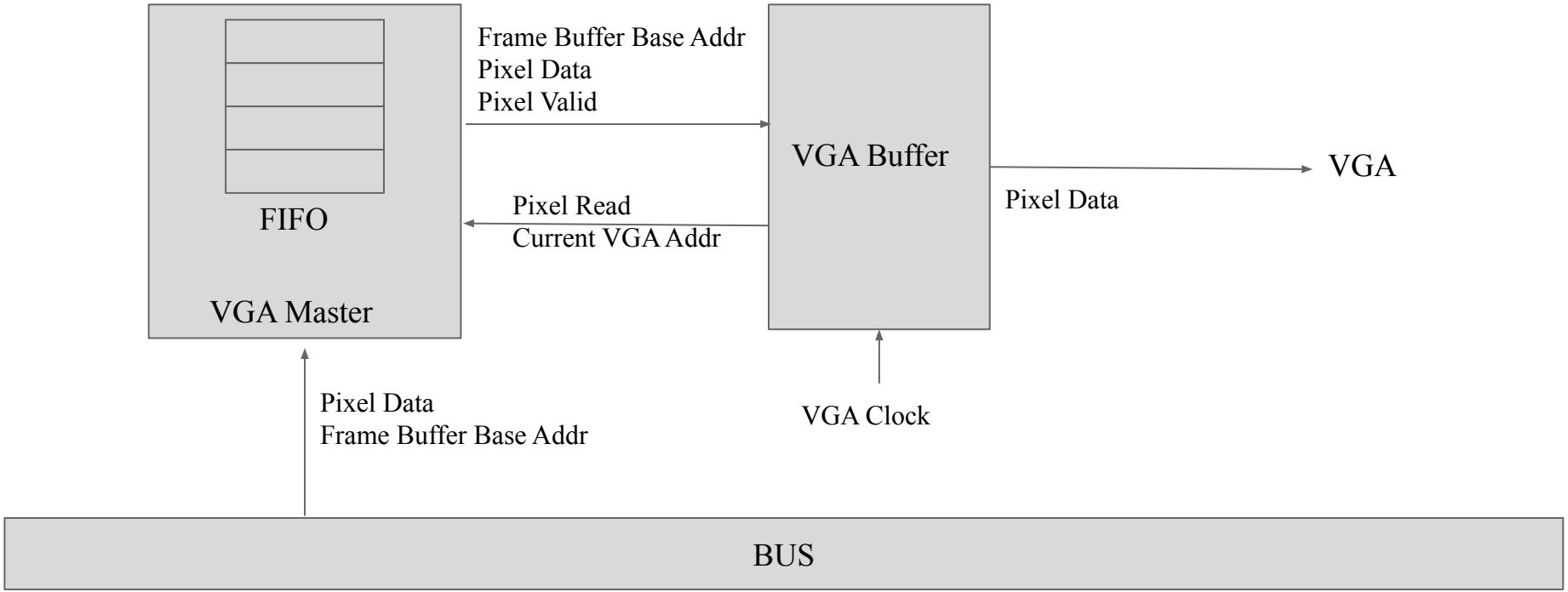
3D Graphics Accelerator

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

System Overview

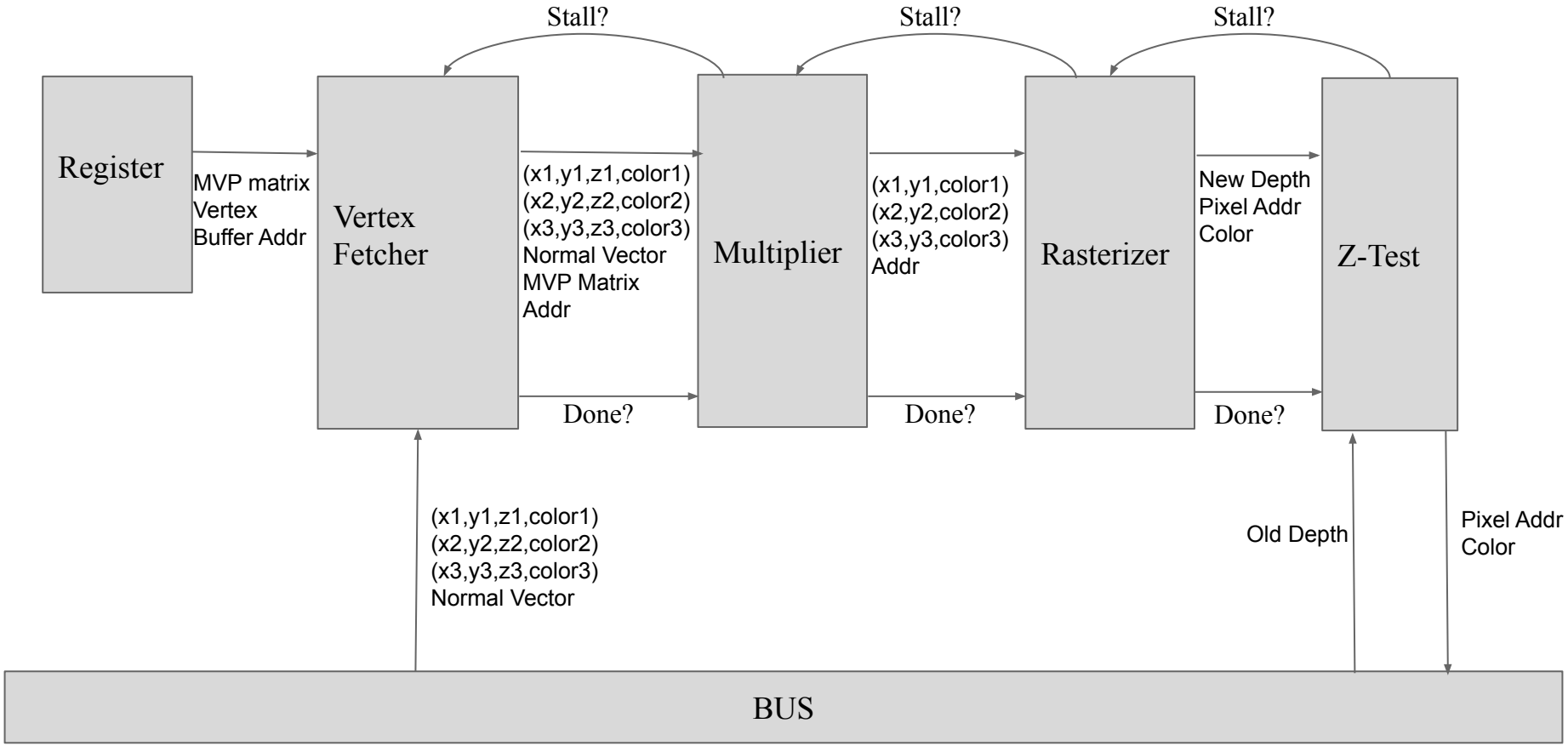
- Software preprocessing data and loading data into board
- Verilator for verification and prototype
- Video display module generating the VGA signals
- Rendering module converting vertex info to 2D image
- Communicating through shared SDRAM
- Pipeline computation and BUS communication

Hardware: VGA Output Module



VGA output module reading from SDRAM

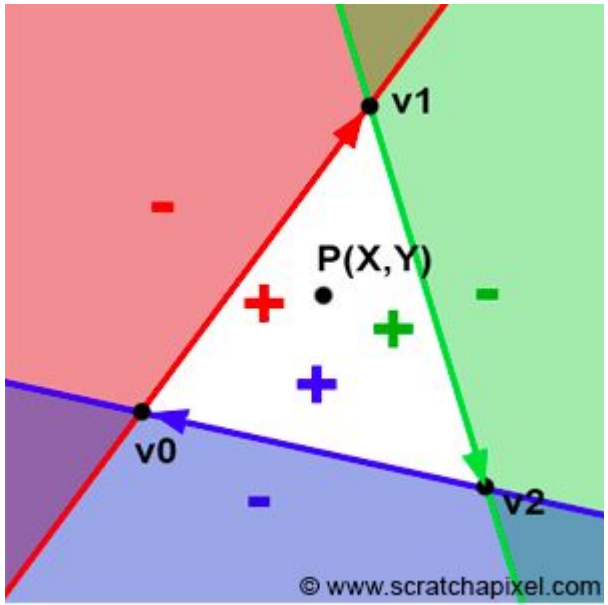
Hardware: Rendering Module



Rasterizing Algorithm

The Edge Function:

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



Color Interpolation

Barycentric Coordinates

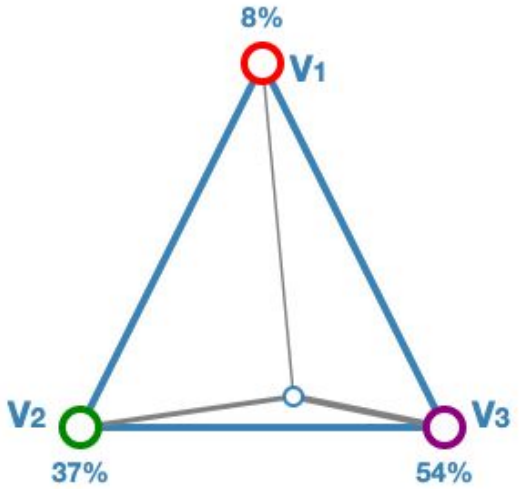
Find weights that balance the following system of equations:

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

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

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

$$W_{v3} = 1 - W_{v1} - W_{v2}$$



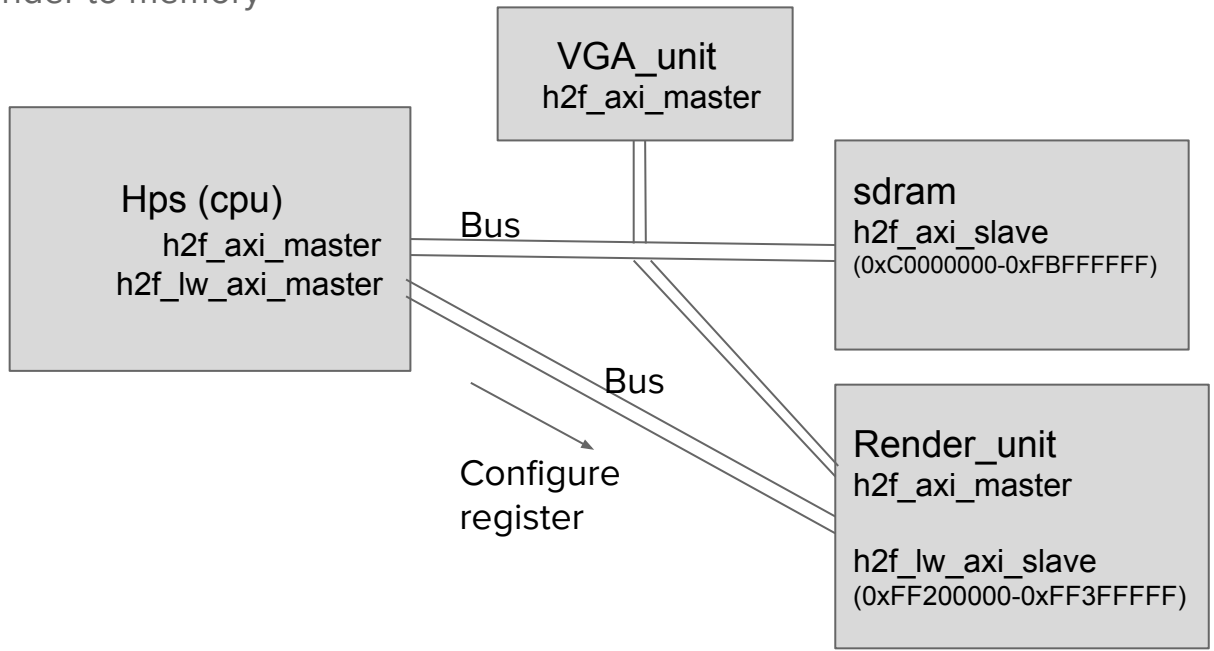
Latency & Pipelining

- The renderer is pipelined to mitigate memory stalls.
- Vertex calculating, rasterizing, z-buffer reading and writing back can be concurrent.
- Division for color interpolation: 12 cycles
- Vector multiplication generally 2-4 cycles
- Dividers & multipliers not pipelined, because memory throughput is the bottleneck

Software Implementation

- Map the physical address of the render device and sdram to virtual address and write memory

```
char*map_sdram=(char*)mmap(0, 64*1024*1024, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0xc0000000); //map the entire sdram
char*map_render=(char*)mmap(0, 4096, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0xff200000); //map the render to memory
```



Generate MVP matrix with GLM

- **Projection matrix : 45° 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
);
```

- **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);
glm::mat4 mvp = Projection * View *Model;
```

Floating point to fixed point

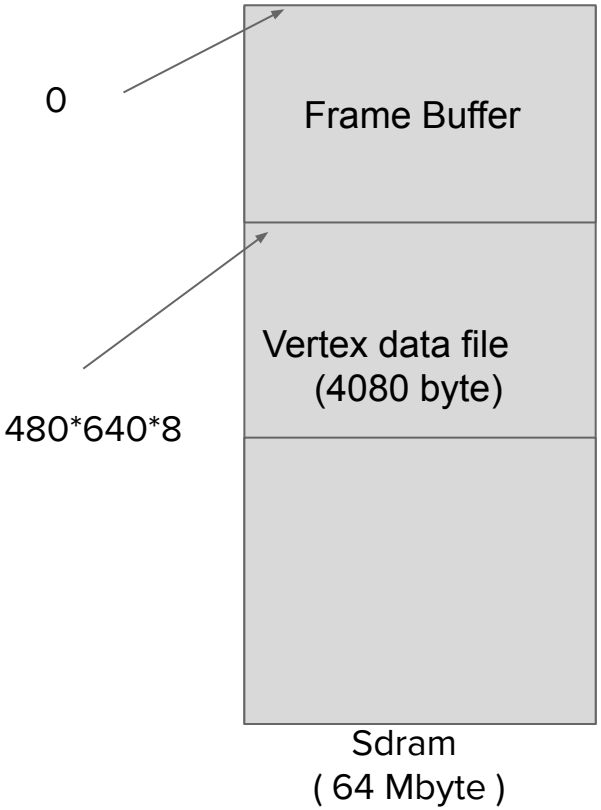
Fractional part 16 bits, integer part 16 bits, 32 bits in total

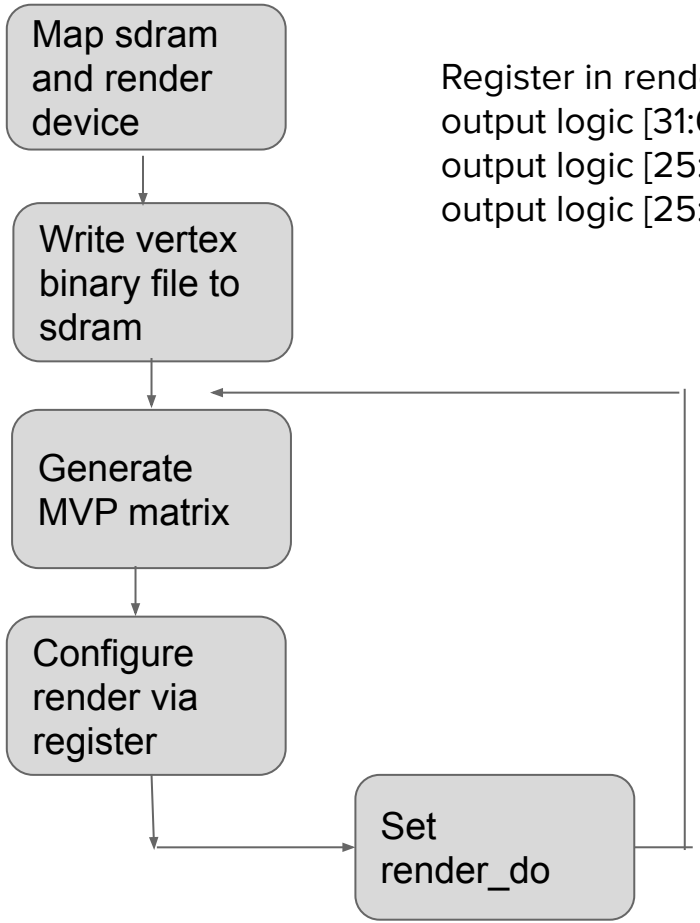
Step 1. Multiply the floating number by 2**16;

Step 2. Round this value to the nearest integer;

Step 3. Assign this value to fixed-point type.

Flow Chart



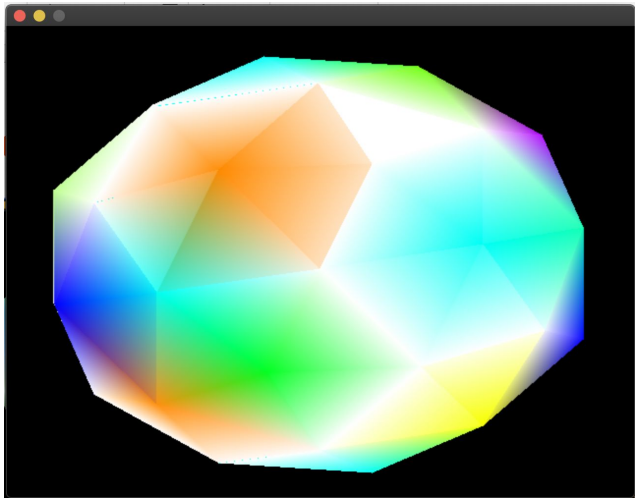


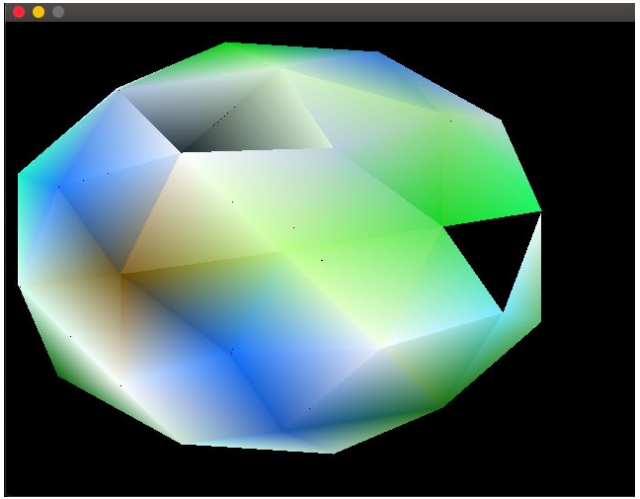
Register in render device:
output logic [31:0] MVP [15:0],
output logic [25:0] frame_buffer_base,
output logic [25:0] vertex_buffer_base,

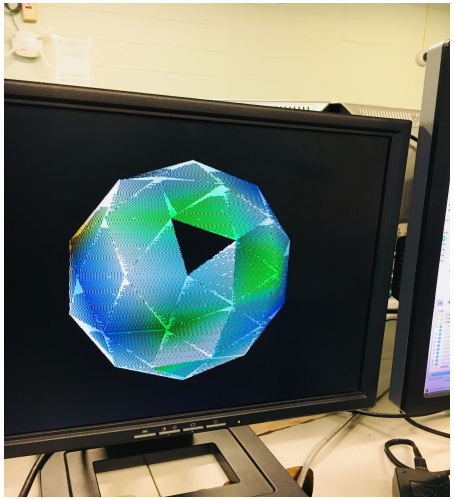
Challenges

- Timing
  - Rasterizer
  - Color interpolation
  - SDRAM configuration
- Pipelining logic

Software Simulation







Lesson Learned

- Better pipeline logic
- Should not use too many combinational logic
- 2 arithmetic operations/cycle