FPGA Raycating

Team Lightspeed

Adam Carpentieri ac4409
Souryadeep Sen ss6400
Outline

Project Overview
Design
Hardware
Software & Hardware Interface
Contributions & Lessons Learned
Questions & Demo
Project Overview

Why?
Perfect to demonstrate advantages of hardware vs general solution
90’s games are the best!

Goal 1 - Perfect 60hz timing
Goal 2 - Better performance 1 column resolution
Goal 3 - Memory efficient design
Goal 4 - Texturing in hardware

Initial Steps
- Port Java code from tutorial
- Framebuffer and usb code from lab 2 / 3
- Texturing in software
- Collision detection + movement

Problems Identified for Hardware
- Floating Point calc in texturing
- How to store texture data in hardware
- Too many calculations for one cycle
Design
Raycasting Algorithm

- Subset of the ray tracing algorithm, with geometric constraints, making it much faster
  - 64x64x64 cubes
  - Cannot rotate around x or z axis -> walls are always perfectly straight (columns)
- Based on basic high school trigonometry
- Rays are traced backwards, from the players eyes; march rays towards walls
- Raycasting is traced in groups (1 ray per column), whereas ray tracing is per pixel
2D grid to a 3D world...
Some definitions...

Field of View:
- Direction the player is looking at
- Field of view
- Imaginary rays "cast" out from player's eyes.

Field and point of view on the grid:
- 60 degrees FOV
- Point of view
- Field of view

The projection plane:
- Center at (320, 200)
- 320 pixels

The ray cast on the screen (projection plane):
- Column 0
- Column 319
- 200 pixels
How the rays are traced to cast an image
Hardware

- Started with vga_ball as blueprint
- Used Qsys to edit input and output
- Written in verilog
- Column, Texture, and Char modules in addition to supplied VGA module
## Column Data Format

### COLUMN DATA FORMAT

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[41:26]</td>
<td>Top of Wall</td>
</tr>
<tr>
<td>[25:10]</td>
<td>Wall height</td>
</tr>
<tr>
<td>[9]</td>
<td>Wall side</td>
</tr>
<tr>
<td>[8:6]</td>
<td>Texture type</td>
</tr>
<tr>
<td>[6:0]</td>
<td>Texture Offset</td>
</tr>
<tr>
<td>[31:0]</td>
<td>Scaling Factor</td>
</tr>
</tbody>
</table>
Pixel Pipeline

➔ Column decoder and renderer custom vga module
  ◆ Six stage pixel pipeline
  ◆ Texture data storage and retrieval
  ◆ Triple buffering
  ◆ Array storage and retrieval: Double barrel design
➔ Tile based character renderer
  ◆ Two stage pipeline
Pixel Pipeline for Scene Rendering

1. Determine Pixel 3 in advance and retrieve data
2. Wait one cycle
3. Wait second cycle
4. Compute texture pixel col + row to retrieve
5. Wait one cycle
6. Pixel data ready
Timing Frames with VBLANK and Triple Buffering

→ Triple buffer design with two back buffers and one front buffer allows async data transfer
  ◆ After each frame, hardware decides which buffer has data that is
    ● Newest
    ● Complete

→ Query register for VBLANK status for software to time itself to 60hz
  ◆ Otherwise have to rely on sleep functions (amateur hour)
  ◆ We use polling but Interrupt is more fussy but less resource intensive
Tile Based Character Module

- Implemented after scene rendering
  - Outputs just 1 bit on or off
  - Stores all font data as array from lab 2 in registers (readmemh)
  - Can display white text on clear background or vice versa
  - Array storage and retrieval: Double barrel design

- Tile based character renderer
  - Two stage pipeline
The Software
Salient features

→ Raycasting engine ported from a Java implementation to C
→ Multi-threaded environment allowing real time update of player coordinates on screen using both a controller and a keyboard. Jumping enabled too!
→ Level select, pause menu, and end of game logic
→ Infinite game loop incorporating all of the above features in logical flow
→ Timing with vblank using a ioctl read
→ Custom map struct holding metadata and map layout for each level of game
Game Flow/Logic

- Libusb1.0 library used to interface with keyboard/controller
- Pthreads library to run keyboard/controller/main threads
typedef struct {
    int width;
    int height;
    int area;
    char name[20];
    short map[1296]; //make this largest area of any map
} maze_t;
Hardware-Software Interface
Things Worthy of Note

➔ Columns data transfer handled by driver
   ◆ Receives pointer to `columns_t` struct
   ◆ Expanded data width to 16 bits
   ◆ Sends over each column as 5 parts, using bit shifting and logical OR ops

➔ Other things it does:
   ◆ Blackout screen, check VBLANK, send char data, reset columns
What Each Member Worked On & Lessons Learned
Questions?

Let’s get to the demo...