COLUMBIA UNIVERSITY

3D Graphics Accelerator
CSEE 4840 PROJECT REPORT

Jie Huang (jh4000), Chao Lin (cl3654)
Zixiong Liu (zl2683), Kaige Zhang(kz2325)

CONTENTS

1 INTRODUCTION

3D graphics rendering mainly consists of repetitive and parallelizable operations, making it possible to be accelerated by FPGAs. In this report, we present a simple graphics accelerator implemented on the DE1-SOC board, capable of rendering 3D models with interpolated color.

In our design, a software part will supply information of a 3D scene and a linear transformation which encodes scaling and viewpoint, and the hardware will render the scene onto the screen. In the 3D model, each surface is considered as a collection of triangles that are defined by a set of vertices, each of the vertices associated with a color.

2 DESIGN

2.1 OVERVIEW

Our target 3D model consists of triangles to form the faces. Each triangle has three vertex and each of them has X, Y, Z coordinates and its RGB color. Our system is designed to process the data with the MVP matrix and send it to VGA display.

Our system design consists of two major parts connected with Avalon Memory Mapped Buses to an SDRAM controller. The SDRAM controller is an instantiated IP core and is in charge of the accesses to the SDRAM. One VGA unit is used for reading the framebuffer stored in the SDRAM and putting pixel data onto the VGA display according to the VGA standard. One rasterization unit, which is our main design part, reads all the vertex data from the SDRAM and writes the computed data back. We only use SDRAM as the main memory to store the raw data which in our case is the 3D vertex and RGB color for each triangle. The size of the vertex data buffer in memory is dependent on the number of triangles of our model, and the address of the start of the vertex buffer is configurable at runtime. The other part of memory is used for VGA framebuffer in which each pixel has 64 bits information, with details discussed below.

The software mainly has two parts. One is responsible for preprocssing the actual 3D model and generating fotmatted data for our system. The other runs on the board to compute the MVP matrix, load all the data in SDRAM and send “start rendering” signal to the hardware.

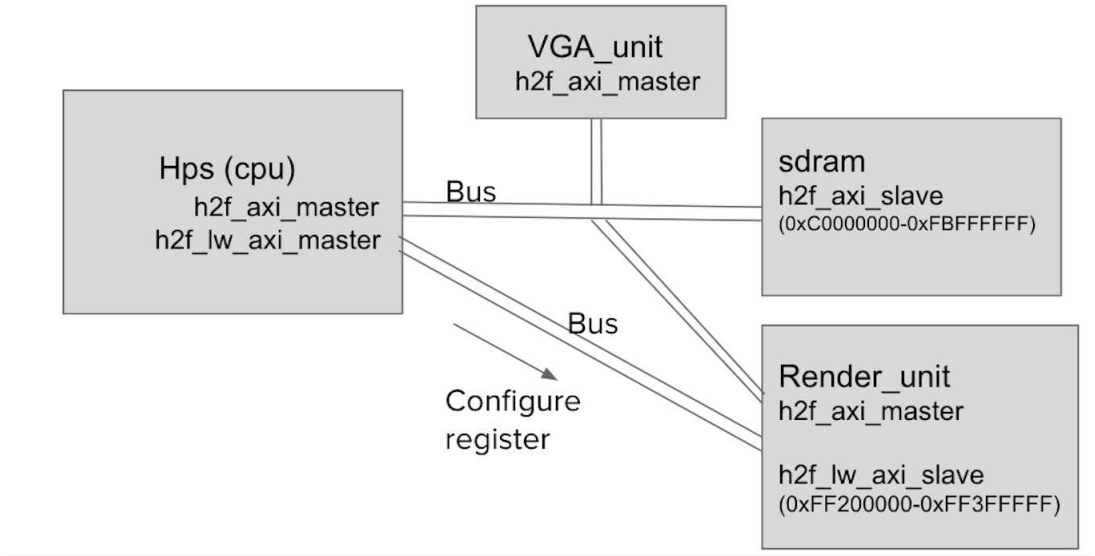

Figure 2.1: System Connection

2.2 Model, View and Projection Matrix Overview

Model Matrix

The model matrix contains every geometric transformation applied to a given object; therefore, it is simply a matrix resulted from multiplying one or more transformation matrices, such as scaling, translation, and rotation, together (Refer to Figure 2.3, 2.4, 2.5).

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R_y = \begin{bmatrix} \cos(\alpha) & 0 & \sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R_z = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 2.2: Rotation in x, y, z directions

$$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 2.3: Scaling and translation

VIEW MATRIX

The view matrix is used to simulate a moving camera view of our 3D objects. Multiplying world coordinates to the view matrix would convert the world space into camera space, which allows us to view the 3D object in various angles and directions.

$$\begin{bmatrix} right_x & up_x & forward_x & position_x \\ right_y & up_y & forward_y & position_y \\ right_z & up_z & forward_z & position_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 2.4: View matrix

PROJECTION MATRIX

There are two types of projection matrices: orthogonal and perspective. For this project, orthogonal projection matrix will be used and is structured as follows, where right, left, top, bottom are positions of the clipping plane.

$$P = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & -\frac{2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 2.5: Projection matrix

2.3 VGA UNIT

The VGA unit consists of a bus master, which reads data from a framebuffer in the SDRAM, and a VGA signal generator, which outputs pixels to the screen with appropriate timing. The framebuffer is an array of 640×480 pixels, with each pixel containing the color information and a Z-coordinate used for Z-buffering. The layout of a pixel is as in figure 2.6.

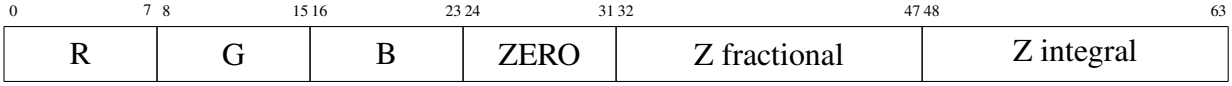

Figure 2.6: Layout of a pixel

The VGA bus master reads pixels from the SDRAM as fast as possible and put them into a FIFO buffer. It stops reading pixels if the FIFO is full. The signal generator asks the bus master

for pixel data at a specific location. If the pixel at that location is in the FIFO, then the FIFO is popped until the desired pixel is on top, and then the pixel data is sent to the signal generator. If the VGA master is too slow in getting the pixels, it would skip a certain number of pixels in order to catch up with the VGA clock. The diagram of the design is in figure 2.7.

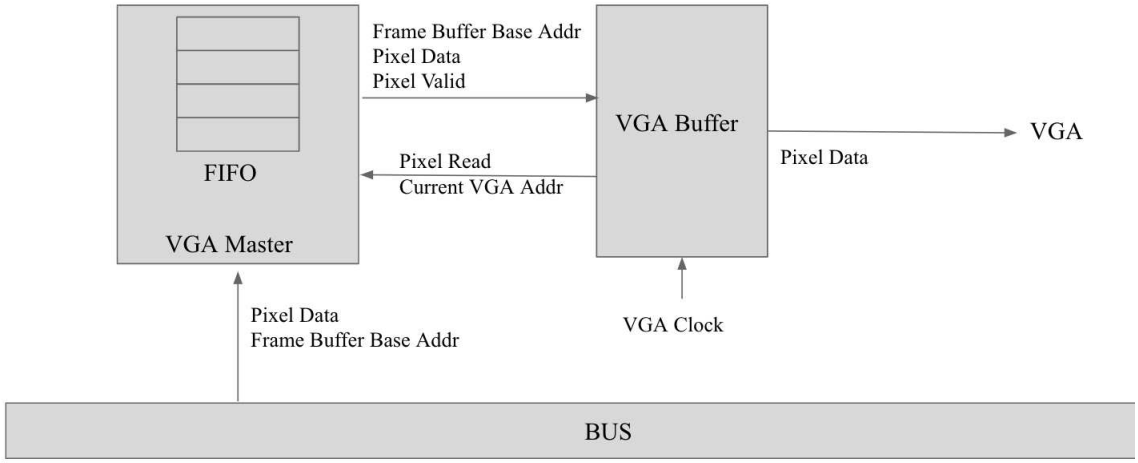

Figure 2.7: VGA Module

2.4 Rasterization Unit

The rasterization unit is composed of five main modules: 1) the configuration registers, 2) the vertex fetcher, 3) the multiplier, 4) the rasterizer and 5) the z-buffering tester. Within this unit, one bus master is connected to the vertex fetcher, while two are connected to the z-test, for the reason that z-test needs to read from and write to the SDRAM separately.

2.4.1 Pipeline Control Protocol

Since we chose to use a pipelined design, we need to define a protocol to make the pipeline stages work together. So for pipeline stage n and $n+1$, we have the following signals: (1) `output_valid` from stage n to stage $n+1$, (2) `stall` from stage $n+1$ to n and (3) `done` from stage n to $n+1$.

Whenever the previous stage has data to send, and is not already sending data, it outputs the data, asserts `output_valid` and wait for at least one cycle. In the next cycle, the next stage acknowledges the data by lowing `stall`, at which time the previous stage knows it can send the next group of data. However, if `stall` stays high, then the pipeline is stalled, and the previous

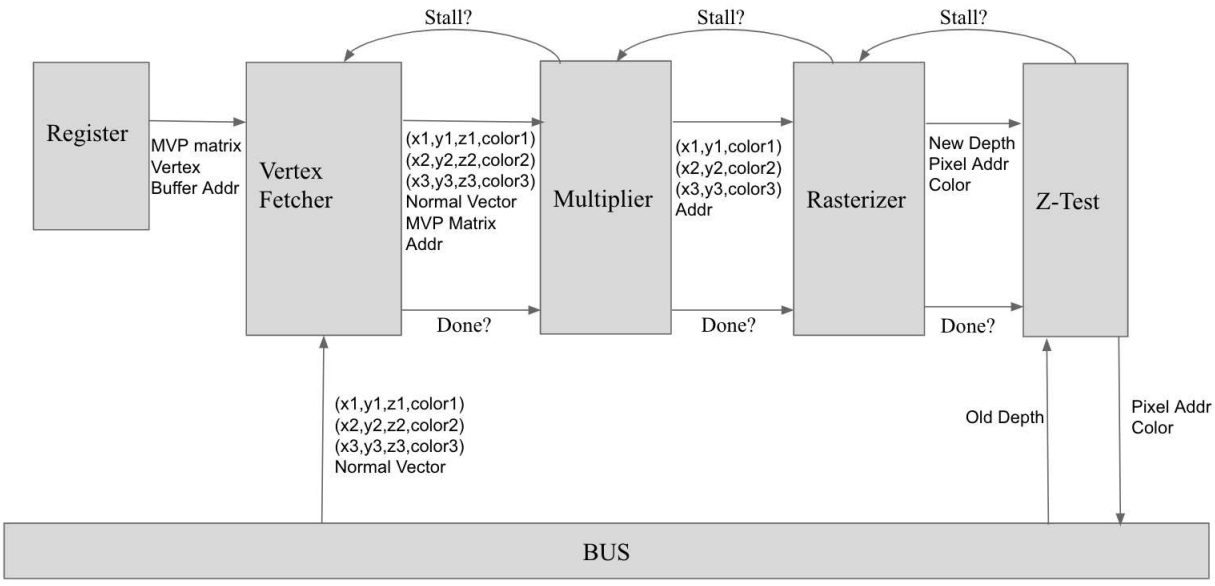

Figure 2.8: Render Unit

stage must retain the last group of data.

When a frame is finished being rendered, the `done` signal is asserted and is propagated stage by stage, and at last it is fed back to the configuration registers, for the CPU to read.

2.4.2 Configuration Registers

The main responsibility of configuration register module is to store the Model-View-Projection matrix generated by software as well as the vertex buffer address into FPGA on-chip registers. The vertex buffer address is the base pointer that points to our 3D model data in the SDRAM.

2.4.3 Vertex Fetcher

The vertex fetcher, containing a bus master, is used to send requests for 3D model data, i.e., the (x,y,z) coordinates and their corresponding colors, to the SDRAM's bus slave. Since we are rasterizing triangles, three vertices are fetched per request. The fetcher sends read request to SDRAM controller with the pipelined bus protocol to improve efficiency. Once the fetcher receives all the data requested, it then transfers the data, including MVP matrix and vertex buffer address obtained from registers, to the multiplier. In our implementation, we use a FIFO buffer of size 16 to temporarily store the received data and it will pop the data when multiplier tells it is ready for the next set of data.

2.4.4 MULTIPLIER

After receiving valid data from vertex fetcher, multiplier performs matrix multiplication of (x,y,z) coordinates to the model-view-projection matrix in order to convert the 3D coordinates into 2D space, including depth information. In addition, all arithmetic operations are done using fixed point. The result from matrix multiplication is transferred to the rasterizer unit.

2.4.5 RASTERIZER

Given three vertices and their colors, the rasterizer module rasterizes the triangle and interpolates color for each pixel within the triangle. The algorithm first determines the bounding box of the triangle by computing its minimum and maximum x,y locations. It then loops through the bounding box line by line and uses edge functions defined in Figure 2.10 to determine whether or not current pixel is inside the triangle. The pixel is inside the triangle if all three edge functions return positive value. If a pixel is determined to be inside the triangle, its SDRAM address is computed and sent to the z-test module, along with its interpolated color. Color interpolation, using the barycentric coordinate technique, is done concurrently with the rasterizing algorithm. The main idea of the barycentric coordinate technique is to find weights that satisfy the system of linear equations defined in Figure 2.12, and those weights can be computed using the functions defined in Figure 2.13. Once the weights of current pixel location are computed, we define the color of current pixel as the weighted average of the RGB values from the three vertices.

$$E_{01}(P) = (P.x - V0.x) * (V1.y - V0.y) - (P.y - V0.y) * (V1.x - V0.x),$$
$$E_{12}(P) = (P.x - V1.x) * (V2.y - V1.y) - (P.y - V1.y) * (V2.x - V1.x),$$
$$E_{20}(P) = (P.x - V2.x) * (V0.y - V2.y) - (P.y - V2.y) * (V0.x - V2.x).$$

Figure 2.9: Edge Functions

2.4.6 Z-TESTER

The final stage of rasterizing is doing the Z-test and write the data back to the SDRAM. The Z-tester first sends read requests to the SDRAM to read the Z-coordinate positions in the frame-buffer. Once the old Z value is received, it is compared to the new depth, in order to decide whether to write back the pixel. If the pixel is to be written back, the Z-tester sends a write request to the corresponding address.

2.4.7 FIXED POINT OPERATION

All the operations in our design are done in fixed point, since Floating point operation can't give us the desired speed. The vertex data as well as MVP matrix are converted into 32 bit fixed point

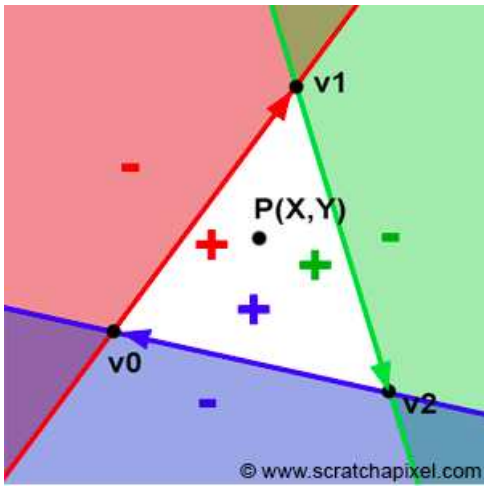

Figure 2.10: Area for rasterization

$$P_x = W_{v1}X_{v1} + W_{v2}X_{v2} + W_{v3}X_{v3}$$
$$P_y = W_{v1}Y_{v1} + W_{v2}Y_{v2} + W_{v3}Y_{v3}$$
$$W_{v1} + W_{v2} + W_{v3} = 1$$

Figure 2.11: Barycentric coordinates

$$W_{v1} = \frac{(Y_{v2} - Y_{v3})(P_x - X_{v3}) + (X_{v3} - X_{v2})(P_y - Y_{v3})}{(Y_{v2} - Y_{v3})(X_{v1} - X_{v3}) + (X_{v3} - X_{v2})(Y_{v1} - Y_{v3})}$$
$$W_{v2} = \frac{(Y_{v3} - Y_{v1})(P_x - X_{v3}) + (X_{v1} - X_{v3})(P_y - Y_{v3})}{(Y_{v2} - Y_{v3})(X_{v1} - X_{v3}) + (X_{v3} - X_{v2})(Y_{v1} - Y_{v3})}$$
$$W_{v3} = 1 - W_{v1} - W_{v2}$$

Figure 2.12: Barycentric weights

value in software before loading to FPGA, The fractional part and integer part are both 16 bits in width. To convert a floating number into fixed point, we first multiply it by 2^{16} and then round it to the nearest integer and cast it to a uint32 type in C++.

3 RESULTS

In the phase one of our project, we implemented the vga unit prototype and tested it with Verilator(a free Verilog HDL simulator). It worked well on verilator and displayed the image we loaded into the memory. A software simulated SDRAM controller and VGA display are also provided for vertilator simulation purpose during this step. However, when it comes to the real synthesizing, it took a while to configure the correct SDRAM working frequency and satisfy timing limitation for our system. Finally, our system is set to work in 100Mhz clock frequency as well as the SDRAM.

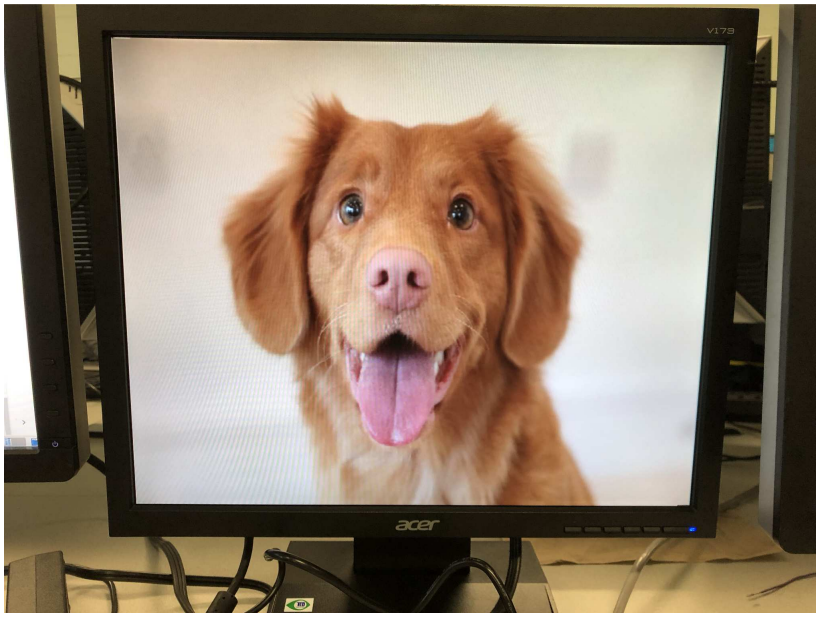

Figure 3.1: VGA Unit Result

In the second phase of our implementation, the same strategy is used. Actually, the decision of pipeline design of our system is made while simulating the rasterization unit. The Cyclone V register map table helps us to find the corresponding mapped memory region to load our data with `busybox devmem`. The timing problems especially stand out when we put the whole thing on the board. Because clock frequency is not an issue in the verilator simulation since every computation could be finished within one cycle with software implementation. We have to put a lot of efforts into simplifying computation for one single cycle and reimplementing pipeline logic. In the end, even though we fixed several major problems related to timing, it still can't meet our expectation.

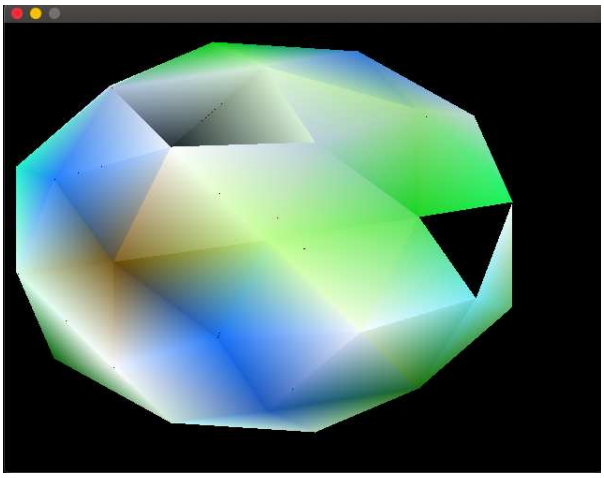

Figure 3.2: verilator simulation

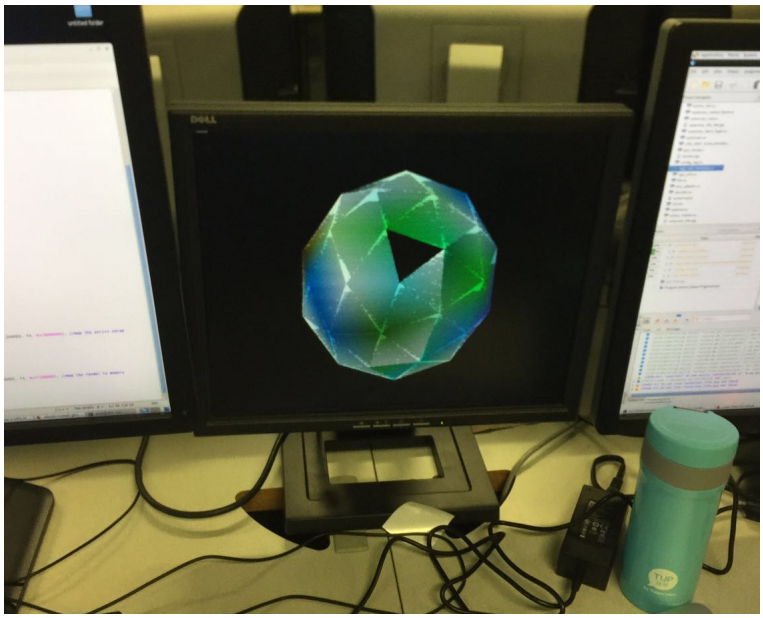

Figure 3.3: final result

Due to time limitation, we only achieved part of what we set out to do. The features we implemented include:

1. Transformation 3D model by a matrix.
2. Rasterize the triangles, though with imperfection in color.
3. Z-buffering to present right occlusion.

We were unable to achieve the following features:

1. Lighting and shadowing, due to the limited number DSP blocks on the board.
2. Smooth animation, due to bandwidth limitation of the SDRAM.

4 Lesson Learned

We learned some lessons from doing this project, which include the following:

1. We should avoid writing along combinational logic. It is advisable to split long arithmetic expressions into simple operations, ideally two operations (multiplication and addition) per cycle.
2. We should have planned more thoughtfully at the beginning, especially about the algorithms used. We chose to use edge functions and barycentric interpolation, which are expensive and made it difficult for the program to pass the static timing checker.

5 Group Members Contribution

All group members contributed significantly to this project.

6 Code Listing

6.1 hardware: VGA_UNIT_MODULES

Listing 1: vga_unit_modules.sv

```
module vga_master (
    input clk,
    input reset,
    input pixel_read,
    input [25:0] base,
    input [25:0] cur_vga_addr,
    input [31:0] bus_data, //data returned by bus
    input master_readdatavalid,
    input master_waitrequest,
    output [25:0] master_address,
    output master_read,
    output [31:0] pixel_data,
    output pixel_valid
);

    reg [25:0] cur_addr;

    reg wr;
    reg rd;
    reg [25:0] din;
    wire [25:0] dout;

    reg [25:0] up_addr;
    reg [25:0] up_addr_next;
    reg [25:0] down_addr;
    reg [25:0] down_addr_next;
    reg addr_invalid;

    logic [31:0] pixel_buffer[32];
    logic [3:0] pixel_rd_ptr;
    logic [3:0] pixel_wr_ptr;
    logic [4:0] pixel_in_progress;
    logic [4:0] pixel_in_progress_next;
    assign rd = master_readdatavalid;

    //fifo
    logic full;
    logic empty;
```

```
logic almost_empty;
logic half_full;
logic almost_full;

fifo #(.DBITS(26)) buffer(
    .clk(clk),
    .reset(reset),
    .wr(wr),
    .rd(rd),
    .din(din),
    .*);

reg sync;
logic [25:0] next_cur, offset8_cur_addr;
assign next_cur = cur_addr - base + 8;
always_comb begin
    if (next_cur > 26'H258000)
        offset8_cur_addr = next_cur - 26'H258000 + base;
    else offset8_cur_addr = next_cur + base;
end

logic [25:0] next_vga, offset8_vga_addr;
assign next_vga = cur_vga_addr - base + 8;
always_comb begin
    if (next_vga > 26'H258000)
        offset8_vga_addr = next_vga - 26'H258000 + base;
    else offset8_vga_addr = next_vga + base;
end

logic [25:0] next_vga128, offset128_vga_addr;
assign next_vga128 = cur_vga_addr - base + 128;
always_comb begin
    if (next_vga128 > 26'H258000)
        offset128_vga_addr = next_vga128 - 26'H258000 + base;
    else offset128_vga_addr = next_vga128 + base;
end

logic [25:0] next_vga120, offset120_vga_addr;
assign next_vga120 = cur_vga_addr - base + 120;
always_comb begin
    if (next_vga120 > 26'H258000)
        offset120_vga_addr = next_vga120 - 26'H258000 + base;
    else offset120_vga_addr = next_vga120 + base;
end
```

```
logic [25:0] next_dout, offset8_dout;
assign next_dout = dout - base + 8;
always_comb begin
    if (next_dout > 26'H258000)
        offset8_dout = next_dout - 26'H258000 + base;
    else offset8_dout = next_dout + base;
end

function logic [25:0] addr_offset_add;
    input logic [25:0] first;
    input logic [25:0] second;

    return ((first - base + second) % 26'H258000) + base;
endfunction

function bit addr_in_range;
    input logic [25:0] addr;
    input logic [25:0] up;
    input logic [25:0] down;

    if (up > down) begin
        return up > addr && addr >= down;
    end
    else if (up < down) begin
        return addr >= down || addr < up;
    end
    else
        return 0;
endfunction

function bit addr_lt;
    input logic [25:0] first;
    input logic [25:0] second;

    if (first < second) return 1;
    else if ((first - base) + 32 * 8 > (second - base) + 26'
       H258000) return 1;
    else return 0;
endfunction

always_ff @(posedge clk or negedge reset)
    begin
        if (!reset) begin
```

```
        cur_addr <= cur_vga_addr + 128;
        pixel_in_progress <= 0;
        pixel_in_progress_next = 0;
        pixel_valid <= 0;
        up_addr <= 0;
        down_addr <= 0;
        addr_invalid <= 1;
        master_read <= 0;
    end
    else begin
        $display("vga_master: ------------------------------");
        $display("vga_master: up_addr = %d, down_addr = %d",
            up_addr, down_addr);
        pixel_in_progress_next = pixel_in_progress;
        up_addr_next = up_addr;
        down_addr_next = down_addr;
        sync = 0;

        if (pixel_in_progress == 16)
            $display("vga_master: pixel buffer full");
        if (master_waitrequest)
            $display("vga_master: sdram asks us to wait");

        if (((master_read && !master_waitrequest) || !
            master_read) && pixel_in_progress < 16) begin
            $display("vga_master: sending request %d",
                offset8_cur_addr);
            wr <= 1;
            din <= offset8_cur_addr;
            master_read <= 1;
            master_address <= offset8_cur_addr;
            cur_addr <= offset8_cur_addr;
            pixel_in_progress_next = pixel_in_progress_next +
                1;
        end
        else begin
            wr <= 0;
            master_address <= cur_addr;
            if (!master_waitrequest)
                master_read <= 0;
        end

        if (pixel_read) begin
```

```
        if (addr_in_range(cur_vga_addr, up_addr, down_addr
            )) begin
            pixel_in_progress_next =
                pixel_in_progress_next - 1;
            pixel_data <= pixel_buffer[(cur_vga_addr / 8)
                % 32];
            $display("write␣pixel␣succeed");
            pixel_valid <= 1;
        end
        else begin
            pixel_valid <= 0;
        end

        $display("vga_master:␣pixel_read␣cur_vga_addr␣=␣%d
            ", cur_vga_addr);

        if (!addr_lt(cur_vga_addr,down_addr))
            down_addr_next = offset8_vga_addr;

        if (!addr_lt(cur_vga_addr, up_addr) && !
            addr_invalid) begin
            sync = 1;
            $display("vga_master:␣syncing!");
            down_addr_next = offset128_vga_addr;
            pixel_in_progress_next = 0;
            up_addr_next = down_addr_next;
            cur_addr <= offset120_vga_addr;
        end

    end

    if (master_readdatavalid && !sync) begin
        $display("vga_master:␣pixel␣data␣at␣%d:␣%d", dout,
            bus_data);
        if (empty)
            $fatal("fifo␣empty");

        if (addr_invalid) begin
            pixel_buffer[(dout / 8) % 32] <= bus_data;
            up_addr_next = offset8_dout;
            down_addr_next = up_addr_next;
            addr_invalid <= 0;
        end else
        if (dout == up_addr) begin
```

```
                    pixel_buffer[(up_addr / 8) % 32] <= bus_data;
                    up_addr_next = offset8_dout;
                end
            end

            pixel_in_progress <= pixel_in_progress_next;
            up_addr <= up_addr_next;
            down_addr <= down_addr_next;
        end
    end
endmodule



module vga_buffer (
    input logic clk,
    input logic reset,
    input [31:0] pixel_data,
    input logic pixel_valid,
    input [25:0] frame_buffer_ptr,
    output pixel_read,
    output [25:0] cur_vga_addr,
    output logic [7:0] VGA_R, VGA_G, VGA_B,
    output logic       VGA_CLK, VGA_HS, VGA_VS,
    VGA_BLANK_n,
    output logic       VGA_SYNC_n
);

    logic [10:0]    hcount, hcount_t;
    logic [9:0]     vcount, vcount_t;

    logic clk50;
    logic clk100;
    logic clk100_c;
    logic clk50_c;

    wire VGA_CLK_PRE;
    //assign VGA_CLK = VGA_CLK_PRE;
    reg [3:0] vga_clk_high_count;

    always_ff @(posedge clk or negedge reset)
        if (!reset) begin
            clk50_c <= 0;
            clk100_c <= 0;
```

```
    end
    else begin
        clk100_c <= clk100_c + 1;
        clk50_c <= clk50_c + 1;
    end


//assign pixel_read = (clk_counter == 4);
assign clk100       = (clk100_c == 0);
assign clk50        = (clk50_c == 0);
assign cur_vga_addr = frame_buffer_ptr + (hcount_t[10:1] + 640 *
   vcount_t) * 8;

vga_counters counters(.clk50(clk50), .reset(reset),.VGA_CLK(
   VGA_CLK_PRE), .clk100(clk), .clk(clk), .*);

typedef enum { R_OUTPUT, R_WAIT, R_CLOCK, R_IDLE } read_state_t;
read_state_t read_state;

always_ff @(posedge clk or negedge reset)
    if (!reset) begin
        {VGA_R, VGA_G, VGA_B} <= {8'hFF, 8'h0, 8'hFF};
        pixel_read <= 0;
        vga_clk_high_count <= 0;
        read_state <= R_IDLE;
    end else begin
        case (read_state)
            R_IDLE: begin
                if (VGA_CLK_PRE) begin
                    hcount_t <= hcount;
                    vcount_t <= vcount;
                    $display("vga_buffer: hcount = %d", hcount
                       [10:1]);
                    if (hcount[10:1] < 640 && vcount < 480)
                        pixel_read <= 1;
                    read_state <= R_WAIT;
                end
            end

            R_WAIT: begin
                VGA_CLK <= 0;
                read_state <= R_OUTPUT;
                pixel_read <= 0;
            end
```

```
                R_OUTPUT: begin
                    if (pixel_valid)
                        {VGA_B, VGA_G, VGA_R} <= {pixel_data[23:16],
                            pixel_data[15:8], pixel_data[7:0]};
                    //{VGA_B, VGA_G, VGA_R} <= {8'hFF, 8'h0, 8'h0};
                    else begin
                        {VGA_B, VGA_G, VGA_R} <= {8'hFF, 8'hFF, 8'hFF
                            };
                        $display("vga_buffer: no pixel");
                    end
                    //VGA_CLK <= 1;
                    read_state <= R_CLOCK;
                end

                R_CLOCK: begin
                    VGA_CLK <= 1;
                    read_state <= R_IDLE;
                end
            endcase
        end
endmodule



/* the following is based on the source code of Lab3 */
module vga_counters (
    input logic          clk, clk50, reset, clk100,
    output logic [10:0] hcount,  // hcount[10:1] is pixel column
    output logic [9:0]  vcount,  // vcount[9:0] is pixel row
    output logic         VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n,
       VGA_SYNC_n
);

/*
    * 640 X 480 VGA timing for a 50 MHz clock: one pixel every other
       cycle
    *
    * HCOUNT 1599 0              1279       1599 0
    *             _______________              ________
    * ___________|    Video      |____________|  Video
    *
    *
```

```
* |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
*       _______________________      _____________
* |____|       VGA_HS          |____|
*/
// Parameters for hcount
parameter HACTIVE      = 11'd 1280,
    HFRONT_PORCH = 11'd 32,
    HSYNC        = 11'd 192,
    HBACK_PORCH  = 11'd 96,
    HTOTAL       = HACTIVE + HFRONT_PORCH + HSYNC +
    HBACK_PORCH; // 1600

// Parameters for vcount
parameter VACTIVE      = 10'd 480,
    VFRONT_PORCH = 10'd 10,
    VSYNC        = 10'd 2,
    VBACK_PORCH  = 10'd 33,
    VTOTAL       = VACTIVE + VFRONT_PORCH + VSYNC +
    VBACK_PORCH; // 525

logic endOfLine;

always_ff @(posedge clk or negedge reset) begin
    if (!reset)          hcount <= 0;
    else if (clk50 & clk100) begin
        if (endOfLine) hcount <= 0;
        else hcount <= hcount + 11'd 1;
    end
end

assign endOfLine = hcount == HTOTAL - 1;

logic endOfField;

always_ff @(posedge clk or negedge reset)
    if (!reset)          vcount <= 0;
    else if (endOfLine & clk50 & clk100) begin
        if (endOfField)   vcount <= 0;
        else              vcount <= vcount + 10'd 1;
    end

assign endOfField = vcount == VTOTAL - 1;

// Horizontal sync: from 0x520 to 0x5DF (0x57F)
```

```
    // 101 0010 0000 to 101 1101 1111
    assign VGA_HS = !( (hcount[10:8] == 3'b101) &
        !(hcount[7:5] == 3'b111));
    assign VGA_VS = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);

    assign VGA_SYNC_n = 1'b0; // For putting sync on the green signal;
        unused

    // Horizontal active: 0 to 1279     Vertical active: 0 to 479
    // 101 0000 0000  1280          01 1110 0000  480
    // 110 0011 1111  1599          10 0000 1100  524
    assign VGA_BLANK_n = !( hcount[10] & (hcount[9] | hcount[8]) ) &
        !( vcount[9] | (vcount[8:5] == 4'b1111) );

    /* VGA_CLK is 25 MHz
    *             __    __    __
    * clk50    __|  |__|  |__|
    *
    *             _____       __
    * hcount[0]__|     |_____|
    */
    assign VGA_CLK = hcount[0]; // 25 MHz clock: rising edge sensitive

endmodule
```

6.2 HARDWARE: VGA_UNIT

Listing 2: vga_unit.sv

```
module vga_unit (
  input clk,
  input reset,
  input [25:0] frame_buffer_ptr,
  input master_readdatavalid,
  input master_waitrequest,
  output master_read,
  output master_write,
  output [3:0] master_byteenable,
  output [31:0] master_writedata,
  input [31:0] bus_data,
  output [25:0] master_address,
  output logic [7:0] VGA_R, VGA_G, VGA_B,
  output logic      VGA_CLK, VGA_HS, VGA_VS,VGA_BLANK_n,
  output logic      VGA_SYNC_n,
  output test_master_read,
```

```
  output test_slave_read,
  output test_waitrequest,
  output [6:0] read_state
);

  logic pixel_read;
  logic [25:0] cur_vga_addr;
  logic [31:0] pixel_data;
  logic pixel_valid;

  logic [3:0] tmp_state;

  assign master_byteenable = 3'b111;

  vga_master master (.clk(clk),
    .reset(reset),
    .pixel_read(pixel_read),
    .pixel_valid(pixel_valid),
    .bus_data(bus_data),
    .cur_vga_addr(cur_vga_addr),
    .pixel_data(pixel_data),
    .master_readdatavalid(master_readdatavalid),
    .master_waitrequest(master_waitrequest),
    .master_address(master_address),
    .master_read(master_read),
    .base(frame_buffer_ptr));

  vga_buffer buffer (.clk(clk),
    .reset(reset),
    .pixel_data(pixel_data),
    .pixel_valid(pixel_valid),
    .frame_buffer_ptr(frame_buffer_ptr),
    .pixel_read(pixel_read),
    .cur_vga_addr(cur_vga_addr),
    .*);

endmodule
```

6.3 HARDWARE: CONFIGURATION_REGISTER

Listing 3: config_reg.sv

```
module config_reg (
    input logic clk,
    input logic reset_n,
```

```
    input logic [31:0] writedata,
    input logic write,
    input logic read,
    input logic [15:0] address,
    output logic [31:0] readdata,
    output logic [31:0] MV [15:0],
    output logic [31:0] MVP [15:0],
    output logic [31:0] lighting [2:0],
    output logic [25:0] frame_buffer_base,
    output logic [25:0] vertex_buffer_base,
    input logic done_in,
    output logic test,
    output logic start_render
);

    logic done_latch;

    always_latch begin
        if (!reset_n)
            done_latch <= 0;
        else
            if (done_in)
                done_latch <= 1;
    end

    always_ff @(posedge clk or negedge reset_n)begin
        if (!reset_n) begin
            start_render <= 0;
            frame_buffer_base <= 0;
            vertex_buffer_base <= 26'h300000;
            test <= 1;
        end
        else if (write) begin
            case (address)
                'h0: begin
                    frame_buffer_base <= writedata;
                    $display("frame_buffer_base: %d", writedata);
                end
                'h4: begin
                    vertex_buffer_base <= writedata;
                    $display("vertex_buffer_base: %d", writedata);
                end
                'h8: begin
                    start_render <= writedata;
```

```
                    test <= 0;
                    $display("start_render: %d", writedata);
                end
                default:
                    if ('h100 <= address && address <= 'h13C)
                        MV[(address - 'h100) / 4] <= writedata;
                else if ('h200 <= address && address <= 'h23C)
                    MVP[(address - 'h200) / 4] <= writedata;
                else if ('h300 <= address & address <= 'h308)
                    lighting[(address - 'h300) / 4] <= writedata;
            endcase
        end
        else if (read && address == 'h8)
            readdata <= done_latch;
        else if (read && address == 'h500) begin
            readdata <= 'hdeadbeef;
            test <= 1;
        end
        else if (start_render == 1)
            start_render <= 0;
    end
endmodule // config_reg
```

6.4 HARDWARE: RASTERIZER_VERTEX_FETCH

Listing 4: rasterizer_vertex_fetch.sv

```
module rasterizer_vertex_fetch (
    input clock,
    input reset,
    output [25:0] master_address,
    output master_read,
    output master_write,
    output [3:0] master_byteenable,
    input [31:0] master_readdata,
    input master_readdatavalid,
    output [31:0]master_writedata,
    input master_waitrequest,
    input fetch_enable,
    input [25:0] vertex_buffer_base,
    /* tasterizer_unit pipeling interface*/
    input stall_in,
    //output stall_out,
    //input done_in,
    output done_out,
```

```
    output output_valid,
    output [31:0] vertex_out[14:0]
);

    parameter FIFO_SIZE = 4;

    typedef enum logic[1:0] {IDLE_S, SEND, TRI_SEND} send_t;
    send_t send_state;

    typedef enum logic[1:0] {IDLE_R, FETCH, TRI_FETCH} rec_t;
    rec_t rec_state;

    logic [31:0] vertex_out_buf[14:0];
    logic [3:0] r_count;
    int output_count;
    logic [25:0] addr;
    int tri_num ;
    int input_count;

    assign master_byteenable = 4'b1111;

    assign done_out = (tri_num != 0) && (ccounter == tri_num);

    logic [3:0] s_count;

    logic [479:0] data_in;
    logic [479:0] data_out;
    logic wrreq;
    logic rdreq;
    logic full;
    logic empty;
    logic almost_full;
    logic half_full;
    logic almost_empty;

    logic fetch_tri;
    logic recv_valid;
    int fifo_size = 2**FIFO_SIZE;
    int fifo_counter;
    logic already_pop;

    fifo #(.DBITS(480), .SIZE(FIFO_SIZE))fifo(
        .clk(clock),
        .reset(reset),
```

```
    .wr(wrreq),
    .rd(rdreq),
    .din(data_in),
    .empty(empty),
    .full(full),
    .almost_full(almost_full),
    .half_full(half_full),
    .almost_empty(almost_empty),
    .dout(data_out)
);

assign vertex_out = '{data_out[479:448],data_out[447:416],
    data_out[415:384],data_out[383:352],data_out[351:320],
    data_out[319:288],data_out[287:256],data_out[255:224],
    data_out[223:192],data_out[191:160],data_out[159:128],
    data_out[127:96],data_out[95:64],data_out[63:32],data_out
       [31:0]};

int ccounter;
//send vertices data from fifo to vertex_cal; receive vertices
   data from bus
always_ff @(posedge clock or negedge reset) begin
    if (!reset) begin
        send_state <= IDLE_S;
        master_read <= 0;
        s_count <= 0;
        fifo_counter = 0;
        already_pop <= 0;
        input_count <= 0;
        fetch_tri <= 0;
        ccounter <= 0;
        output_valid <= 0;
    end
    else begin
        //deal with fifo counter

        case(send_state)
            IDLE_S: begin
                if (input_count < tri_num && fifo_counter <
                   fifo_size) begin
                    $display("vertex_fetch: read addr = %d", addr)
                       ;
                    master_address <= addr;
                    master_read <= 1;
```

```
            addr <= addr + 4;
            input_count <= input_count + 1;
            fifo_counter = fifo_counter + 1;
            s_count <= s_count + 1;
            send_state <= SEND;
        end
        if (tri_num == 0 && fetch_enable && !fetch_tri)
           begin
            $display("vertex_fetch: vertex_buffer_base = %
               d", vertex_buffer_base);
            master_address <= vertex_buffer_base;
            master_read <= 1;
            fetch_tri <= 1;
            addr <= vertex_buffer_base + 4;
            send_state <= TRI_SEND;
        end else if (done_out)
        fetch_tri <= 0;
    end
    SEND: begin
        if (s_count < 15) begin
            if (!master_waitrequest) begin
                master_address <= addr;
                master_read <= 1;
                addr <= addr + 4;
                s_count <= s_count + 1;
            end else begin
                master_address <= addr - 4;
                master_read <= 1;
            end
        end else begin
            if (!master_waitrequest) begin
                master_read <= 0;
                s_count <= 0;
                send_state <= IDLE_S;
            end else begin
                master_address <= addr - 4;
                master_read <= 1;
            end
        end
    end
    TRI_SEND: begin
        if (!master_waitrequest) begin
            $display("vertex_fetch: tri_num read request
               sent");
```

```
                master_read <= 0;
                send_state <= IDLE_S;
            end else begin
                master_address <= addr - 4;
                master_read <= 1;
            end
        end
        default: begin end
    endcase
    if (recv_valid && !full) begin
        data_in <= {vertex_out_buf[14],vertex_out_buf[13],
            vertex_out_buf[12],vertex_out_buf[11],
             vertex_out_buf[10],vertex_out_buf[9],
                vertex_out_buf[8],vertex_out_buf[7],
             vertex_out_buf[6],vertex_out_buf[5],vertex_out_buf
                [4],vertex_out_buf[3],
             vertex_out_buf[2],vertex_out_buf[1],vertex_out_buf
                [0]};
        $display("vertex_fetch: triangle:");
        $display("vertex_fetch: [%x], [%x], [%x]",
            vertex_out_buf[0], vertex_out_buf[1],
            vertex_out_buf[2]);
        $display("vertex_fetch: [%x], [%x], [%x]",
            vertex_out_buf[4], vertex_out_buf[5],
            vertex_out_buf[6]);
        $display("vertex_fetch: [%x], [%x], [%x]",
            vertex_out_buf[8], vertex_out_buf[9],
            vertex_out_buf[10]);
        $display("vertex_fetch: =====");
        wrreq <= 1;
    end else begin
        wrreq <= 0;
        data_in <= 'hff;
    end

    if (!stall_in && !empty && !already_pop) begin
        rdreq <= 1;
        fifo_counter = fifo_counter - 1;
        ccounter <= ccounter + 1;
        $display("ccalc:[%d]",ccounter);
        output_valid <= 1;
        already_pop <= 1;
    end else if (already_pop) begin
        rdreq <= 0;
```

```
            output_valid <= 0;
            already_pop <= 0;
        end else begin
            rdreq <= 0;
            if (!empty)
                output_valid <= 1;
            if (empty)
                output_valid <= 0;
        end
    end
end

always_ff @(posedge clock or negedge reset) begin
    if (!reset) begin
        rec_state <= IDLE_R;
        recv_valid <= 0;
        r_count <= 0;
        tri_num <= 0;//number of total vertices
        output_count <= 0;
    end
    else begin
        case (rec_state)
            IDLE_R: begin//fetch total number
                recv_valid <= 0;
                if (master_readdatavalid && tri_num != 0) begin
                    vertex_out_buf[r_count] <= master_readdata;
                    $display("vertex_fetch: data = %d",
                        master_readdata >> 16);
                    r_count <= r_count + 1;
                    rec_state <= FETCH;
                end

                if (master_readdatavalid && tri_num == 0) begin
                    $display("vertex_fetch: tri_num = %d",
                        master_readdata);
                    tri_num <= master_readdata;
                end
                else if (done_out)
                    tri_num <= 0;
            end
            FETCH: begin//fetch vertices
                if (master_readdatavalid) begin
                    if (r_count == 14) begin
                        r_count <= 0;
```

```
                        vertex_out_buf[r_count] <= master_readdata
                            ;
                        $display("vertex_fetch: data = %d",
                            master_readdata >> 16);
                        recv_valid <= 1;
                        output_count <= output_count + 1;
                        rec_state <= IDLE_R;
                    end else begin
                        vertex_out_buf[r_count] <= master_readdata
                            ;
                        $display("vertex_fetch: data = %d",
                            master_readdata >> 16);
                        r_count <= r_count + 1;
                    end
                end
            end
            default: begin end
        endcase
    end
  end
endmodule
```

6.5 HARDWARE: VERTE_CALC

Listing 5: vertex_calc.sv

```
//refernce: https://stackoverflow.com/questions/27758667/verilog-
    floating-points-multiplication

module vertex_calc (
    input clock,
    input reset,
    input logic [31:0] mat[15:0],
    input logic [31:0] vertex_in [14:0],
    input logic [31:0] lighting[2:0],
    input logic [23:0] color_input1,
    input logic [23:0] color_input2,
    input logic [23:0] color_input3,
    input logic input_data_valid,
    input logic done_in,
    input logic stall_in,
    output reg [31:0] x_out[3:0],
    output reg [31:0] y_out[3:0],
    output reg [31:0] z_out[3:0],
    output reg [31:0] w_out[3:0],
```

```
    output reg [23:0] color_out1,
    output reg [23:0] color_out2,
    output reg [23:0] color_out3,
    output reg out_data_valid,
    output reg done_out,
    output reg stall_out
);

    //v_in 0-14 = x1 y1 z1 rgb1 x2 y2 z2 rgb2 x3 y3 z3 rgb3 nx ny nz

    //fixed point multiplication
    function logic signed [31:0] fp_m(
            input logic signed [31:0] a,
            input logic signed [31:0] b
        );
        fp_m = (64'(a) * 64'(b)) >>> 16;
    endfunction

    logic [31:0] w;

    logic signed [31:0] width;
    logic signed [31:0] height;
    logic signed [31:0] tmp_x[2:0];
    logic signed [31:0] tmp_y[2:0];
    logic signed [31:0] tmp_z[2:0];
    logic signed [31:0] tmp_w[2:0];
    logic [31:0] color1_r;
    logic [31:0] color1_g;
    logic [31:0] color1_b;
    logic [31:0] color2_r;
    logic [31:0] color2_g;
    logic [31:0] color2_b;
    logic [31:0] color3_r;
    logic [31:0] color3_g;
    logic [31:0] color3_b;
    logic [31:0] color1_rnew;
    logic [31:0] color1_gnew;
    logic [31:0] color1_bnew;
    logic [31:0] color2_rnew;
    logic [31:0] color2_gnew;
    logic [31:0] color2_bnew;
    logic [31:0] color3_rnew;
    logic [31:0] color3_gnew;
    logic [31:0] color3_bnew;
```

```
logic [31:0]cosine;
logic [31:0] v_in [14:0];

logic [23:0] color_in1;
logic [23:0] color_in2;
logic [23:0] color_in3;


assign w            = (1 << 16);
assign width[31:16] = 16'd320;
assign width[15:0]  = 0;
assign height[31:16] = 16'd240;
assign height[15:0]  = 0;



//assign cosine = fp_m(v_in[12],lighting[0])+fp_m(v_in[13],
    lighting[1])+fp_m(v_in[14],lighting[2]);

function void output_triangle();
    x_out[0] = fp_m(tmp_x[0], width) + width;
    y_out[0] = fp_m(tmp_y[0], height) + height;
    z_out[0] = fp_m(tmp_z[0], width) + width;
    w_out[0] = fp_m(tmp_w[0], height) + height;

    x_out[1] = fp_m(tmp_x[1], width) + width;
    y_out[1] = fp_m(tmp_y[1], height) + height;
    z_out[1] = fp_m(tmp_z[1], width) + width;
    w_out[1] = fp_m(tmp_w[1], height) + height;

    x_out[2] = fp_m(tmp_x[2], width) + width;
    y_out[2] = fp_m(tmp_y[2], height) + height;
    z_out[2] = fp_m(tmp_z[2], width) + width;
    w_out[2] = fp_m(tmp_w[2], height) + height;

    $display("vertex_calc: triangle = (%d, %d, %d), (%d, %d, %d), (
        %d, %d, %d)",
        $signed(v_in[0]) >>> 16, $signed(v_in[1]) >>> 16, $signed(
            v_in[2]) >>> 16,
        $signed(v_in[4]) >>> 16, $signed(v_in[5]) >>> 16, $signed(
            v_in[6]) >>> 16,
        $signed(v_in[8]) >>> 16, $signed(v_in[9]) >>> 16, $signed(
            v_in[10]) >>> 16);
```

```
    color_out1 = {color1_r[23:16],color1_g[23:16],color1_b
       [23:16]};
    color_out2 = {color2_r[23:16],color2_g[23:16],color2_b
       [23:16]};
    color_out3 = {color3_r[23:16],color3_g[23:16],color3_b
       [23:16]};

    done_out = done_in;
    $display("vertex_calc: color = %d, %d, %d", color_in1,
       color_in2, color_in3);
endfunction

typedef enum logic [1:0] {S_IDLE, S_CALC, S_OUTPUT, S_HOLD}
   state_t;
state_t state;

int calc_counter;
always_ff @(posedge clock or negedge reset) begin
    if (!reset) begin
        done_out = 0;
        stall_out <= 0;
        out_data_valid <= 0;
        calc_counter <= 0;
    end
    else begin
        $display("vertex_calc: state = %d", state);
        $display("vertex_calc: input_data_valid = %d",
           input_data_valid);
        $display("vertex_calc: stall_out = %d", stall_out);
        case (state)
            S_IDLE: begin
                assert(!out_data_valid);
                if (input_data_valid) begin
                    stall_out <= 1;
                    state <= S_CALC;
                    v_in <= vertex_in;
                    color_in1 <= color_input1;
                    color_in2 <= color_input2;
                    color_in3 <= color_input3;
                end
            end

            S_CALC: begin
                calc_counter <= calc_counter + 1;
```

```
case(calc_counter)
    0: begin
        tmp_x[0] <= fp_m(mat[0],v_in[0]);
        tmp_y[0] <= fp_m(mat[4],v_in[0]);
        tmp_z[0] <= fp_m(mat[8],v_in[0]);
        tmp_w[0] <= fp_m(mat[12],v_in[0]);

        tmp_x[1] <= fp_m(mat[0],v_in[4]);
        tmp_y[1] <= fp_m(mat[4],v_in[4]);
        tmp_z[1] <= fp_m(mat[8],v_in[4]);
        tmp_w[1] <= fp_m(mat[12],v_in[4]);

        tmp_x[2] <= fp_m(mat[0],v_in[8]);
        tmp_y[2] <= fp_m(mat[4],v_in[8]);
        tmp_z[2] <= fp_m(mat[8],v_in[8]);
        tmp_w[2] <= fp_m(mat[12],v_in[8]);
    end
    1: begin
        tmp_x[0] <= tmp_x[0] + fp_m(mat[1], v_in
            [1]);
        tmp_y[0] <= tmp_y[0] + fp_m(mat[5], v_in
            [1]);
        tmp_z[0] <= tmp_z[0] + fp_m(mat[9], v_in
            [1]);
        tmp_w[0] <= tmp_w[0] + fp_m(mat[13], v_in
            [1]);

        tmp_x[1] <= tmp_x[1] + fp_m(mat[1], v_in
            [5]);
        tmp_y[1] <= tmp_y[1] + fp_m(mat[5], v_in
            [5]);
        tmp_z[1] <= tmp_z[1] + fp_m(mat[9], v_in
            [5]);
        tmp_w[1] <= tmp_w[1] + fp_m(mat[13], v_in
            [5]);

        tmp_x[2] <= tmp_x[2] + fp_m(mat[1], v_in
            [9]);
        tmp_y[2] <= tmp_y[2] + fp_m(mat[5], v_in
            [9]);
        tmp_z[2] <= tmp_z[2] + fp_m(mat[9], v_in
            [9]);
        tmp_w[2] <= tmp_w[2] + fp_m(mat[13], v_in
            [9]);
```

```
end

2: begin
    tmp_x[0] <= tmp_x[0] + fp_m(mat[2], v_in
        [2]);
    tmp_y[0] <= tmp_y[0] + fp_m(mat[6], v_in
        [2]);
    tmp_z[0] <= tmp_z[0] + fp_m(mat[10], v_in
        [2]);
    tmp_w[0] <= tmp_w[0] + fp_m(mat[14], v_in
        [2]);

    tmp_x[1] <= tmp_x[1] + fp_m(mat[2], v_in
        [6]);
    tmp_y[1] <= tmp_y[1] + fp_m(mat[6], v_in
        [6]);
    tmp_z[1] <= tmp_z[1] + fp_m(mat[10], v_in
        [6]);
    tmp_w[1] <= tmp_w[1] + fp_m(mat[14], v_in
        [6]);

    tmp_x[2] <= tmp_x[2] + fp_m(mat[2], v_in
        [10]);
    tmp_y[2] <= tmp_y[2] + fp_m(mat[6], v_in
        [10]);
    tmp_z[2] <= tmp_z[2] + fp_m(mat[10], v_in
        [10]);
    tmp_w[2] <= tmp_w[2] + fp_m(mat[14], v_in
        [10]);
end
3: begin
    tmp_x[0] <= tmp_x[0] + fp_m(mat[3], w);
    tmp_y[0] <= tmp_y[0] + fp_m(mat[7], w);
    tmp_z[0] <= tmp_z[0] + fp_m(mat[11], w);
    tmp_w[0] <= tmp_w[0] + fp_m(mat[15], w);

    tmp_x[1] <= tmp_x[1] + fp_m(mat[3], w);
    tmp_y[1] <= tmp_y[1] + fp_m(mat[7], w);
    tmp_z[1] <= tmp_z[1] + fp_m(mat[11], w);
    tmp_w[1] <= tmp_w[1] + fp_m(mat[15], w);

    tmp_x[2] <= tmp_x[2] + fp_m(mat[3], w);
    tmp_y[2] <= tmp_y[2] + fp_m(mat[7], w);
    tmp_z[2] <= tmp_z[2] + fp_m(mat[11], w);
```

```
                    tmp_w[2] <= tmp_w[2] + fp_m(mat[15], w);
                    state <= S_OUTPUT;
                end

                default: begin end
            endcase

            color1_r <= {8'b0,color_in1[23:16],16'b0};
            color1_g <= {8'b0,color_in1[15:8],16'b0};
            color1_b <= {8'b0,color_in1[7:0],16'b0};
            color2_r <= {8'b0,color_in2[23:16],16'b0};
            color2_g <= {8'b0,color_in2[15:8],16'b0};
            color2_b <= {8'b0,color_in2[7:0],16'b0};
            color3_r <= {8'b0,color_in3[23:16],16'b0};
            color3_g <= {8'b0,color_in3[15:8],16'b0};
            color3_b <= {8'b0,color_in3[7:0],16'b0};
        end

        S_OUTPUT: begin
            calc_counter <= 0;
            output_triangle();
            out_data_valid <= 1;
            done_out = done_in;
            stall_out <= 0;
            state <= S_HOLD;
        end

        S_HOLD: begin
            stall_out <= 1;
            if (!stall_in) begin
                out_data_valid <= 0;
                state <= S_IDLE;
            end
        end

    endcase
  end
 end
endmodule
```

6.6 HARDWARE: RASTERIZER

Listing 6: rasterizer.sv

```
module rasterizer (
```

```
    input clock,
    input reset,
    input [31:0] x1,
    input [31:0] y1,
    input [31:0] z1,
    input [31:0] x2,
    input [31:0] y2,
    input [31:0] z2,
    input [31:0] x3,
    input [31:0] y3,
    input [31:0] z3,
    input [23:0] color1, //RGB for v1 = (x1_t, y1_t)
    input [23:0] color2, //RGB for v2 = (x2_t, y2_t)
    input [23:0] color3, //RGB for v3 = (x3_t, y3_t)
    input [25:0] addr_in, //frame buffer base
    input in_data_valid,
    input done_in,//
    input stall_in,//
    output [25:0] addr_out,
    //output [23:0] color_out,
    output logic [23:0] color_out_1,
    output logic [23:0] color_out_2,
    output logic [23:0] color_out_3,
    output logic [31:0] w1_out,
    output logic [31:0] w2_out,
    output logic [31:0] w3_out,
    output [31:0] depth_out,
    output output_valid,
    output stall_out,//
    output done_out//
);

    reg signed [31:0] x1_t;
    reg signed [31:0] y1_t;
    reg signed [31:0] z1_t;
    reg signed [31:0] x2_t;
    reg signed [31:0] y2_t;
    reg signed [31:0] z2_t;
    reg signed [31:0] x3_t;
    reg signed [31:0] y3_t;
    reg signed [31:0] z3_t;
    reg [23:0] color1_t;
    reg [23:0] color2_t;
    reg [23:0] color3_t;
```

```
reg [25:0] addr_in_t;

logic [1:0] counter;


logic signed [31:0] cur_x;
logic signed [31:0] cur_y;
logic signed [31:0] e12;
logic signed [31:0] e23;
logic signed [31:0] e31;
logic signed [31:0] e12_2;
logic signed [31:0] e23_2;
logic signed [31:0] e31_2;
logic is_inside;

//640 x 480

logic [31:0] minX_tmp;
logic [31:0] minY_tmp;
logic [31:0] maxX_tmp;
logic [31:0] maxY_tmp;

reg [31:0] minX;
reg [31:0] minY;
reg [31:0] maxX;
reg [31:0] maxY;

//fixed point multiplication
function logic signed [31:0] fp_m(
        input logic signed [31:0] a,
        input logic signed [31:0] b
    );
    fp_m = (32'(a) * 64'(b)) >>> 16;
endfunction

function logic signed [31:0] byte_to_fp(
        input logic [7:0] b
    );

    byte_to_fp = {8'b0, b, 16'b0};
endfunction

function logic [7:0] fp_to_byte(
        input logic signed [31:0] f
```

```
    );
    fp_to_byte = f[23:16];
endfunction

always_comb begin
    if (x1[31:16] < x2[31:16])
        maxX_tmp = x2;
    else
        maxX_tmp = x1;

    if (maxX_tmp[31:16] < x3[31:16])
        maxX_tmp = x3;
end

always_comb begin
    if (x1[31:16] < x2[31:16])
        minX_tmp = x1;
    else
        minX_tmp = x2;

    if (minX_tmp[31:16] > x3[31:16])
        minX_tmp = x3;
end

always_comb begin
    if (y1[31:16] < y2[31:16])
        maxY_tmp = y2;
    else
        maxY_tmp = y1;

    if (maxY_tmp[31:16] < y3[31:16])
        maxY_tmp = y3;
end

always_comb begin
    if (y1[31:16] < y2[31:16])
        minY_tmp = y1;
    else
        minY_tmp = y2;

    if (minY_tmp[31:16] > y3[31:16])
        minY_tmp = y3;
end
```

```
logic signed [31:0] w1;
logic signed [31:0] w2;
logic signed [31:0] w3;
logic signed [31:0] w1_tmp;
logic signed [31:0] w1_tmp2;
logic signed [31:0] w2_tmp;
logic signed [31:0] w2_tmp2;
logic signed [31:0] denom;
logic [23:0] cur_color;
logic signed[31:0] cur_depth;

logic [25:0] tmp_addr_out;
logic [23:0] tmp_color_out;

reg signed [31:0] denom_inv;
reg signed [31:0] denom_inv_reg;

reg signed [31:0] denom1;
reg signed [31:0] denom2;
//color interpolation using Barycentric Coordinates
always_comb begin
    //denom = fp_m(y2_t - y3_t, x1_t - x3_t) + fp_m(x3_t - x2_t,
       y1_t - y3_t);
end

logic signed [31:0] cur_x_int;
logic signed [31:0] cur_y_int;

logic signed [31:0] x1_t_int;
logic signed [31:0] y1_t_int;
logic signed [31:0] x2_t_int;
logic signed [31:0] y2_t_int;
logic signed [31:0] x3_t_int;
logic signed [31:0] y3_t_int;

always_comb begin
    cur_x_int = cur_x >>> 16;
    cur_y_int = cur_y >>> 16;
    x1_t_int = x1_t >>> 16;
    y1_t_int = y1_t >>> 16;
    x2_t_int = x2_t >>> 16;
    y2_t_int = y2_t >>> 16;
    x3_t_int = x3_t >>> 16;
    y3_t_int = y3_t >>> 16;
```

```
    /*
    e12 = (signed'(cur_x_int - x1_t_int) * signed'(y2_t_int -
        y1_t_int) - signed'(cur_y_int - y1_t_int) * signed'(
        x2_t_int - x1_t_int)) >= 0;
    e23 = (signed'(cur_x_int - x2_t_int) * signed'(y3_t_int -
        y2_t_int) - signed'(cur_y_int - y2_t_int) * signed'(
        x3_t_int - x2_t_int)) >= 0;
    e31 = (signed'(cur_x_int - x3_t_int) * signed'(y1_t_int -
        y3_t_int) - signed'(cur_y_int - y3_t_int) * signed'(
        x1_t_int - x3_t_int)) >= 0;
    */
    //e23 = (signed'(cur_x - x2_t) * signed'(y3_t - y2_t)) -
        signed'(cur_y - y2_t), signed'(x3_t - x2_t))) >= 0;
    //e31 = (fp_m(signed'(cur_x - x3_t),signed'(y1_t - y3_t)) -
        fp_m(signed'(cur_y - y3_t), signed'(x1_t - x3_t))) >= 0;

    is_inside = e12 >= 0 && e23 >= 0 && e31 >= 0;
end

function void  move_to_next();
    cur_x = cur_x + (1 << 16);

    if (cur_x > maxX) begin
        $display("newline");
        cur_x = minX;
        cur_y = cur_y + (1 << 16);
    end
endfunction

typedef enum logic[2:0] {R_IDLE, R_START_NEW_TRI, R_PIXEL_CALC,
   R_PIXEL_CALC2, R_RASTERIZE, R_WAIT} r_state_t;
r_state_t r_state;


lpm_divide
`ifdef VERILATOR
#(.lpm_widthn(40),
.lpm_widthd(32),
.lpm_nrepresentation("SIGNED"),
.lpm_drepresentation("SIGNED"),
.lpm_pipeline(16)) area_divider(
`else
#(.LPM_WIDTHN(40),
```

```
    .LPM_WIDTHD(32),
    .LPM_NREPRESENTATION("SIGNED"),
    .LPM_DREPRESENTATION("SIGNED"),
    .MAXIMIZE_SPEED(9),
    .LPM_PIPELINE(16)) area_divider(
    `endif
    .clock(clock),
    .clken(1'b1),
    .numer(32'b1 << 32),
    .denom(denom),
    .quotient(denom_inv)
);

logic [4:0] div_counter;

always_ff @(posedge clock or negedge reset) begin
    if (!reset) begin
        cur_x = 0;
        cur_y = 0;
        output_valid <= 0;
        done_out = 0;
        stall_out <= 1;
        div_counter <= 0;
        r_state <= R_IDLE;
    end else begin
        $display("␣rasterizer:␣[%d]==x[%d]==y[%d]", r_state, cur_x >>
            16, cur_y >> 16);
        $display("␣rasterizer:␣MAXY:[%d]", maxY>>16);
        $display("␣rasterizer:␣MAXX:[%d]", maxX>>16);
        $display("␣rasterizer:␣OUTPUT_VALID[%d]", output_valid);
        case(r_state)
            R_IDLE: begin
                $display( "rasterizer:␣in_data_valid␣=␣[%d]",
                    in_data_valid);
                if (in_data_valid) begin
                    $display("␣rasterizer:␣new␣triangle");
                    stall_out <= 0;
                    x1_t <= x1;
                    y1_t <= y1;
                    z1_t <= z1;
                    x2_t <= x2;
                    y2_t <= y2;
                    z2_t <= z2;
                    x3_t <= x3;
```

```
            y3_t <= y3;
            z3_t <= z3;
            color1_t <= color1;
            color2_t <= color2;
            color3_t <= color3;
            addr_in_t <= addr_in;
            div_counter <= 0;

            maxX <= maxX_tmp;
            maxY <= maxY_tmp;
            minX <= minX_tmp;
            minY <= minY_tmp;

            cur_x = minX_tmp;
            cur_y = minY_tmp;

            denom1 <= fp_m(y2 - y3, x1 - x3);
            denom2 <= fp_m(x3 - x2, y1 - y3);
            r_state <= R_START_NEW_TRI;
        end
        output_valid <= 0;
    end
    R_START_NEW_TRI: begin
        stall_out <= 1;
        if (div_counter == 0) begin
            denom <= denom1 + denom2;
            div_counter <= div_counter + 1;
        end else if (div_counter == 17) begin
            div_counter <= 0;
            denom_inv_reg <= denom_inv;
            $display("␣rasaterizer:␣denom␣=␣%f,␣denom_inv␣=␣%f
                ",
                 $itor(denom) / $itor(1 << 16),
                 $itor(denom_inv) / $itor(1 << 16)
            );
            e12 <= signed'(cur_x_int - x1_t_int) * signed'(
               y2_t_int - y1_t_int);
            e23 <= signed'(cur_x_int - x2_t_int) * signed'(
               y3_t_int - y2_t_int);
            e31 <= signed'(cur_x_int - x3_t_int) * signed'(
               y1_t_int - y3_t_int);

            e12_2 <= signed'(cur_y_int - y1_t_int) * signed'(
               x2_t_int - x1_t_int);
```

```
            e23_2 <= signed'(cur_y_int - y2_t_int) * signed'(
                x3_t_int - x2_t_int);
            e31_2 <= signed'(cur_y_int - y3_t_int) * signed'(
                x1_t_int - x3_t_int);
            w1_tmp <= fp_m(y2_t - y3_t, cur_x - x3_t); // +
                fp_m(x3_t - x2_t, cur_y - y3_t);
            w2_tmp <= fp_m(y3_t - y1_t, cur_x - x3_t); // +
                fp_m(x1_t - x3_t, cur_y - y3_t);
            r_state <= R_PIXEL_CALC;
        end else begin
            div_counter <= div_counter + 1;
        end
    end

    R_PIXEL_CALC: begin
        e12 <= e12 - e12_2;
        e23 <= e23 - e23_2;
        e31 <= e31 - e31_2;
        //e12 <= signed'(cur_x_int - x1_t_int) * signed'(
            y2_t_int - y1_t_int) - signed'(cur_y_int - y1_t_int
            ) * signed'(x2_t_int - x1_t_int);
        //e23 <= signed'(cur_x_int - x2_t_int) * signed'(
            y3_t_int - y2_t_int) - signed'(cur_y_int - y2_t_int
            ) * signed'(x3_t_int - x2_t_int);
        //e31 <= signed'(cur_x_int - x3_t_int) * signed'(
            y1_t_int - y3_t_int) - signed'(cur_y_int - y3_t_int
            ) * signed'(x1_t_int - x3_t_int);
        //w1 <= fp_m(w1_tmp, denom_inv_reg);
        //w2 <= fp_m(w2_tmp, denom_inv_reg);
        w1_tmp2 <= fp_m(x3_t - x2_t, cur_y - y3_t);
        w2_tmp2 <= fp_m(x1_t - x3_t, cur_y - y3_t);
        r_state <= R_PIXEL_CALC2;
    end

    R_PIXEL_CALC2: begin
        w1_out <= fp_m(w1_tmp + w1_tmp2, denom_inv_reg);
        w2_out <= fp_m(w2_tmp + w2_tmp2, denom_inv_reg);
        r_state <= R_RASTERIZE;
    end
    R_RASTERIZE: begin
        if (e12 >= 0 && e23 >= 0 && e31 >= 0) begin
            $display("␣rasterizer:␣is␣inside");
            if (!output_valid) begin
                addr_out <= addr_in_t + ((cur_y >> 16) * 640 +
```

```
                (cur_x >> 16) << 3);
            $display("␣rasterizer:␣addr_out␣=␣%x",
               addr_out);
            color_out_1 <= color1_t;
            color_out_2 <= color2_t;
            color_out_3 <= color3_t;

            w3_out <= (1 << 16) - w1_out - w2_out;
            depth_out <= fp_m(w1_out, z1_t) + fp_m(w2_out,
                z2_t) + fp_m((1 << 16) - w1_out - w2_out,
               z3_t);
            output_valid <= 1;
            move_to_next();
            if (cur_y > maxY) begin
                done_out = done_in;
                r_state <= R_IDLE;
            end else
            r_state <= R_WAIT;
        end
    end else begin
        output_valid <= 0;
        move_to_next();
        if (cur_y > maxY) begin
            done_out = done_in;
            r_state <= R_IDLE;
        end else
        begin
            r_state <= R_WAIT;
        end
    end
end
R_WAIT: begin
    $display("␣rasterizer:␣STALL_IN[%d]", stall_in);
    if (output_valid <= 0) begin
        e12 <= signed'(cur_x_int - x1_t_int) * signed'(
           y2_t_int - y1_t_int);
        e23 <= signed'(cur_x_int - x2_t_int) * signed'(
           y3_t_int - y2_t_int);
        e31 <= signed'(cur_x_int - x3_t_int) * signed'(
           y1_t_int - y3_t_int);
        e12_2 <= signed'(cur_y_int - y1_t_int) * signed'(
           x2_t_int - x1_t_int);
        e23_2 <= signed'(cur_y_int - y2_t_int) * signed'(
           x3_t_int - x2_t_int);
```

```
                    e31_2 <= signed'(cur_y_int - y3_t_int) * signed'(
                        x1_t_int - x3_t_int);
                    w1_tmp <= fp_m(y2_t - y3_t, cur_x - x3_t); // +
                        fp_m(x3_t - x2_t, cur_y - y3_t);
                    w2_tmp <= fp_m(y3_t - y1_t, cur_x - x3_t); // +
                        fp_m(x1_t - x3_t, cur_y - y3_t);
                    output_valid <= 0;
                    r_state <= R_PIXEL_CALC;
                end else
                if (!stall_in) begin
                    if (cur_y > maxY) begin
                        done_out = done_in;
                        r_state <= R_IDLE;
                    end else begin
                        //move_to_next();
                        e12 <= signed'(cur_x_int - x1_t_int) * signed
                            '(y2_t_int - y1_t_int);
                        e23 <= signed'(cur_x_int - x2_t_int) * signed
                            '(y3_t_int - y2_t_int);
                        e31 <= signed'(cur_x_int - x3_t_int) * signed
                            '(y1_t_int - y3_t_int);
                        e12_2 <= signed'(cur_y_int - y1_t_int) *
                            signed'(x2_t_int - x1_t_int);
                        e23_2 <= signed'(cur_y_int - y2_t_int) *
                            signed'(x3_t_int - x2_t_int);
                        e31_2 <= signed'(cur_y_int - y3_t_int) *
                            signed'(x1_t_int - x3_t_int);
                        w1_tmp <= fp_m(y2_t - y3_t, cur_x - x3_t); //
                            + fp_m(x3_t - x2_t, cur_y - y3_t);
                        w2_tmp <= fp_m(y3_t - y1_t, cur_x - x3_t); //
                            + fp_m(x1_t - x3_t, cur_y - y3_t);
                        output_valid <= 0;
                        r_state <= R_PIXEL_CALC;
                    end
                end
            end
        endcase
    end
end
endmodule
```

6.7 HARDWARE: RASTERIZER_FETCH_LOGIC

Listing 7: rasterizer_fetch_logic.sv

```
module rasterizer_fetch_logic (
    input clock,
    input reset,
    /* bus interface */
    output [25:0] master_address,
    output master_read,
    output master_write,
    output [3:0] master_byteenable,
    input [31:0] master_readdata,
    input master_readdatavalid,
    output [31:0]master_writedata,
    input master_waitrequest,
    /* pipeline control */
    input stall_in,
    output stall_out,
    input done_in,
    output done_out,
    /* data input */
    input input_valid,
    input [25:0] addr_in,
    //input [23:0] color_in,
    input [23:0] color_in_1,
    input [23:0] color_in_2,
    input [23:0] color_in_3,
    input signed [31:0] w1,
    input signed [31:0] w2,
    input signed [31:0] w3,
    input [31:0] depth_in,
    /* data output */
    output output_valid,
    output [25:0] addr_out,
    output [31:0] old_depth_out,
    output [31:0] new_depth_out,
    output [23:0] color_out
);

    reg [25:0] addr_in_r;
    reg [23:0] color_in_r;
    reg [31:0] depth_in_r;
    reg input_valid_r;
    reg done_in_r;

    logic [95:0] data_in;
    logic [95:0] data_out;
```

```
    logic rdreq;
    logic wrreq;
    logic almost_full;
    logic almost_empty;
    logic full;
    logic empty;
    logic half_full;

    reg [95:0] data_out_reg;
    assign addr_out      = data_out_reg[25:0];
    assign color_out     = data_out_reg[49:26];
    assign new_depth_out = data_out_reg[81:50];
    assign done_out      = data_out_reg[82];

    assign rdreq = master_readdatavalid;

    fifo fifo(
        .din(data_in),
        .dout(data_out),
        .wr(wrreq),
        .rd(rdreq),
        .reset(reset),
        .clk(clock),
        .full(full),
        .empty(empty),
        .half_full(half_full),
        .almost_full(almost_full),
        .almost_empty(almost_empty)
    );

//fixed point multiplication
    function logic signed [31:0] fp_m(
            input logic signed [31:0] a,
            input logic signed [31:0] b
        );
        fp_m = (32'(a) * 64'(b)) >>> 16;
    endfunction

    function logic signed [31:0] byte_to_fp(
            input logic [7:0] b
        );

        byte_to_fp = {8'b0, b, 16'b0};
    endfunction
```

```
    function logic [7:0] fp_to_byte(
            input logic signed [31:0] f
        );
        fp_to_byte = f[23:16];
    endfunction

    reg signed [31:0] color_step [8:0];
    always_ff @(posedge clock or negedge reset) begin
        if (!reset) begin

        end else begin
            if (enqueue || !input_valid_r) begin
                //color_in_r[7:0] = fp_to_byte(w1 * color_in_1[7:0] +
                   w2 * color_in_2[7:0] + w3 * color_in_3[7:0]);
                //color_in_r[15:8] = fp_to_byte(w1 * color_in_1[15:8]
                   + w2 * color_in_2[15:8] + w3 * color_in_3[15:8]);
                //color_in_r[23:16] = fp_to_byte(w1 * color_in_1
                   [23:16] + w2 * color_in_2[23:16] + w3 * color_in_3
                   [23:16]);

                color_step[0] <= w1 * color_in_1[7:0];
                color_step[1] <= w2 * color_in_2[7:0];
                color_step[2] <= w3 * color_in_3[7:0];
                color_step[3] <= w1 * color_in_1[15:8];
                color_step[4] <= w2 * color_in_2[15:8];
                color_step[5] <= w3 * color_in_3[15:8];
                color_step[6] <= w1 * color_in_1[23:16];
                color_step[7] <= w2 * color_in_2[23:16];
                color_step[8] <= w3 * color_in_3[23:16];
//color_in_r[15:8] = fp_to_byte(fp_m(w1, byte_to_fp(color_in_1[15:8]))
    + fp_m(w2, byte_to_fp(color_in_2[15:8])) + fp_m(w3, byte_to_fp(
   color_in_3[15:8])));
//              color_in_r[23:16] = fp_to_byte(fp_m(w1, byte_to_fp(
   color_in_1[23:16])) + fp_m(w2, byte_to_fp(color_in_2[23:16])) +
   fp_m(w3, byte_to_fp(color_in_3[23:16])));
                addr_in_r <= addr_in;
                depth_in_r <= depth_in;
                input_valid_r <= input_valid;
            end
        end
    end

    assign color_in_r[7:0]   = fp_to_byte(color_step[0] + color_step
```

```
   [1] + color_step[2]);
assign color_in_r[15:8]  = fp_to_byte(color_step[3] + color_step
   [4] + color_step[5]);
assign color_in_r[23:16] = fp_to_byte(color_step[6] + color_step
   [7] + color_step[8]);

typedef enum logic { S_IDLE,  S_HOLD } state_t;
state_t state;
state_t next_state;

reg enqueue;
always @* begin
    enqueue = 0;
    case (state)
        S_IDLE: begin
            if (full) begin
                next_state = S_IDLE;
            end
            else if (input_valid_r) begin
                next_state = S_HOLD;
                enqueue = 1;
            end else
            next_state = S_IDLE;
        end

        S_HOLD: begin
            if (!master_waitrequest && !stall_in) begin
                if (!almost_full && input_valid_r) begin
                    next_state = S_HOLD;
                    enqueue = 1;
                end else begin
                    enqueue = 0;
                    next_state = S_IDLE;
                end
            end else begin
                if (master_waitrequest) begin
                    next_state = S_HOLD;
                end else begin
                    next_state = S_IDLE;
                end
                enqueue = 0;
            end
        end
    endcase
```

```
end

assign stall_out = !enqueue;

always_ff @(posedge clock or negedge reset) begin
    if (!reset) begin
        wrreq <= 0;
        output_valid <= 0;
        state <= S_IDLE;
        //stall_out <= 1;
    end else begin
        if (full)
            $display("depth_fetcher: fifo is full");
        // deal with input port
        //
        $display("depth_fetcher: master_waitrequest = %d",
           master_waitrequest);
        $display("input_valid = %d", input_valid);
        if (enqueue)
            begin
                // enqueue the fetch request
                $display("depth_fetcher: enqueue fetch request
                   addr = %x", addr_in);
                wrreq <= 1;
                data_in[25:0] <= addr_in_r;
                data_in[49:26] <= color_in_r;
                data_in[81:50] <= depth_in_r;
                data_in[82] <= done_in_r;
                data_in[95:83] <= 0;

                master_address <= addr_in_r + 4;
                master_write <= 0;
                master_byteenable <= 4'b11;
            end
        else begin
            wrreq <= 0;
            $display("depth_fetcher: not enqueuing.");
        end

        if (next_state == S_IDLE) begin
            $display("depth_fetcher: master_read = 0");
            master_read <= 0;
        end else begin
            master_read <= 1;
```

```
                $display("depth_fetcher: master_read = 1");
            end

            $display("depth_fetcher: next_state = %d", next_state);

            state <= next_state;

            // check if there is any output from sdram
            if (master_readdatavalid) begin
                $display("depth_fetcher: got fetch request addr = %x",
                    data_out[25:0]);
                assert(data_out[25:0] != addr_out);
                old_depth_out <= master_readdata;
                data_out_reg <= data_out;
                output_valid <= 1;
                if (empty)
                    $fatal("fifo is empty!");
            end else
            output_valid <= 0;
        end
    end

endmodule
```

6.8 HARDWARE: ZTEST

Listing 8: ztest.sv

```
module ztest (
    input clock,
    input reset,
    input input_valid,
    input [25:0] addr_in, //addr_out from rasterizer_fetch_logic
    input [31:0] old_depth_out,
    output [31:0] new_depth_out,//input->output
    input [31:0] color_in, //color out from rasterizer_fetch_logic
    //output [31:0] color_out,
    //output [25:0] addr_out,
    input done_in,
    output stall_out,
    output done_out,
    output [25:0] master_address,
    output master_read,
    output master_write,
    output [3:0] master_byteenable,
```

```
    input [31:0] master_readdata,
    input master_readdatavalid,
    output [31:0]master_writedata,
    input master_waitrequest
);

    logic [114:0] data_in;
    logic [114:0] data_out;
    logic wrreq;
    logic rdreq;
    logic almost_full;
    logic almost_empty;
    logic full;
    logic empty;
    logic half_full;

    assign master_byteenable = 4'b1111;

    fifo #(.DBITS(115), .SIZE(6))fifo(
        .din(data_in),
        .dout(data_out),
        .wr(wrreq),
        .rd(rdreq),
        .reset(reset),
        .clk(clock),
        .full(full),
        .empty(empty),
        .half_full(half_full),
        .almost_full(almost_full),
        .almost_empty(almost_empty)
    );

    assign stall_out = half_full;

    always_ff @(posedge clock) begin
        if (!almost_full && input_valid) begin
            $display("ztest: receive data from addr = %d", addr_in);
            wrreq <= 1;
            data_in[25:0] <= addr_in;
            data_in[49:26] <= color_in;
            data_in[81:50] <= old_depth_out;
            data_in[113:82] <= new_depth_out;
            data_in[114] <= done_in;
        end
```

```
        else begin
            if (almost_full)
                $display("ztest: almost full!");
            wrreq <= 0;
        end
    end

    wire [25:0] addr_out;
    wire [23:0] color_out;
//wire [31:0] old_depth_out;
//wire [31:0] new_depth_out;
    wire done_out_temp;
    logic [31:0]real_old_depth_out;
    reg [31:0] real_new_depth_out;
    assign addr_out           = data_out[25:0];
    assign color_out          = data_out[49:26];
    assign real_old_depth_out = data_out[81:50];
    assign new_depth_out      = data_out[113:82];
    assign done_out_temp      = data_out[114];

    typedef enum logic[1:0] {S_IDLE, S_WRITE_COLOR, S_WRITE_DEPTH,
       S_POP} state_t;
    state_t state;

    always_ff @(posedge clock or negedge reset) begin
        if (!reset) begin
            done_out <= 0;
            state <= S_IDLE;
        end else begin
            case (state)
                S_IDLE: begin
                    if (!empty) begin
                        if (new_depth_out < real_old_depth_out) begin
                            rdreq <= 1;
                            $display("ztest: writing color to %d",
                               addr_out);
                            master_address <= addr_out;
                            master_writedata <= color_out;
                            master_write <= 1;
                            real_new_depth_out <= new_depth_out;
                            state <= S_WRITE_COLOR;
                        end else begin
                            $display("ztest: failed");
```

```
                rdreq <= 1;
                master_write <= 0;
                done_out <= done_out_temp;
                state <= S_POP;
            end
        end
    end

    S_WRITE_COLOR: begin
        rdreq <= 0;
        if (!master_waitrequest) begin
            $display("ztest: writing depth to %d",
                master_address + 4);
            master_address <= master_address + 4;
            master_writedata <= real_new_depth_out;
            master_write <= 1;
            state <= S_WRITE_DEPTH;
        end
    end

    S_WRITE_DEPTH: begin
        if (!master_waitrequest) begin
            if (!empty) begin
                if (new_depth_out < real_old_depth_out)
                    begin
                    rdreq <= 1;
                    $display("ztest: writing color to %d",
                        addr_out);
                    master_address <= addr_out;
                    master_writedata <= color_out;
                    master_write <= 1;
                    real_new_depth_out <= new_depth_out;
                    state <= S_WRITE_COLOR;
                end else begin
                    $display("ztest: failed");
                    rdreq <= 1;
                    master_write <= 0;
                    done_out <= done_out_temp;
                    state <= S_IDLE;
                end
            end else begin
                master_write <= 0;
                state <= S_IDLE;
            end
```

```
                    end
                end

                S_POP: begin
                    rdreq <= 0;
                    state <= S_IDLE;
                end
            endcase
        end
    end

endmodule // ztest
```

6.9 HARDWARE: RASTERIZER_UNIT

Listing 9: rasterizer_unit.sv

```
module rasterizer_unit (
    input clock,
    input reset,
    // to config_reg
    input [31:0] writedata,
    input logic write,
    input logic read,
    input logic [15:0] address,
    output logic [31:0] readdata,
    output logic test,
    output [25:0] master_address,
    output master_read,
    output master_write,
    output [3:0] master_byteenable,
    input [31:0] master_readdata,
    input master_readdatavalid,
    output [31:0]master_writedata,
    input master_waitrequest,
    output [25:0] master_address_2,
    output master_read_2,
    output master_write_2,
    output [3:0] master_byteenable_2,
    input [31:0] master_readdata_2,
    input master_readdatavalid_2,
    output [31:0]master_writedata_2,
    input master_waitrequest_2,
    output [25:0] master_address_3,
    output master_read_3,
```

```
    output master_write_3,
    output [3:0] master_byteenable_3,
    input [31:0] master_readdata_3,
    input master_readdatavalid_3,
    output [31:0]master_writedata_3,
    input master_waitrequest_3
);


    wire stall1;
    wire stall2;
    wire stall3;
    wire stall4;

    wire done1;
    wire done2;
    wire done3;
    wire done4;
    wire done5;

//output of config_reg

    logic [31:0] MV [15:0];
    logic [31:0] MVP [15:0];
    logic [31:0] lighting [2:0];
    logic [25:0] frame_buffer_base;
    logic [25:0] vertex_buffer_base;
    logic do_render;

    logic [23:0] color1;
    logic [23:0] color2;
    logic [23:0] color3;

//output of vertex fetch
    logic output_valid;
    logic fetch_busy;
    logic fetch_finish;
    logic [31:0] vertex_out[14:0];
    logic input_data_valid;

//output of vertex cal
    logic [31:0] x_out[3:0];
    logic [31:0] y_out[3:0];
    logic [31:0] z_out[3:0];
```

```
    logic [31:0] w_out[3:0];
    logic out_data_valid;

//output of rasterizer
    logic [25:0] addr_out;
    logic [23:0] color_out_1;
    logic [23:0] color_out_2;
    logic [23:0] color_out_3;
    logic signed [31:0] w1;
    logic signed [31:0] w2;
    logic signed [31:0] w3;
    logic rasterizer_output_valid;

//fetch logic
    logic [31:0] depth_in;
    logic fetch_output_valid;
    logic [25:0] fetch_addr_out;
    logic [31:0] old_depth_out;
    logic [31:0] new_depth_out;
    logic [23:0] fetch_color_out;
    logic wait_request;

//final output
    logic [25:0] final_addr_out;
    logic [23:0] final_color_out;


    config_reg c_reg (
        .clk(clock),
        .reset_n(reset),
        .writedata(writedata),
        .write(write),
        .read(read),
        .address(address),
        .readdata(readdata),
        .MV(MV),
        .MVP(MVP),
        .lighting(lighting),
        .frame_buffer_base(frame_buffer_base),
        .vertex_buffer_base(vertex_buffer_base),
        .start_render(do_render),
        .test(test),
        .done_in(done5));
```

```
rasterizer_vertex_fetch vertex_fetch (
    .clock(clock),
    .reset(reset),
    .master_address(master_address),
    .master_read(master_read),
    .master_write(master_write),
    .master_byteenable(master_byteenable),
    .master_readdata(master_readdata),
    .master_readdatavalid(master_readdatavalid),
    .master_writedata(master_writedata),
    .master_waitrequest(master_waitrequest),
    /* pipeline interface */
    .fetch_enable(do_render),
    .vertex_buffer_base(vertex_buffer_base),
    .stall_in(stall1),
    .done_out(done1),
    /* data for one triangle is ready */
    .output_valid(input_data_valid),
    /* triangle data 15-32bits x1,y1,z1,rgb, ..., nx, ny, nz */
    .vertex_out(vertex_out));


vertex_calc v_calc (
    .clock(clock),
    .reset(reset),
    .mat(MVP),
    .vertex_in(vertex_out),
    .lighting(lighting),
    .input_data_valid(input_data_valid),
    .x_out(x_out),
    .y_out(y_out),
    .z_out(z_out),
    .w_out(w_out),
    .done_in(done1),
    .done_out(done2),
    .color_input1(vertex_out[3][23:0]),
    .color_input2(vertex_out[7][23:0]),
    .color_input3(vertex_out[11][23:0]),
    .color_out1(color1),
    .color_out2(color2),
    .color_out3(color3),
    .stall_in(stall2),
    .stall_out(stall1),
```

```
    .out_data_valid(out_data_valid));
always@(x_out )begin
    $display("vertex_cal_out x1:%d y1:%d z1:%d",x_out[0][31:16],
        y_out[0][31:16],z_out[0][31:16]);
    $display("vertex_cal_out x2:%d y2:%d z2:%d",x_out[1][31:16],
        y_out[1][31:16],z_out[1][31:16]);
    $display("vertex_cal_out x3:%d y3:%d z3:%d",x_out[2][31:16],
        y_out[2][31:16],z_out[2][31:16]);
end

rasterizer raster (
    .clock(clock),
    .reset(reset),
    .x1(x_out[0]),
    .y1(y_out[0]),
    .x2(x_out[1]),
    .y2(y_out[1]),
    .x3(x_out[2]),
    .y3(y_out[2]),
    .z1(z_out[0]),
    .z2(z_out[1]),
    .z3(z_out[2]),
    .color1(color1), //from vertex calc
    .color2(color2),
    .color3(color3),
    .addr_in(frame_buffer_base), //from config_reg
    .in_data_valid(out_data_valid),

    .color_out_1(color_out_1),
    .color_out_2(color_out_2),
    .color_out_3(color_out_3),
    .w1_out(w1),
    .w2_out(w2),
    .w3_out(w3),

    .done_in(done2),
    .stall_in(stall3),
    .addr_out(addr_out),
    .depth_out(depth_in),
    .output_valid(rasterizer_output_valid),

    .stall_out(stall2),
    .done_out(done3)
```

```
);

rasterizer_fetch_logic fetch_logic (
    .clock(clock),
    .reset(reset),
    .master_address(master_address_2),
    .master_read(master_read_2),
    .master_write(master_write_2),
    .master_byteenable(master_byteenable_2),
    .master_readdata(master_readdata_2),
    .master_readdatavalid(master_readdatavalid_2),
    .master_writedata(master_writedata_2),
    .master_waitrequest(master_waitrequest_2),

    .input_valid(rasterizer_output_valid), //from rasterizer
    .addr_in(addr_out), //from rasterizer
    //.color_in(color_out), //from rasterizer
    .color_in_1(color_out_1),
    .color_in_2(color_out_2),
    .color_in_3(color_out_3),
    .w1(w1),
    .w2(w2),
    .w3(w3),
    .depth_in(depth_in), /// from bus...

    .output_valid(fetch_output_valid),
    .addr_out(fetch_addr_out),
    .old_depth_out(old_depth_out),
    .new_depth_out(new_depth_out),
    .color_out(fetch_color_out),
    .done_in(done3), //from rasterizer
    .done_out(done4),
    .stall_in(stall4),
    .stall_out(stall3));

ztest z_test (
    .clock(clock),
    .reset(reset),
    .input_valid(fetch_output_valid),
    .addr_in(fetch_addr_out),
    .old_depth_out(old_depth_out),
    .new_depth_out(new_depth_out),
    .color_in(fetch_color_out),
```

```
        .done_in(done4),
        .done_out(done5),

        .master_address(master_address_3),
        .master_read(master_read_3),
        .master_write(master_write_3),
        .master_byteenable(master_byteenable_3),
        .master_readdata(master_readdata_3),
        .master_readdatavalid(master_readdatavalid_3),
        .master_writedata(master_writedata_3),
        .master_waitrequest(master_waitrequest_3),

        .stall_out(stall4));

//need one to write to SDRAM controller

endmodule // rasterizer_unit
```

6.10 HARDWARE: FIFO

Listing 10: fifo.sv

```
module fifo (
    input logic clk,
    input logic reset,
    input logic wr,
    input logic rd,
    input logic [DBITS-1:0] din,
    output logic empty,
    output logic full,
    output logic almost_full,
    output logic half_full,
    output logic almost_empty,
    output logic [DBITS-1:0] dout
);

    //DBITS: # of address bits
    //SIZE: 2^SZIE elements in buffer
    parameter DBITS = 96,
        SIZE = 4;

    //16 element fifo, each element is 26 bis
    logic [DBITS-1:0] buffer[2**SIZE - 1:0];
    //write pointer
    logic [SIZE - 1:0] wr_ptr;
```

```
//read pointer
logic [SIZE - 1:0] rd_ptr;
logic [SIZE:0] counter;

//wire almost_full;
//wire almost_empty;

assign almost_full  = counter >= (2**SIZE - 1);
assign almost_empty = counter < 2;

//counter keeps track of number of elements in buffer
assign empty     = (counter == 0);
assign full      = (counter == 2**SIZE);
assign half_full = (counter >= ((2**SIZE) >> 1));

assign dout = buffer[rd_ptr];

logic [SIZE:0] new_counter;
always_ff @(posedge clk or negedge reset)
    begin
        if (!reset) begin
            wr_ptr <= 0;
            rd_ptr <= 0;
            counter <= 0;
            new_counter = 0;
        end

        else begin
            assert(rd_ptr + counter == wr_ptr);
            new_counter = counter;
            if (rd && !empty) begin
                buffer[rd_ptr] <= 0; // poison
                rd_ptr <= rd_ptr + 1;
                new_counter = new_counter - 1;
            end

            if (wr && !full) begin
                buffer[wr_ptr] <= din;
                wr_ptr <= wr_ptr + 1;
                new_counter = new_counter + 1;
            end
            counter <= new_counter;
        end
    end
```

```
endmodule
```

6.11 PLY_LOADER

Listing 11: loader.h

```
#pragma once

#include <array>
#include <vector>
#include <math.h>

using namespace std;

class Vec3 {

public:
        Vec3() : X(0), Y(0), Z(0) {}
        Vec3(double X_, double Y_, double Z_) : X(X_), Y(Y_), Z(Z_) {}

        Vec3 operator+(const Vec3& right) const {
                return Vec3(this->X + right.X, this->Y + right.Y, this
                   ->Z + right.Z);
        }

        Vec3 operator-(const Vec3& right) const {
                return Vec3(this->X - right.X, this->Y - right.Y, this
                   ->Z - right.Z);
        }

        Vec3 operator-() {
                return Vec3(-this->X, -this->Y, -this->Z);
        }

        void operator=(const Vec3& right) {
                this->X = right.X;
                this->Y = right.Y;
                this->Z = right.Z;
        }

        void load(const array<double, 3>& a) {
                this->X = a[0];
                this->Y = a[1];
                this->Z = a[2];
```

```
        }

        Vec3 crossProduct(const Vec3& v2) {
                return Vec3(this->Y*v2.Z - this->Z*v2.Y,
                                        this->Z*v2.X - this->X*v2.Z,
                                        this->X*v2.Y - this->Y*v2.X);
        }

        double dotProduct(const Vec3& v2) {
                return (this->X*v2.X + this->Y*v2.Y + this->Z*v2.Z);
        }

    void normalize() {
        double x = X, y = Y, z = Y;
        X = X/sqrt(x*x + y*y + z*z);
        Y = Y/sqrt(x*x + y*y + z*z);
        Z = Z/sqrt(x*x + y*y + z*z);
    }

        double X, Y, Z;
};
```

Listing 12: loader.cpp

```
#include "happly.h"
#include "loader.h"
#include <string.h>
#include <tgmath.h>
#include <bitset>
#include <fstream>

typedef uint32_t fixed_point_t;
#define FIXED_POINT_FRACTIONAL_BITS 16
using namespace std;

/* calculate normal vector for each triangle */
Vec3 calcFaceNormal(Vec3 vpos[3], Vec3 vnormal[3]) {
        Vec3 p0 = vpos[1] - vpos[0];
        Vec3 p1 = vpos[2] - vpos[0];
        Vec3 faceNormal = p0.crossProduct(p1);

        Vec3 vertexNormal = vnormal[0];
        double dot = faceNormal.dotProduct(vertexNormal);
    faceNormal.normalize();
```

```
        return (dot < 0) ? -faceNormal : faceNormal;
}

/* double to fixed point conversion */
inline fixed_point_t float2fixed(double input)
{
    return (fixed_point_t)(round(input * (1 <<
       FIXED_POINT_FRACTIONAL_BITS)));
}

int main(int argc, char** argv) {
        // Construct the data object by reading from file
        happly::PLYData plyIn("simple.ply");

        // Get mesh-style data from the object
        std::vector<std::array<double, 3>> vPos = plyIn.
           getVertexPositions();
        std::vector<std::array<unsigned char, 3>> vColor = plyIn.
           getVertexColors();
        std::vector<std::vector<size_t>> fInd = plyIn.getFaceIndices<
           size_t>();
        std::vector<double> vNx = plyIn.getElement("vertex").
           getProperty<double>("nx");
        std::vector<double> vNy = plyIn.getElement("vertex").
           getProperty<double>("ny");
        std::vector<double> vNz = plyIn.getElement("vertex").
           getProperty<double>("nz");

        std::vector<std::array<double, 3>> fNormal(fInd.size(),
           {0,0,0});


        std::cout << vPos.size() << " vectex in total" << std::endl;
        std::cout << fInd.size() << " triangles in total" << std::endl
           ;

        for (int i = 0; i < fInd.size(); i++) {
                Vec3 vpos[3];
                Vec3 vnormal[3];

                vpos[0].load(vPos[fInd[i][0]]);
                vpos[1].load(vPos[fInd[i][1]]);
                vpos[2].load(vPos[fInd[i][2]]);
```

```
            vnormal[0].load({vNx[fInd[i][0]], vNy[fInd[i][0]], vNz
                [fInd[i][0]]});
            vnormal[1].load({vNx[fInd[i][1]], vNy[fInd[i][1]], vNz
                [fInd[i][1]]});
            vnormal[2].load({vNx[fInd[i][2]], vNy[fInd[i][2]], vNz
                [fInd[i][2]]});
            // calculate normal vector
            Vec3 faceNormal = calcFaceNormal(vpos, vnormal);

            fNormal[i] = {faceNormal.X, faceNormal.Y, faceNormal.Z
                };
    }

    // fixed point test
    uint64_t u;
    memcpy(&u, &vPos[0][2], sizeof(vPos[0][2]));
    std::cout << "before␣conversion:␣" << vPos[0][2] << "␣" << std
        ::hex << u << std::endl;

    std::cout << "after␣conversion:␣" << std::bitset<32>(
        float2fixed(vPos[0][2])) << std::endl;
cout << hex << float2fixed(vPos[0][2]) << endl;


    /* data_out: X|Y|Z—X|Y|Z—X|Y|Z—NX|NY|NZ
     * color_out: R|G|B—R|G|B—R|G|B
     * index for each triangle
     */
    std::vector<std::vector<fixed_point_t>> data_out;
    std::vector<std::vector<unsigned char>> color_out;

    for (int i = 0; i < fInd.size(); i++) {

            /* get data for each triangle*/
            std::vector<fixed_point_t> pos_tmp;
            std::vector<unsigned char> col_tmp;
            for (int j = 0; j < 3; j++) {

                    /* X Y Z */
                    size_t index = fInd[i][j];
                    pos_tmp.emplace_back(float2fixed(vPos[index
                        ][0]));
                    pos_tmp.emplace_back(float2fixed(vPos[index
                        ][1]));
```

```
                pos_tmp.emplace_back(float2fixed(vPos[index
                    ][2]));
    if (argc > 1)
    std::cout << "( " << std::dec << vPos[index][0]
        << " " << vPos[index][1]
        << " " << vPos[index][2] << " )"  << std::endl;

                /* R G B */
                col_tmp.emplace_back(vColor[index][0]);
                col_tmp.emplace_back(vColor[index][1]);
                col_tmp.emplace_back(vColor[index][2]);

    if (argc > 1)
    std::cout << "RGB( " << std::dec << (unsigned int)vColor[
        index][0]
        << " " << (unsigned int)vColor[index][1]
        << " " << (unsigned int)vColor[index][2] << " )"  <<
            std::endl;
        }

        /* NX NY NZ */
        pos_tmp.emplace_back(float2fixed(fNormal[i][0]));
        pos_tmp.emplace_back(float2fixed(fNormal[i][1]));
        pos_tmp.emplace_back(float2fixed(fNormal[i][2]));
if (argc > 1) {
std::cout << "NORMAL( " << std::dec << fNormal[i][0]
    << " " << fNormal[i][1]
    << " " << fNormal[i][2] << " )"  << std::endl;
std::cout << std::endl;
}

        data_out.emplace_back(pos_tmp);
        color_out.emplace_back(col_tmp);
}

std::cout << std::dec << data_out.size() << " triangles loaded
    in total" << std::endl;

std::ofstream file_out("data.binary", std::ios_base::out | std
    ::ios_base::binary);

int tri = data_out.size();
file_out.write((char*)&tri, sizeof(int));
```

```
    /* output file layout:
     * X|Y|Z|R|G|B—X|Y|Z|R|G|B—X|Y|Z|R|G|B—NX|NY|NZ for each
        triangle
     */
    for (int i = 0; i < data_out.size(); i++) {
            for (int j = 0; j < 3; j++) {
                    file_out.write((char*)&data_out[i][0 + 3*j],
                       sizeof(fixed_point_t));
                    file_out.write((char*)&data_out[i][1 + 3*j],
                       sizeof(fixed_point_t));
                    file_out.write((char*)&data_out[i][2 + 3*j],
                       sizeof(fixed_point_t));

                    file_out << color_out[i][0 + 3*j];
                    file_out << color_out[i][1 + 3*j];
                    file_out << color_out[i][2 + 3*j];
        file_out << '\0';
            }

            file_out.write((char*)&data_out[i][9], sizeof(
               fixed_point_t));
            file_out.write((char*)&data_out[i][10], sizeof(
               fixed_point_t));
            file_out.write((char*)&data_out[i][11], sizeof(
               fixed_point_t));
    }
    file_out.close();

    std::cout << "output␣file␣generated" << std::endl;

    std::ifstream file_in("data.binary", std::ios_base::in | std::
       ios_base::binary);

    fixed_point_t buffer[9];
    file_in.read((char*)buffer, 9 * sizeof(fixed_point_t));

    for (int i = 3; i < 6; i++)
            std::cout << std::bitset<32>(data_out[0][i]) << "␣";
    for (int i = 3; i < 6; i++)
    std::cout << std::bitset<8>(color_out[0][i]);

    std::cout << std::endl;
std::cout << "|||||SHOULD␣BE␣THE␣SAME|||||" << std::endl;
    for (int i = 5; i < 9; i++)
```

```
                std::cout << std::bitset<32>(buffer[i]) << "␣";
    cout << endl;


        file_in.close();

        return 0;

}
```

6.12 RENDER

Listing 13: render.cpp

```
// g++ -std=c++11 render.cpp -o render
#include <cstddef>
#include <cstdint>
#include <fcntl.h>
#include <fstream>
#include <iostream>
#include <sys/mman.h>
#include <sys/types.h>
#define GLM_ENABLE_EXPERIMENTAL
#include <glm/ext.hpp>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtx/string_cast.hpp>
#include <glm/vec4.hpp>
#include <math.h>
#include <unistd.h>
using namespace std;
// generate MVP
typedef uint32_t fixed_point_t;
#define FIXED_POINT_FRACTIONAL_BITS 16
inline fixed_point_t float2fixed(double input) {
  return (fixed_point_t)(round(input * (1 <<
     FIXED_POINT_FRACTIONAL_BITS)));
}
fixed_point_t *load_matrix() {
  // Projection matrix : 45Âř Field of View, 4:3 ratio, display range
     : 0.1
  // unit <-> 100 units
  //    glm::mat4 Projection = glm::perspective(
  //      glm::radians(45.0f), (float)640 / (float)480, 0.1f, 100.0f);
  // Or, for an ortho camera :
```

```
  glm::mat4 Projection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, 0.0f
      , 100.0f);
  // // In world coordinates
  // Camera matrix
  glm::mat4 View = glm::lookAt(
      glm::vec3(4, 3, 3), // Camera is at (4,3,3), in World Space
      glm::vec3(0, 0, 0), // and looks at the origin
      glm::vec3(0, 1, 0)  // Head is up (set to 0,-1,0 to look upside-
          down)
  );
  // Model matrix : an identity matrix (model will be at the origin)
  glm::mat4 Model = glm::mat4(1.0f);
  //     Model = glm::rotate(Model,1.57,glm::vec3(0,0,1));
  // Our ModelViewProjection : multiplication of our 3 matrices
  glm::mat4 mvp =
      Projection * View *
      Model; // Remember, matrix multiplication is the other way
          around

  glm::vec4 test = mvp * glm::vec4(0, 0, -8, 1);
  cout << "rasterizer: reference = " << glm::to_string(test) << endl;
  fixed_point_t *fixed_matrix = new fixed_point_t[16];
  for (int i = 0; i < 4; i++) {
    for (int j = 0; j < 4; j++) {
      fixed_matrix[4 * i + j] = float2fixed(mvp[j][i]);
    }
  }
  return fixed_matrix;
}

int main() {

  // write vertex into sdram
  int vertex_buffer_offset = 480 * 640 * 8;
  int fd; // manage physical address
  if ((fd = open("/dev/mem", O_RDWR | O_SYNC)) == -1) {
    cerr << "fd open failed" << endl;
    exit(-1);
  }
  std::ifstream file("./data.binary",
                     std::ios::in | std::ios::binary | std::ios::ate);
  if (!file.is_open()) {
    cout << "file open failed";
```

```
    std::abort();
  }
  size_t size = file.tellg();
  char *map_sdram =
      (char *)mmap(0, 64 * 1024 * 1024, PROT_READ | PROT_WRITE,
         MAP_SHARED, fd,
                   0xc0000000); // map the entire sdram
  cout << "binary file size = " << dec << size << endl;
  char *vertex_buffer_base = map_sdram + vertex_buffer_offset;

  file.seekg(0, ios::beg);
  file.read(vertex_buffer_base, size);
  file.close();
  cout << "the entire binary file content is in sdram\n";
  // configure the rasterizer

  char *map_render = (char *)mmap(0, 4096, PROT_READ | PROT_WRITE,
     MAP_SHARED,
                                  fd, 0xff200000); // map the render
                                     to memory

  for (int i = 0; i < 480 * 640; i++) {
    *(int *)(map_sdram + i * 8) = 0;
    *(int *)(map_sdram + i * 8 + 4) = -1;
  }

  *(int *)(map_render + 4) = vertex_buffer_offset;

  fixed_point_t *mat = load_matrix();
  memcpy(map_render + 0x200, mat, 16 * 4);

  __sync_synchronize();
  *(int *)(map_render + 8) = 1;

}
```

6.13 WRITE_TEST_IMAGE

Listing 14: write_test_image.cpp

```
#define cimg_use_jpeg

#include <CImg.h>
#include <cstddef>
#include <string>
```

```
#include <fcntl.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <vector>
#include <iostream>

using namespace cimg_library;
using namespace std;

void loadFrameBuffer(void *buf, const string &file)
{
    CImg<unsigned char> img;
        img.load_jpeg(file.c_str());
    img.resize(640, 480);

    for (int i = 0; i < 480; i++) {
        for (int j = 0; j < 640; j++) {
            auto ptr = (unsigned char *)buf + (i * 640 + j) * 8;
            *(ptr) = img(j, i, 0, 0);
            *(ptr + 1) = img(j, i, 0, 1);
            *(ptr + 2) = img(j, i, 0 ,2);
        }
    }

}

//gcc -g -std=c++17 write_test_image.cpp -lX11 -lpthread -lm -ljpeg -
   lstdc++
int main() {
    int size = 640 * 480 * 8;
    vector<unsigned char> buf(size + 50);

    loadFrameBuffer(buf.data(), "puppy.jpg");

    int fd;
    if((fd = open("/dev/mem", O_RDWR | O_SYNC)) == -1) {
        cerr << "fd␣open␣failed" << endl;
        exit(-1);
    }

    void* map_base;
    map_base = mmap(0, size + 50, PROT_READ | PROT_WRITE, MAP_SHARED,
       fd, 0xc0000000);
```

```
    if(map_base == (void *) -1) {
        cerr << "mmap failed" << endl;
        exit(-1);
    }

    memcpy(map_base, buf.data(), size + 50);
    return 0;
}
```

6.14 SDRAM_CONTROLLER_SIMULATION

Listing 15: sdram_controller.h

```
#pragma once
#include <vector>
#include <list>
#include <iostream>
#include <cstdint>
#include <cstdlib>
#include <ctime>
#include <cstring>

using namespace std;

#define MEMORY_LATENCY 10
#define MAX_QUEUE_LENGTH 8

template<typename Value>
class SDRAMController {
private:
    struct ReadRequest {
        uint32_t port;
        uint32_t address;
        int targetTick;
    };

    struct WriteRequest {
        uint32_t address;
        Value value;
        int targetTick;
    };

    list<ReadRequest> readRequests;
    list<WriteRequest> writeRequests;
```

```
    int tickCount = 0;
public:
    vector<Value> memory;
    SDRAMController(uint32_t memSize)
        : memory(memSize / sizeof(Value), 0xcc)
    {
        cout << "Initializting SDRAM model, size = 0x"
             << hex << memSize << endl;
        cout << "SDRAM controller word size = "
             << sizeof(Value) << endl;
        srand(time(NULL));
    }

    void tick(int port, uint32_t addr, bool read, bool write,
              Value *readdata, unsigned char &readvalid, Value *
                 writedata,
              unsigned char &waitrequest)
    {
        tickCount++;

        // handle output
        if (!readRequests.empty()
            && readRequests.front().targetTick <= tickCount
            && readRequests.front().port == port) {

            if (rand() % 2 != 0) {
                Value v = memory[readRequests.front().address / sizeof
                   (Value)];
                memcpy(readdata, &v, sizeof(Value));
                cout << "sdram read address " << hex << readRequests.
                   front().address << " data " << hex << *readdata <<
                   endl;
                readRequests.pop_front();
                readvalid = true;
            } else {
                readvalid = false;
            }
        } else {
            readvalid = false;
        }

        // handle write queue
        if (!writeRequests.empty()
            && writeRequests.front().targetTick <= tickCount) {
```

```
            WriteRequest &req = writeRequests.front();

            cout << "sdram handle write request to addr " << hex <<
                req.address << endl;
            memory[req.address / sizeof(Value)] = req.value;
            writeRequests.pop_front();
        }

        // check of any queue is full
        if (readRequests.size() >= MAX_QUEUE_LENGTH
            || writeRequests.size() >= MAX_QUEUE_LENGTH) {

            cout << "sdram delay" << endl;
            waitrequest = true;
            return;
        } else {
            waitrequest = false;
        }

        // handle input
        if (read) {
            cout << "sdram: read " << dec << addr << endl;
            ReadRequest req;
            req.address = addr;
            req.port = port;
            req.targetTick = tickCount + MEMORY_LATENCY;
            readRequests.push_back(req);
        } else if (write) {
            cout << "sdram: write " << dec << addr <<  " with " << hex
                << *writedata << endl;
            WriteRequest req;
            req.address = addr;
            memcpy(&req.value, writedata, sizeof(Value));
            req.targetTick = tickCount + MEMORY_LATENCY;
            writeRequests.push_back(req);
        } else {
        }

    }
};
```

6.15 VGA_SIMULATION

Listing 16: vgasim.h

```
#pragma once

#include <cstdint>
#include "SDL2/SDL.h"

class VGASimulator {
private:
    SDL_Window *window;
    SDL_Renderer *renderer;
    SDL_Event event;

    int hcount, vcount;
    bool prevClock = 0;
public:
    VGASimulator();

    void poll();

    void tick(bool clk, uint8_t r, uint8_t g, uint8_t b,
            bool hsync, bool vsync);
};
```

Listing 17: vgasim.cpp

```
#include <iostream>
#include "vgasim.h"

using namespace std;

VGASimulator::VGASimulator()
{
    SDL_Init(SDL_INIT_VIDEO);
    SDL_CreateWindowAndRenderer(640, 480, 0, &window, &renderer);

    SDL_SetRenderDrawColor(renderer, 0, 0, 0, 0);
    SDL_RenderClear(renderer);
    SDL_RenderPresent(renderer);

    hcount = vcount = 0;
}

void VGASimulator::poll()
{
    SDL_PollEvent(&event);
```

```
    if (event.type == SDL_QUIT)
        exit(0);
}

void VGASimulator::tick(bool clk, uint8_t r, uint8_t g, uint8_t b,
            bool, bool)
{
        cout << "vga clock: " << clk << endl;
        if (!(clk && !prevClock)) {
        prevClock = clk;
        return;
    }
    prevClock = clk;
    cout << "vgasim:  r " << (int)r << " g " << (int)g << " b " << (
       int)b << endl;
        cout << "vgasim: hcount: " << dec << hcount << endl;
        cout << "vgasim : vcount: " << dec << vcount << endl;
        if (hcount < 640 && vcount < 480) {
        SDL_SetRenderDrawColor(renderer, r, g, b, 255);
        SDL_RenderDrawPoint(renderer, hcount, vcount);
        SDL_RenderPresent(renderer);
    }
    if (hcount < 799)
        hcount ++;
    else {
        hcount = 0;
        if (vcount < 524) {
            vcount ++;
        }
        else {
            vcount = 0;
            SDL_RenderPresent(renderer);
        }
    }
}

//vsim.poll();
```

6.16 RASTERIZER_TEST

Listing 18: rasterizer_test.cpp

```
#include <string.h>
#include <tgmath.h>
#include <verilated.h>
```

```
#include <cassert>
#include <fstream>
#define GLM_ENABLE_EXPERIMENTAL
#include <glm/glm.hpp>
#include <glm/gtx/string_cast.hpp>
#include <glm/vec4.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/ext.hpp>
#include <iostream>
#include "SDL2/SDL.h"
#include "Vrasterizer_unit.h"
#include "sdram_controller.h"
using namespace std;
// using Vvga_module = Vvga_unit;
// Vvga_module* vga;  // Instantiation of module
using Vras = Vrasterizer_unit;
Vras *top;
typedef uint32_t fixed_point_t;
#define FIXED_POINT_FRACTIONAL_BITS 16
void loadFrameBuffer(void *buf, const string &file);

class VGADisplay {
   public:
    VGADisplay(void *_framebuffer) {
        framebuffer = _framebuffer;
        SDL_Init(SDL_INIT_VIDEO);
        SDL_CreateWindowAndRenderer(640, 480, 0, &window, &renderer);

        SDL_SetRenderDrawColor(renderer, 0, 0, 0, 0);
        SDL_RenderClear(renderer);
        SDL_RenderPresent(renderer);
    }

    void poll() {
        SDL_PollEvent(&event);
        if (event.type == SDL_QUIT) exit(0);
    }
    void refresh() {
        SDL_RenderClear(renderer);
        uint8_t *ptr = (uint8_t *)framebuffer;

        for (int y = 0; y < 480; y++) {
            for (int x = 0; x < 640; x++) {
                uint8_t r = *ptr;
```

```
                uint8_t g = *(ptr + 1);
                uint8_t b = *(ptr + 2);

                ptr += 8;

//                if (r || g || b)
//                    printf("refresh r: %d, g: %d, b: %d\n", r, g, b)
    ;
                SDL_SetRenderDrawColor(renderer, r, g, b, 255);
                SDL_RenderDrawPoint(renderer, x, y);
            }
        }
        SDL_RenderPresent(renderer);
    }

    private:
     void *framebuffer;
     SDL_Window *window;
     SDL_Renderer *renderer;
     SDL_Event event;
};

vluint64_t main_time = 0;  // Current simulation time
// This is a 64-bit integer to reduce wrap over issues and
// allow modulus.  You can also use a double, if you wish.

double sc_time_stamp() {  // Called by $time in Verilog
    return main_time;     // converts to double, to match
                          // what SystemC does
}
inline fixed_point_t float2fixed(double input) {
    return (fixed_point_t)(round(input * (1 <<
       FIXED_POINT_FRACTIONAL_BITS)));
}
fixed_point_t *load_matrix() {
    // Projection matrix : 45Âř Field of View, 4:3 ratio, display
       range : 0.1
    // unit <-> 100 units
    //glm::mat4 Projection = glm::perspective(
    //    glm::radians(45.0f), (float)640 / (float)480, 0.1f, 100.0f);
    // Or, for an ortho camera :
    glm::mat4 Projection = glm::ortho(-10.0f,10.0f,-10.0f,10.0f,0.0f
       ,100.0f);
    // // In world coordinates
```

```
    // Camera matrix
    glm::mat4 View = glm::lookAt(
        glm::vec3(4, 3, 3),  // Camera is at (4,3,3), in World Space
        glm::vec3(0, 0, 0),  // and looks at the origin
        glm::vec3(0, 1, 0)   // Head is up (set to 0,-1,0 to look
            upside-down)
    );
    // Model matrix : an identity matrix (model will be at the origin)
    glm::mat4 Model = glm::mat4(1.0f);
    // Our ModelViewProjection : multiplication of our 3 matrices
    glm::mat4 mvp =
        Projection * View *
        Model;  // Remember, matrix multiplication is the other way
            around

    glm::vec4 test = mvp * glm::vec4(0, 0, -8, 1);
    cout << "rasterizer: reference = " << glm::to_string(test) << endl
        ;
    fixed_point_t *fixed_matrix = new fixed_point_t[16];
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            fixed_matrix[4 * i + j] = float2fixed(mvp[j][i]);
        }
    }
    // memcpy(matrix_base,fixed_matrix,16*4);
    return fixed_matrix;
}
int main(int argc, char **argv) {
    Verilated::commandArgs(argc, argv);  // Remember args
    // simulate a 64M sdram block
    SDRAMController<uint32_t> sdramController(64 * 1024 * 1024);
    char *framebuffer_base = (char *)sdramController.memory.data();
    uint32_t vertex_buffer_offset =
        640 * 480 * 8;  // byte addressable,one pixel 8 bytes
    // load vertex in sdram
    std::ifstream file("../ply_loader/data.binary",
                       std::ios::in | std::ios::binary | std::ios::ate
                           );
    if (!file.is_open()) {
        cout << "file open failed";
        std::abort();
    }
    size_t size = file.tellg();
    cout << "binary file size = " << dec << size << endl;
```

```
char *vertex_buffer_base = framebuffer_base + vertex_buffer_offset
    ;
file.seekg(0, ios::beg);
file.read(vertex_buffer_base, size);
file.close();
cout << (void *)framebuffer_base << "\n";
cout << (void *)vertex_buffer_base << "\n";
cout << "the entire binary file content is in sdram\n";
// initialize framebuffer
for (int i = 0; i < 480 * 640; i++) {
    unsigned long *p = (unsigned long *)(framebuffer_base + i * 8)
        ;
    *p = 0xFFFFFFFFUL << 32;
}
VGADisplay *display = new VGADisplay(framebuffer_base);
// Create instance
top = new Vras;
// rasterizer;
top->clock = 0;
top->reset = 1;  // Set some inputs
top->eval();
top->reset = 0;
top->eval();
top->reset = 1;

// configuration

for (int i = 0; i < 3; i++) {
    switch (i) {
        case 0:
            top->writedata =
                0x0;  // frame buffer start from the address 0 of
                    sdram
            break;
        case 1:
            top->writedata = vertex_buffer_offset;
            break;
        case 2:
            top->writedata = 1;  //
            break;
    }
    top->address = 4 * i;
    top->write = 1;
    sdramController.tick(0, top->master_address, top->master_read,
```

```
                            top->master_write, &top->master_readdata,
                            top->master_readdatavalid, &top->
                                master_writedata,
                            top->master_waitrequest);
    sdramController.tick(
        1, top->master_address_2, top->master_read_2, top->
            master_write_2,
        &top->master_readdata_2, top->master_readdatavalid_2,
        &top->master_writedata_2, top->master_waitrequest_2);
    sdramController.tick(
        2, top->master_address_3, top->master_read_3, top->
            master_write_3,
        &top->master_readdata_3, top->master_readdatavalid_3,
        &top->master_writedata_3, top->master_waitrequest_3);
    top->eval();
    top->clock = 1;
    top->eval();
    top->clock = 0;
    top->eval();
}
uint16_t config_MVreg_addr = 0x100;
uint16_t config_MVPreg_addr = 0x200;
uint16_t config_lightingreg_addr = 0x300;
fixed_point_t *matrix_base = load_matrix();
uint32_t lighting[3];
lighting[0]=0x0;
lighting[1]=0x0;
lighting[2]=0x00010000;
for (int i = 0; i < 16 + 16 + 3; i++) {
    // MV address 256->316  60

    if (i >= 0 && i < 16) {
        top->writedata = 1;
        top->address = config_MVreg_addr + 0x4 * i;
    }
    // MVP address 512->572 60
    else if (i >= 16 && i < 32) {
        top->writedata = matrix_base[i - 16];
        cout << "MVP matrix:" << matrix_base[i - 16] << "\n";
        top->address = config_MVPreg_addr + 4 * (i - 16);
    } else {
        // lighting address 768->176  3
        cout<<"i="<<i;
        cout<<lighting[i-32]<<i;
```

```
            top->writedata = lighting[i-32];
            top->address = config_lightingreg_addr + 0x4 * (i - 32);
        }
        top->write = 1;
        sdramController.tick(0, top->master_address, top->master_read,
                             top->master_write, &top->master_readdata,
                             top->master_readdatavalid, &top->
                                 master_writedata,
                             top->master_waitrequest);
        sdramController.tick(
            1, top->master_address_2, top->master_read_2, top->
                master_write_2,
            &top->master_readdata_2, top->master_readdatavalid_2,
            &top->master_writedata_2, top->master_waitrequest_2);
        sdramController.tick(
            2, top->master_address_3, top->master_read_3, top->
                master_write_3,
            &top->master_readdata_3, top->master_readdatavalid_3,
            &top->master_writedata_3, top->master_waitrequest_3);
        top->eval();
        top->clock = 1;
        top->eval();
        top->clock = 0;
        top->eval();
    }
    cout << "configuration␣done!\n";
    // begin rasterization
    for (;;) {
        display->poll();
        //display->refresh();
        cout << "tick\n";
        sdramController.tick(0, top->master_address, top->master_read,
                             top->master_write, &top->master_readdata,
                             top->master_readdatavalid, &top->
                                 master_writedata,
                             top->master_waitrequest);
        sdramController.tick(
            1, top->master_address_2, top->master_read_2, top->
                master_write_2,
            &top->master_readdata_2, top->master_readdatavalid_2,
            &top->master_writedata_2, top->master_waitrequest_2);
        sdramController.tick(
            2, top->master_address_3, top->master_read_3, top->
                master_write_3,
```

```
            &top->master_readdata_3, top->master_readdatavalid_3,
            &top->master_writedata_3, top->master_waitrequest_3);
        top->eval();
        top->clock = 1;
        top->eval();

        top->clock = 0;
        top->eval();

        if (main_time % 2000 == 0)
            display->refresh();
        main_time++;
    }

    // vga->final();
    delete top;
}
```

REFERENCES

- https://solarianprogrammer.com/2013/05/22/opengl-101-matrices-projection-view-model/
- http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/
- https://www.scratchapixel.com/lessons/3d-basic-rendering/ rasterization-practical-implementation/rasterization-stage
- https://codeplea.com/triangular-interpolation