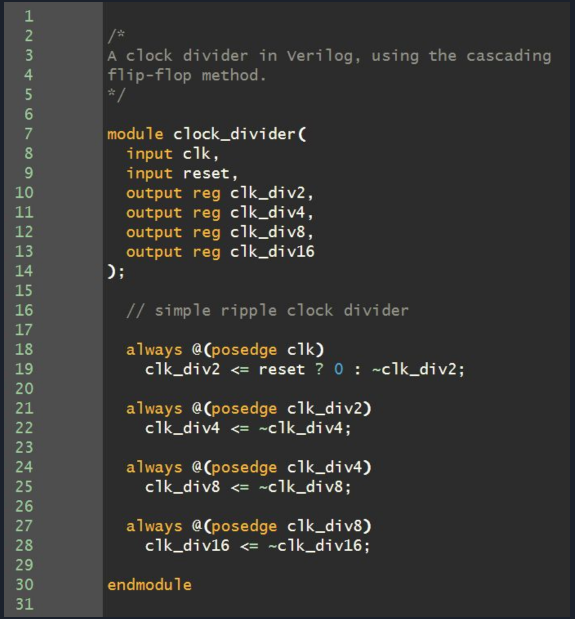
- HDL (Haardware Description Language)
    - Language
- EDA (Electronic Design Automation)
    - Compiler
- FPGA (Field Programmable Gate Array)
    - Hardware

HDL
Hardware Description Language

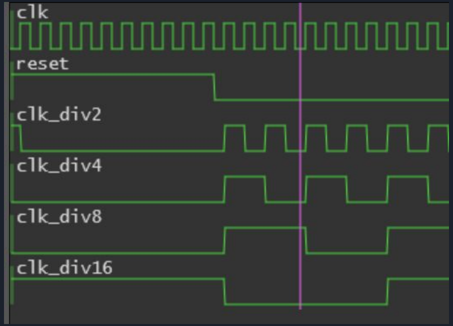
HDL

- VHDL
  - Ada
- Verilog
  - C
- Connect modules to design a whole chip
- Clock is the way to sync all modules
  - 100M Hz - generate 100M signals in one second
- You can think module as a function in C
- Every module can only do very limited works
  - Works need to be finished in one clock
- Books for begineer
  - [Programming FPGAs: Getting Started with Verilog]
  - [Introduction to Verilog]





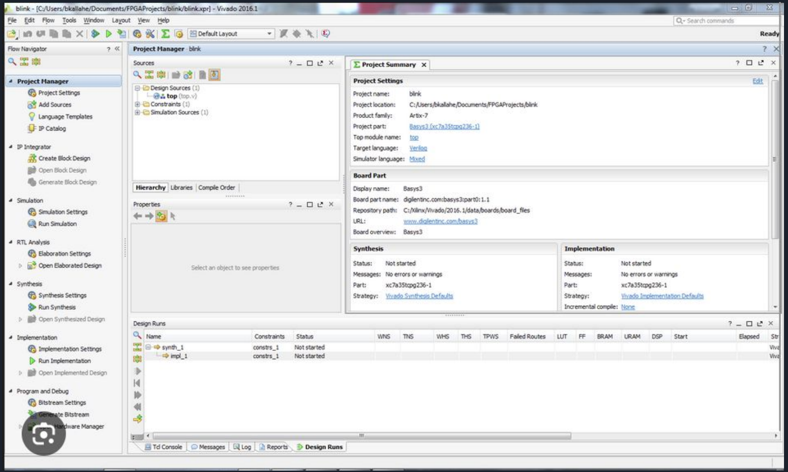
```
/*
A clock divider in Verilog, using the cascading
flip-flop method.
*/

module clock_divider(
  input clk,
  input reset,
  output reg clk_div2,
  output reg clk_div4,
  output reg clk_div8,
  output reg clk_div16
);

  // simple ripple clock divider

  always @(posedge clk)
    clk_div2 <= reset ? 0 : ~clk_div2;

  always @(posedge clk_div2)
    clk_div4 <= ~clk_div4;

  always @(posedge clk_div4)
    clk_div8 <= ~clk_div8;

  always @(posedge clk_div8)
    clk_div16 <= ~clk_div16;

endmodule
```



EDA
Electronic Design Automation

EDA

- Convert HDL to bitstream file
- Upload bitstream file to FPGA to execute
- Many steps
  - IC Design
  - Synthesis
  - Verification
  - Physical Design
- You can think EDA as a compiler
- FPGA makers provide basic EDA for free
  - Xilinx Vivado
- You can also try EDA online
  - https://8bitworkshop.com/v3.10.1/?platform=verilog&file=clock_divider.v



Project Summary

Project Settings

Project name: blink
Project location: C:/Users/bkallahe/Documents/FPGAProjects/blink
Product family: Artix-7
Project part: Basys3 (xc7a35tcpg236-1)
Top module name: top
Target language: Verilog
Simulator language: Mixed

Board Part

Display name: Basys3
Board part name: digilentinc.com:basys3:part0:1.1
Repository path: C:/Xilinx/Vivado/2016.1/data/boards/board_files
URL: www.digilentinc.com/basys3
Board overview: Basys3

Synthesis

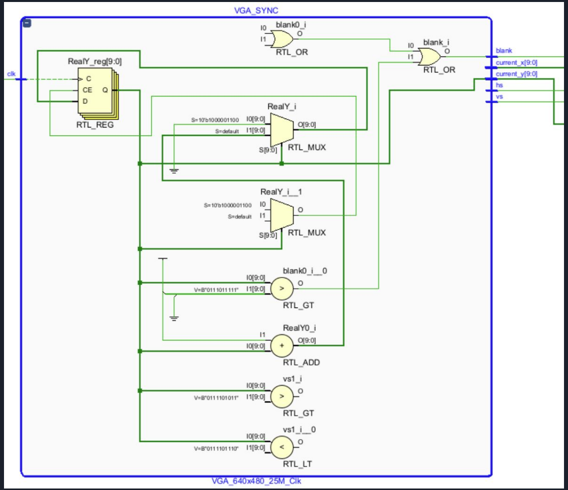
Status: Not started
Messages: No errors or warnings
Part: xc7a35tcpg236-1
Strategy: Vivado Synthesis Defaults

Implementation

Status: Not started
Messages: No errors or warnings
Part: xc7a35tcpg236-1
Strategy: Vivado Implementation Defaults
Incremental compile: None

Design Runs

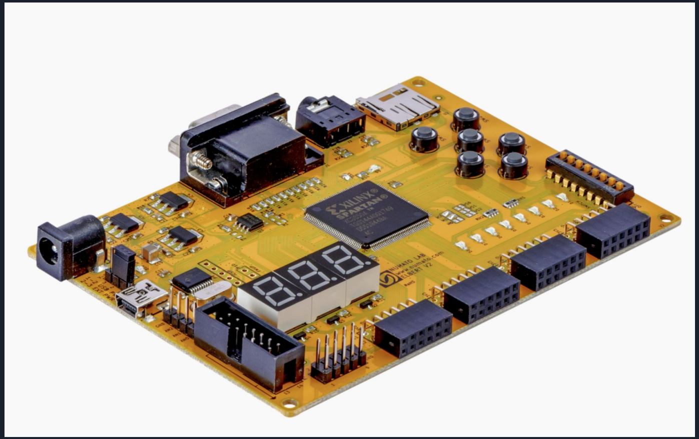
Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed
synth_1	constrs_1	Not started													
impl_1	constrs_1	Not started													

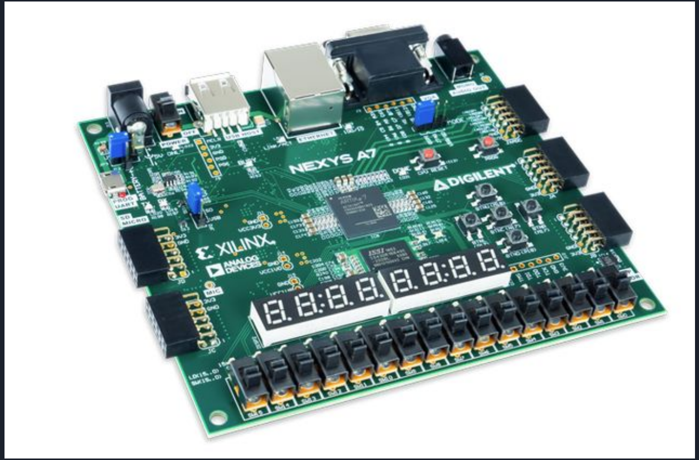


FPGA
Field Programmable Gate Array

FPGA

- Upload bitstream file to configure logic blocks
- FPGA development board integrate many components
  - VGA/HDMI output
  - Memory
  - LED
  - 7 segment disply
  - SD Card
  - SoC
- FPGA has different number of logic cells
  - Which decides how complex the design can be
- Elbert V2 for beginner
- Nexys A7 for more complex design

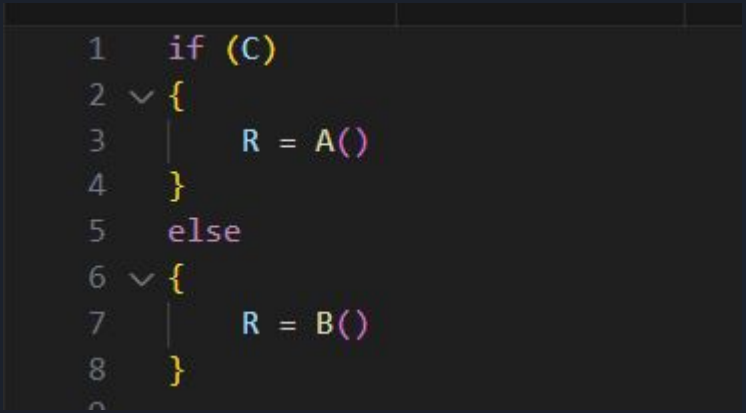




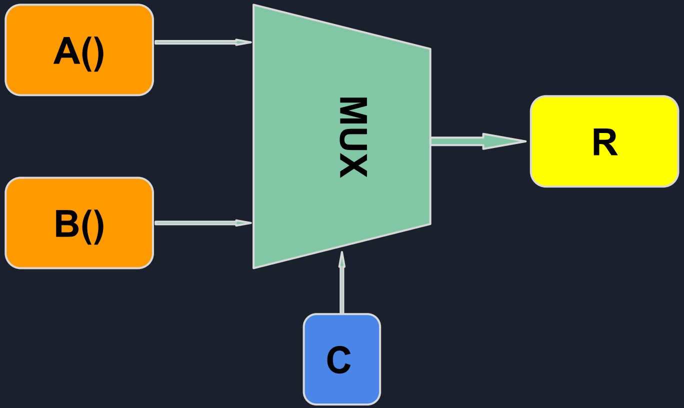
Think Differently

Parallel

- Software is serial
- Hardware is parallel
- Every modules work simultaneously
- Software optimizatios may not work with hardware
- Don’t use software thinking when designing hardware



```
if (C)
{
    R = A()
}
else
{
    R = B()
}
```

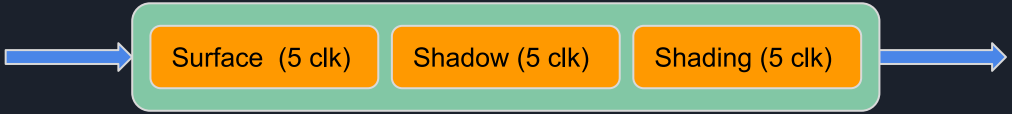


Latency v.s. Throughput

- CPU performance depends on latency
- Low latency means that the instruction can be completed quickly
- Instruction has order dependency
- GPU only care how long it takes to finish a frame
- Pixel doesn’t have order dependency
- Pixels can be executed simultaneously
- If the latency of one pixel is 32 clock
- The hardware executes 64 streams simultaneously
- The throughput will be 2 pixel per clock

Pipeline

- Hardware has many different modules
- All modules need to work simultaneously to get the best performance
- Use pipeline to split the tasks
- Every module process different pixel at the same time



15 clk/pixel

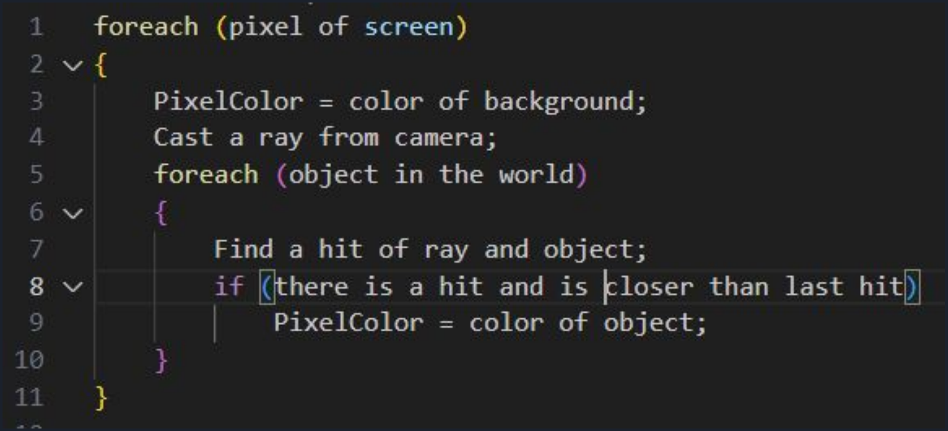
	Surface	Shadow	Shading
clk 0	pixel 1		
clk 5	pixel2	pixel 1	
clk 10	pixel3	pixel2	pixel 1
clk 15	pixel 4	pixel 3	pixel 2

5 clk/pixel

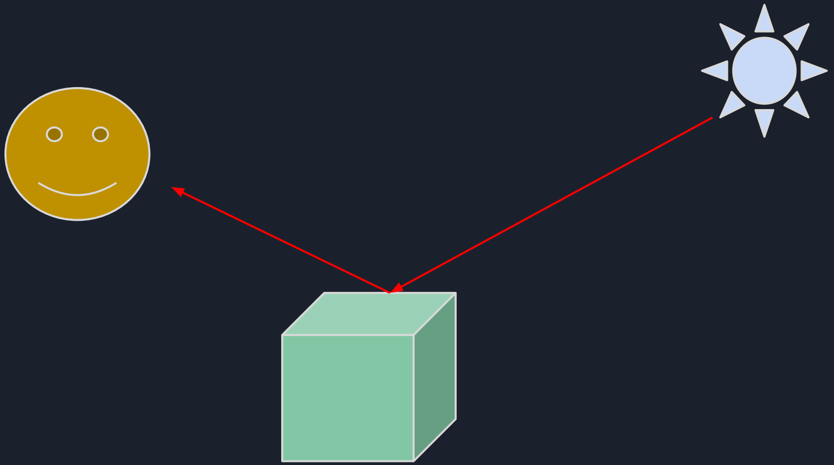
Ray Tracing

Ray Tracing

- Ray tracing is easy to implement
- Ray hit objects then reflect to camera
- Invert the ray
- Cast a ray from each pixel of the screen
- Find a closest hit of ray and objects
- Decide the color of the pixel
    - If there is a hit, the color of hit object
    - If there is no hit, the color of background
- Ray Tracing in One Weekend at GitHub

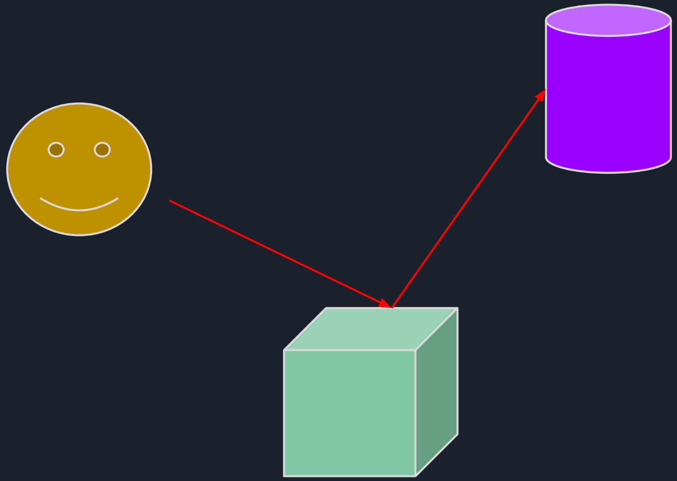


```
foreach (pixel of screen)
{
    PixelColor = color of background;
    Cast a ray from camera;
    foreach (object in the world)
    {
        Find a hit of ray and object;
        if (there is a hit and is closer than last hit)
            PixelColor = color of object;
    }
}
```



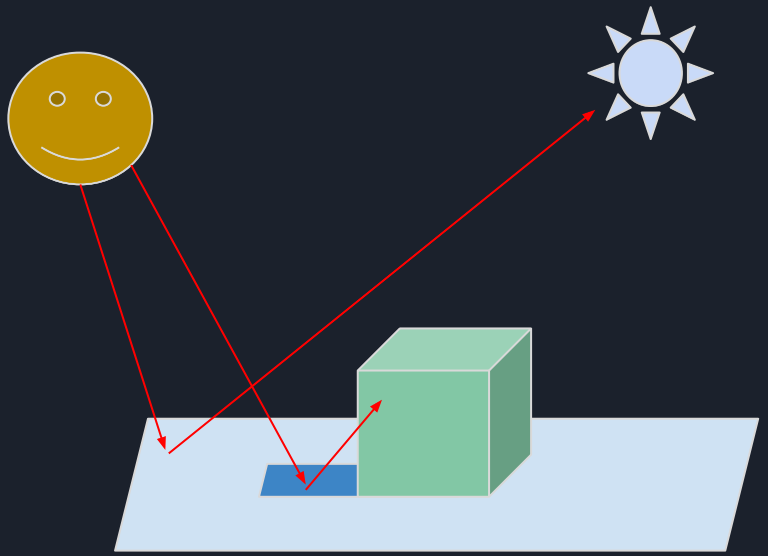
Ray Tracing - Reflection

- For the hit on a object
- If the object is reflective
- Cast a reflection ray from hit point
- Find the closet hit of the reflection ray
- Recursively cast reflection rays
- Blend the colosr of reflected objects into final shading



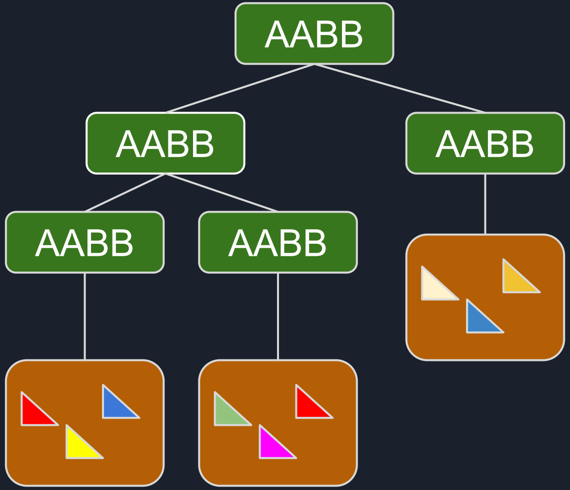
Ray Tracing - Shadow

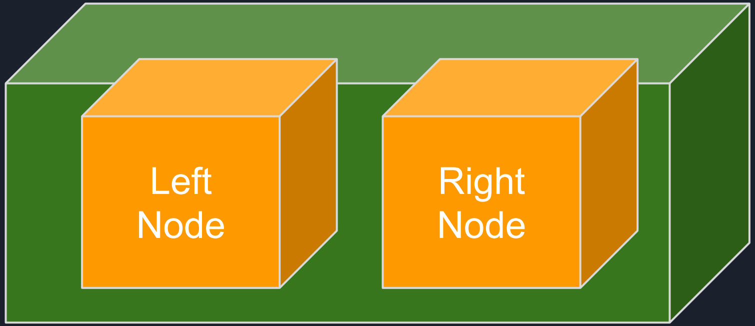
- For the hit on a object
- Cast a new ray from hit point toward light
- Is there is any hit of the new ray
- If yes, the pixel is in shadow
- Otherwise the pixel is not in shadow



BVH(Bounding Volume Hierarchy)

- Accelerate the hit detection between ray and primitives
    - Quickly exclude the nodes which don't have intersection
- Use AABB(Axis-aligned Bounding Box) to split the space
- Traverse the AABB until reach the leaf
- Detect the hits between ray and primitive in leaf

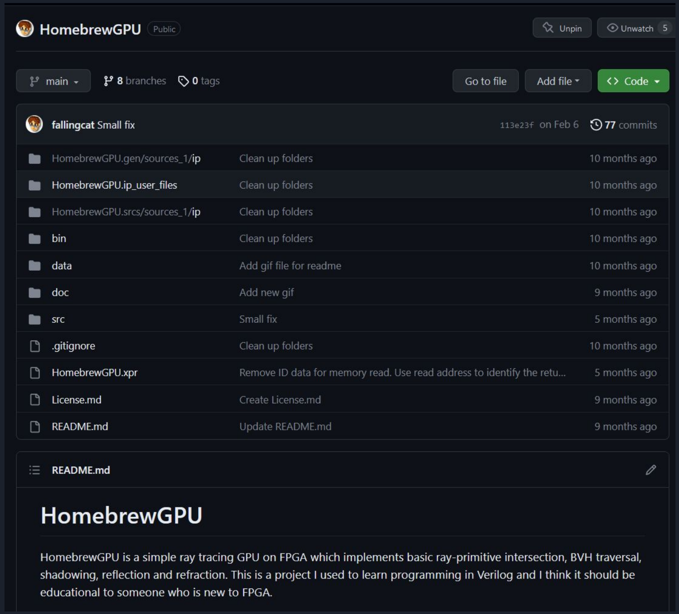




HomebrewGPU Project

HomebrewGPU project

- Open sourced project
  - https://github.com/fallingcat/HomebrewGPU
- Implement a basic ray tracing GPU
  - Voxel based rendering
  - BVH acceleration
  - Shading/Reflection/Refraction/Shadow
- 6 months from zero to complete


HomebrewGPU Public

Unpin | Unwatch 5

main · 8 branches · 0 tags · Go to file · Add file · Code

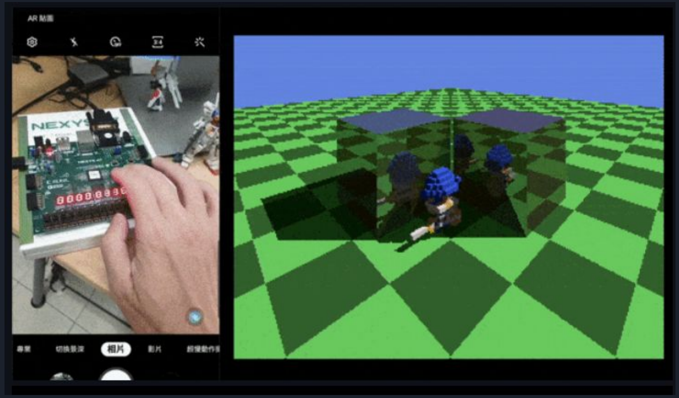
fallingcat Small fix — 113e23f on Feb 6 — 77 commits

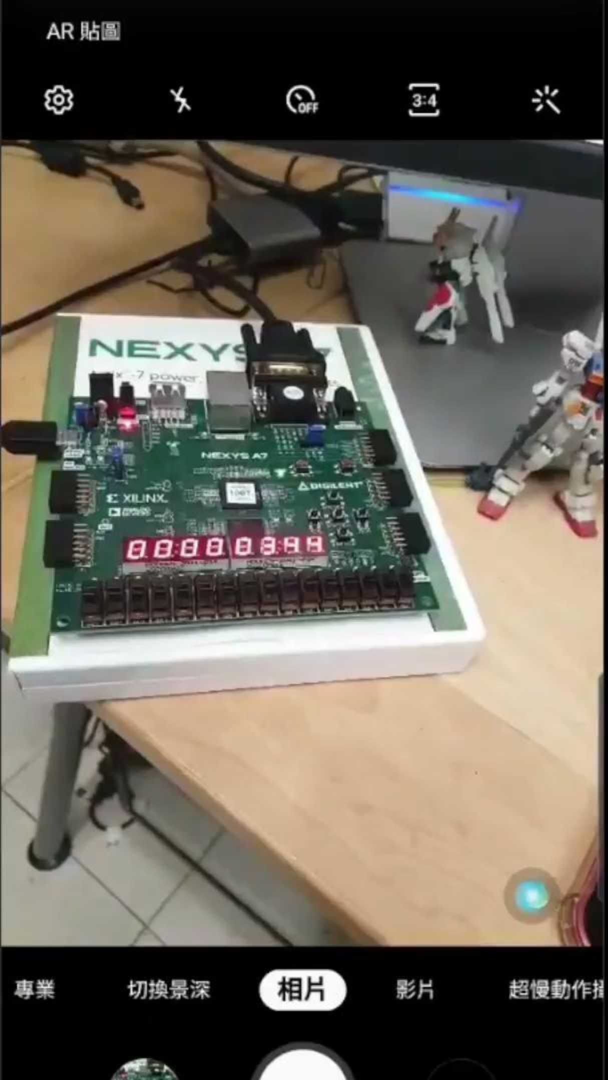
HomebrewGPU.gen/sources_1/ip	Clean up folders	10 months ago
HomebrewGPU.ip_user_files	Clean up folders	10 months ago
HomebrewGPU.srcs/sources_1/ip	Clean up folders	10 months ago
bin	Clean up folders	10 months ago
data	Add gif file for readme	10 months ago
doc	Add new gif	9 months ago
src	Small fix	5 months ago
.gitignore	Clean up folders	10 months ago
HomebrewGPU.xpr	Remove ID data for memory read. Use read address to identify the retu...	5 months ago
License.md	Create License.md	9 months ago
README.md	Update README.md	9 months ago

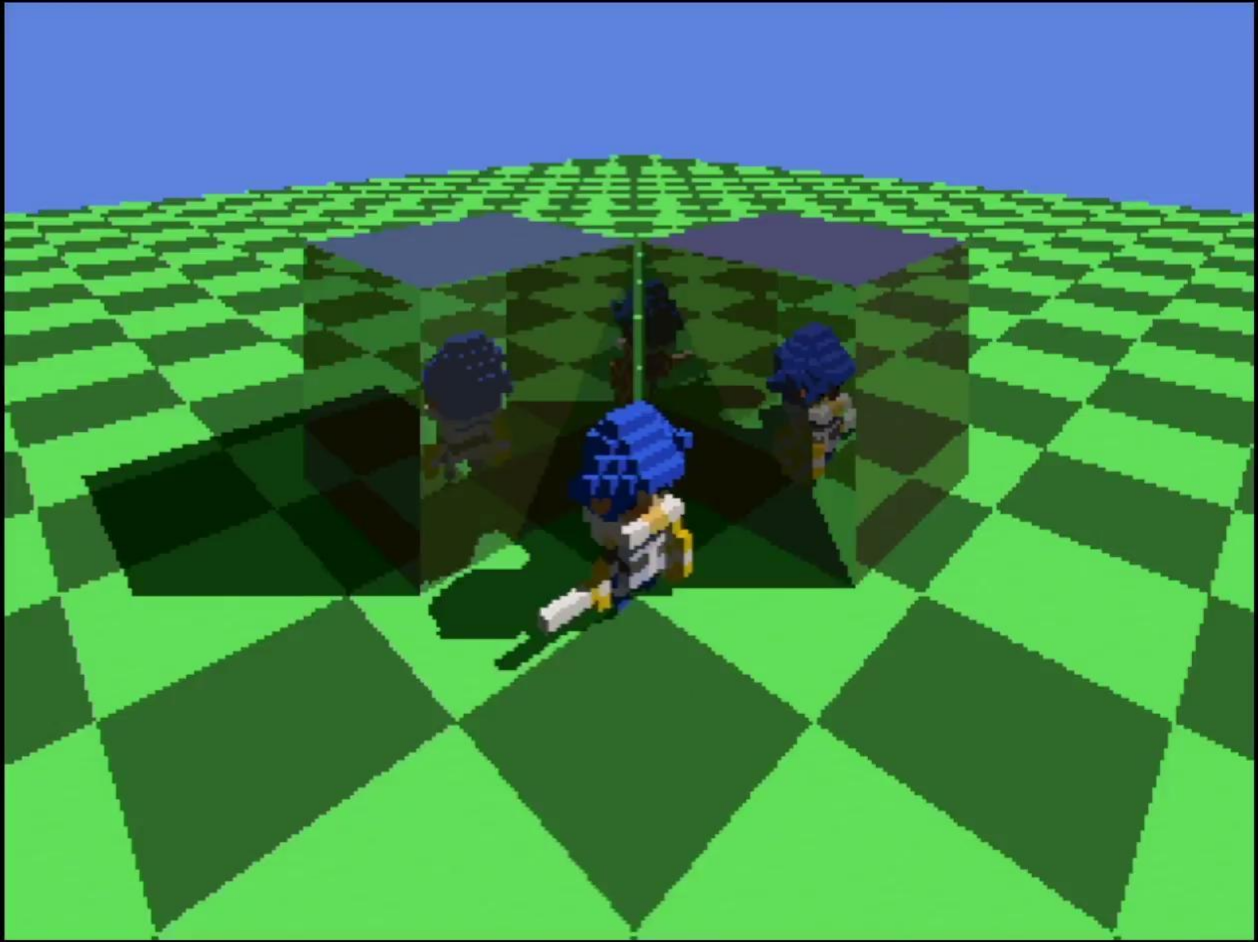
README.md

HomebrewGPU

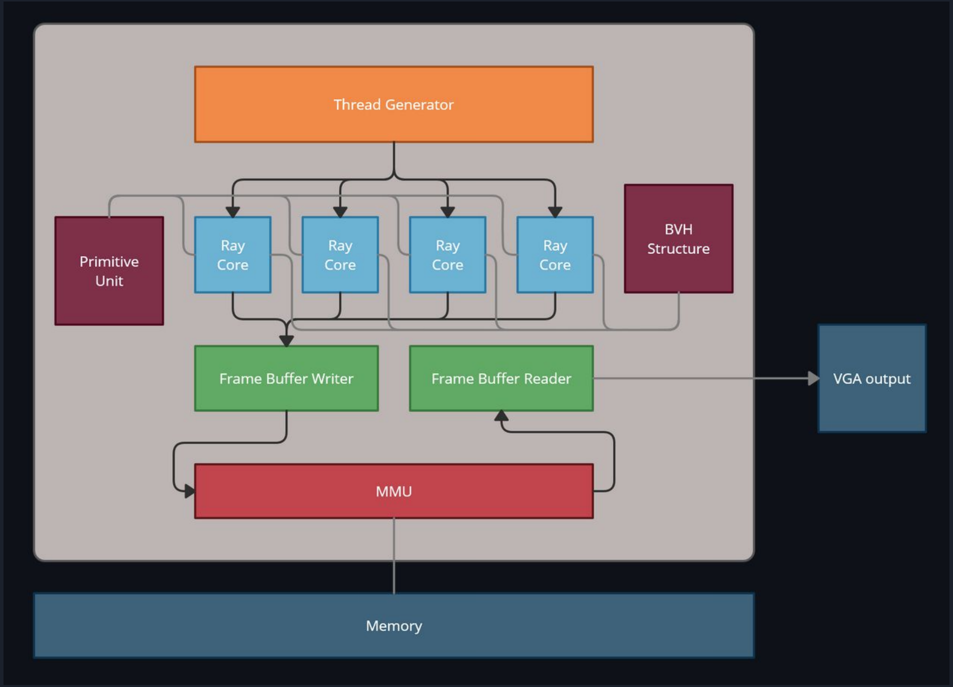
HomebrewGPU is a simple ray tracing GPU on FPGA which implements basic ray-primitive intersection, BVH traversal, shadowing, reflection and refraction. This is a project I used to learn programming in Verilog and I think it should be educational to someone who is new to FPGA.

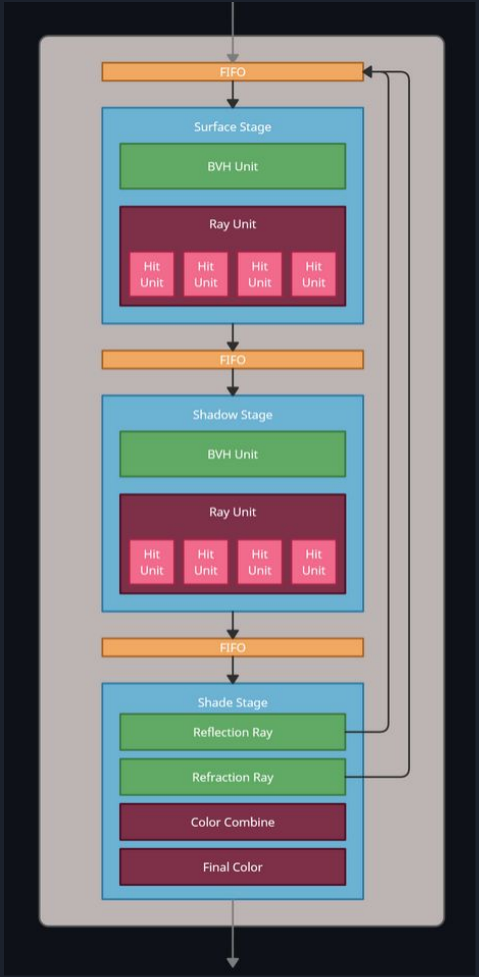



AR 貼圖
NEXYS A7
XILINX
DIGILENT
專業
切換景深
相片
影片
超慢動作攝

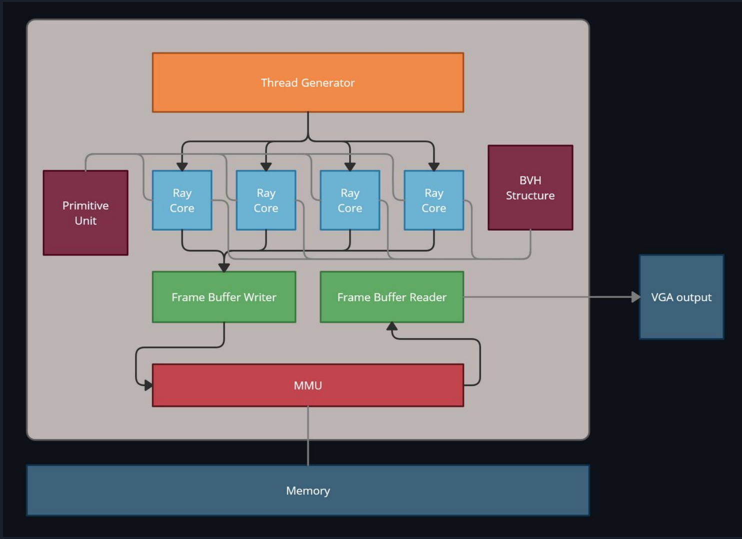


Architecture





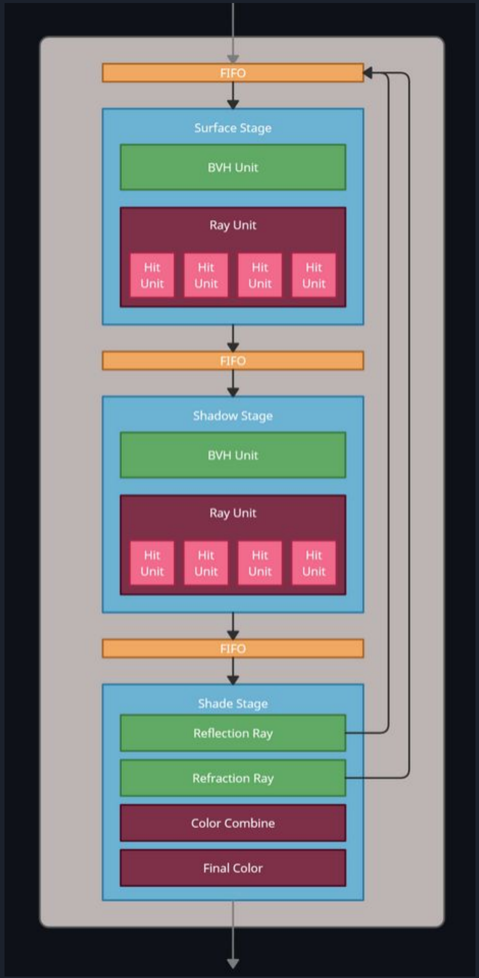
Architecture



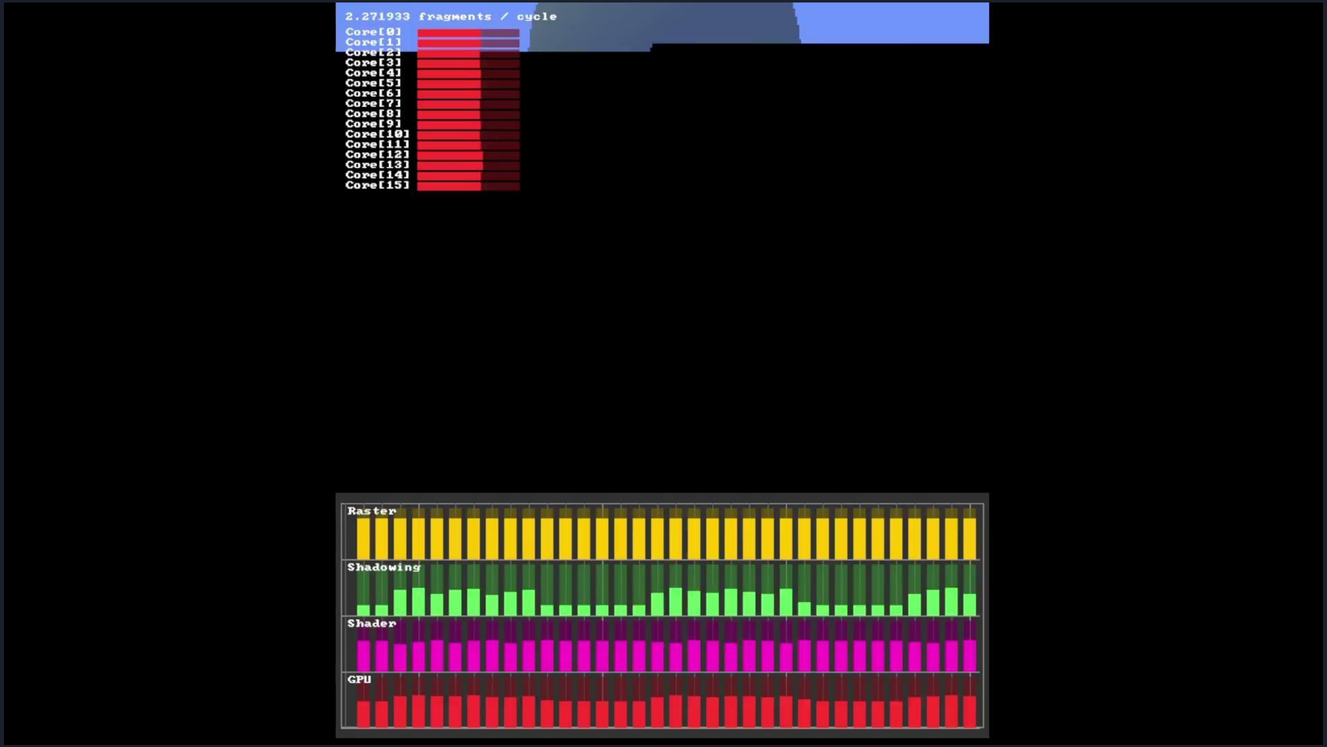
- Thread Generator
  - Generate one thread per clock for each ray core
  - Each thread presents one pixel
  - The thread will go through ray core and output the final color
- BVH Structure
  - BVH structure stores the BVH tree structure data
  - Accepts the node or leaf query from ray core
  - Output the node or leaf data to ray core
- Primitive Unit
  - Primitive Unit stores the raw data of all primitives
  - Accepts the query from ray core and output primitive data
- Ray Core
  - Ray core process one thread to output the final color
  - Accepts the thread from thread generator or reflection/refraction ray
- Frame Buffer Writer
  - Cache the output of ray cores and write the pixel to frame buffer
  - Some threads with reflection/refraction take longer to get the final color
  - Wait util all threads in one cache set are finished then write the data to the frame buffer

Architecture

- Surface stage
  - Process the ray from camera and find the closest hit of the ray
  - Pass the hit information to next stage
- Shadow stage
  - Cast a ray from closest hit position to light source
  - It will pass the shadow information to next stage
- Shade stage
  - Use the closest hit information to decide if it's the final color or reflection/refraction will occur
  - Cast a reflection/refraction ray and pass the data back to surface stage
  - Recursively feed back to surface stage



Design Verification



Q & A

- Owen (fallingcat@gmail.com)
- https://github.com/fallingcat/HomebrewGPU

Thanks!