CUDDoom: Raycasting Video Game

Alden Goldstein, Edward Garcia
Minyun Gu, Wei-Hao Yuan, Yiming Xu
Original Proposal

Milestone 1
- Implement raycasting algorithm in software
- Design several mazes

Milestone 2
- Integrate the algorithm with FPGA
- Realize hardware acceleration for the algorithm
- Display the world properly on screen

Milestone 3
- Add audio output to the game
- Complete game features, i.e. player movement, interface

30 FPS Goal, Raycasting in software, SRAM Framebuffer
Actual Implementation

- All milestones complete
- 60 Frames per second
- Hardware raycasting acceleration
- Wall textures, floor textures, sky generation
- Multiple wall heights
- Background music from flash memory
Software Overview

• Keeps track of player position
  – Local copy of world map
  – Polls keyboard and updates player direction /position

• Keeps track of casting rays from player’s FOV
  – Calculates and stores angle measurements
  – Passes individual rays and player position to hardware

• Generates Music
  – Keeps track of sound generation through interrupts
  – Fetches new samples from flash memory
Hardware Overview

• Two main clock domains: Nios components (50 MHz), VGA Components (25 MHz)
• Raycasting acceleration calculates ray extension loop and generates intermediate variables such as wall heights
• Memory buffer for intermediate variables protected by dual clock FIFO
• Separate wall texture, floor texture, and sky components generate pixel calculations on VGA timings
• We cast rays ....obviously!
• Based on perspective, farther walls appear smaller...more precisely, column height = inverse distance
• 2-D map layout, based on a matrix, thus all walls must be square (can be diagonal in more advanced ray casters)
• So, we cast rays to find wall on 2-D map, and used the distance to calculate the perceived column height
How to find walls

DDA

- Modified Bresenham’s
- Covers ALL Walls
- Used in LodeV’s software template
- FAST and never misses a wall
- Seems ideal…right??

Lode Vandevenne, Lode’s Computer Graphics Tutorial,
LoopBack

• Because hops in DDA are quantized, it can be prone to ugly, erratic, errors if not enough precision is used (such as fixed point software)

• A happy medium is to employ a loopback, in which edges are refined after iteration.

• The artifact from missing a wall is much more predictable and less ugly than those of DDA, and rest of the wall is smooth as with DDA. Slower, but more robust = less risky option for our project
Ray FSM

Motivation: Casting Rays is an iterative procedure... can be very slow as mentioned

To get across a 32 X 32 map using 1/32 of a square increments, can be as large as....
32 X 32 X sqrt(2) = 1500 iterations per column

→ 1500 X 640 =

almost 1 million iterations per frame!
Ray FSM (continued)

• Loops in software carry large overhead + serial instructions within loop
• Why not increment at 50 MHz --> need hardware
• Share burden between hardware and software = efficient pipelining
Input Ray Casting Parameters
Input Column Address
Output Column Address
Control Signal
Ready
CLK (50 MHz)

NIOS II Avalon Interface
FIFO Interface
VGA Controller Interface

RAY FSM

Output Ray Casting Parameters
256-bit FIFO data
Vertical Blank Out
WRREQ
WRFULL
Vertical Blank In
READY STATE
--ASSERTS READY SIGNAL HIGH FOR SOFTWARE
--READY = '1'

---FIRST RAY EXTENSION STAGE
---INCREMENT COORDINATE

---REDUCE SIZE OF RAY INCREMENTS

---LOOP BACKWARD
--EDGE REFINE

ADDITIONAL CALCULATIONS

PERFORM INTEGER LONG DIVISION

INITIALIZE DIVISION VALUES

INCREMENT RAYS BY 1 INCREMENT

RETURN TO READY STATE

---COMBINATIONAL PROCESS,
--FINALLY RECEIVES ALL STABLE INPUTS
--EXECUTES DURING DIVISION STATES

WAIT STATE
CHECK IF FIFO IS FULL

CHECK IF LAST COLUMN ON SCREEN

WAIT FOR VERTICAL BLANK SIGNAL

Transition: Rising Edge of Control Signal from Software
Action: Latch Inputs from software

Transition: when wall is hit (ROM output is not zero)
Action: Nothing

Transition: Always
Action: Nothing

Transition: Always
Action: Nothing

Transition: Always
Action: Nothing

Transition: Always
Action: Nothing

Transition: Always
Action: Nothing

Transition: After 32 cycles (32 bit integer division)
Action: Nothing

Transition: Last Column? = false
Action: WRREQ = '1' (to FIFO)

Transition: FIFO is not full
Action: Nothing

Transition: Last Column? = true
Action: Nothing

Transition: V_BLANK = '1' (from VGA)
Action: WRREQ = '1' (to FIFO)
Frame Sync

• Edwards wanted Frame Sync, so naturally we put it in...
• David wanted Anti-Aliasing, so naturally we left it out...
Why Frame Sync was annoying

• Not to big of a deal...just double memories, have software wait on V-blank (via FSM), and toggle memories on V-blank... so why is it hard?

• 224 bits of parameters per column X 640 columns, naturally lends itself to 256 bits of parameters and 1024 addresses...

• But this is too big when we double it!!
Memory

- Two Solutions: Cut bits or cut addresses
- Cut down bits to 192, and splice a 128 bit by 1024 memory, with a 64 bit by 1024 memory. Requires 96 M4K blocks according to Megafunction Wizard.
- Use a full 256 bits, but have a memory with 512 addresses and a memory with 128 addresses. Requires 74 M4K blocks according to Megafunction Wizard.
- We used second scheme, to preserve memory and bits (help image more), however, switching addresses lead to a strange ugly line where switch was. We fix this by using memory addressing scheme on the next page
MEMORY INTERFACE

RDREQ

CLK (25 MHz)

To FIFO

256 bit Data from FIFO

Output Column Address (Same as input, latched)

Vertical Blank Out

Output Ray Casting Parameters

WREN

D Flip-Flop

WRAddress

MEMORY INTERFACE

FIFO Interface

TexGen Interface

VGA Interface

WREN

RDAddress

Column

drawStart

drawMid

drawEnd

Output Ray Casting Parameters

To FIFO

Clk

CLK (25 MHz)

D Flip-Flop
FIFO

Motivation: System crashes, seemed to be result of corrupted M4K memory. M4K was interacting with two clocked domains, read addresses and outputs went to 25 MHz domain, writes came from 50 MHz domain (writes come from FSM)

Solution: Make M4K run on 25 MHz, add FIFO to allow Ray FSM to write to M4K Ram. Column addresses (and blank signal), appended to data, M4K constantly reads from FIFO and writes to address encapsulated within data
VGA Rastering

VGA RASTER

- VGA_CLK
- VGA_HS
- VGA_VS
- VGA_BLANK
- VGA_BLANK_SIG
- VGA_SYNC
- VGA_R
- VGA_G
- VGA_B
- CLK (25 MHz)
- Reset

VGA Display

RGB Generator

Row & Column Iterator

- Draw Start
- Draw Middle
- Draw End
- Texture Color
- Sky Color
- Texture Number
- Side
- Current Row
- Current Column
- Wall Position

Clk

Ctrl
VGA Rastering

Diagram:
- Counter
- VGA Signals Generator
- VGA Color
- RGB Generator
- Row Iterator
- Column Iterator
- Multiplexer
- Reset
- Horizontal
- Vertical
- Current Column
- Current Row
- Wall Position
- Texture Color
- Sky Color
- VGA Output
- vga_blank
Texture Generation

- MEMORY
- VGA RASTER
- TEXTURE ROM
- Wall
- Floor
- Mux
- Output
- Current Row
- Texture Color
- Texture Generation
- Current Column
- Combinational Logic
- Pixel Address
Critical Timing
Sky Generation

Row $\ll 9 + \text{angle} \gg 1$

Block RAM Interface

SRAM Interface

MUX

VGA Interface

Sky data

row information

angle data

Avalon bus

cs
addr
write data
r/w
byte_en
read data
Sky Generation Timing

- There is no clock in SRAM, it is controlled by address
- Maximum Data Delay is 15 ns, whether the clock is 40ns or 20ns
Asynchronous SRAM interface

IS61LV25616

TRUTH TABLE

<table>
<thead>
<tr>
<th>Mode</th>
<th>WE</th>
<th>CE</th>
<th>OE</th>
<th>LB</th>
<th>UB</th>
<th>I/O0-I/O7</th>
<th>I/O8-I/O15</th>
<th>Vcc Current</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not Selected</td>
<td>X</td>
<td>H</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>High-Z</td>
<td>High-Z</td>
<td>Isb1, Isb2</td>
</tr>
<tr>
<td>Output Disabled</td>
<td>H</td>
<td>L</td>
<td>H</td>
<td>X</td>
<td>X</td>
<td>High-Z</td>
<td>High-Z</td>
<td>Icc</td>
</tr>
<tr>
<td>Read</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>L</td>
<td>Dout</td>
<td>High-Z</td>
<td>Icc</td>
</tr>
<tr>
<td>Write</td>
<td>L</td>
<td>L</td>
<td>X</td>
<td>L</td>
<td>H</td>
<td>Din</td>
<td>High-Z</td>
<td>Icc</td>
</tr>
<tr>
<td></td>
<td>L</td>
<td>L</td>
<td>X</td>
<td>L</td>
<td>L</td>
<td>Din</td>
<td>Din</td>
<td>Icc</td>
</tr>
</tbody>
</table>
SDRAM architecture
SDRAM controller diagram

- The SOPC generated controller would transmit data to DRAM according FSM
SDRAM interface

- After SOPC generating SDRAM controller, we integrated the controller according to this diagram.
- We also need to generate PLL for proper operation:
  - To generate a DRAM clock ahead by 3 ns.
Keyboard

NIOS II

KEYBOARD CONTROLLER

PS2 interface

AVALON BUS
PS2 interface

• Keyboard controller from Lab3
• Serial interface with CK, DAT
• Data would be put to register for Nios polling
**Sampled data (8-bit width, 22kHz sampling rate)**

Loaded by Flash programmer in advance
As the sound controller can see:

Writing to sound_controller happens (write&chipselect);
Clear the irq

Data_request comes from wm8731, set the irq and starts waiting for the data

Keeps idle; waiting for data_request

Have set irq but wait to ensure the write really happens

Other issues: Why Flash? How about Buffering? Choose the Sampling frequency
Key Difficulties

• Raycasting Speed
• SRAM Clock Domain
• SDRAM Clock Domain
• M4K Clock Domain
• Audio Interrupts
• Memory Division Screen Glitches
• Debugging with Unreliable Peripheral
Lessons Learned

• Pay attention to Clock Domains (Eddy)
• Hardware Debugging is as valuable as software debugging (Minyun and Wei)
• ModelSim is invaluable (Yiming)
• No Printf’s in interrupts (Minyun)
• Persistance is key; You can accomplish anything if you have the patience to learn it. (Alden)