3D Hardware Rasterizer

CSEE: Embedded Systems

Spring 2026

Gianna Belmont, Srika Chagarlamudi,

Sathvik Rajampalli, Shlok Desai, Aarush Agarwal

1. Introduction

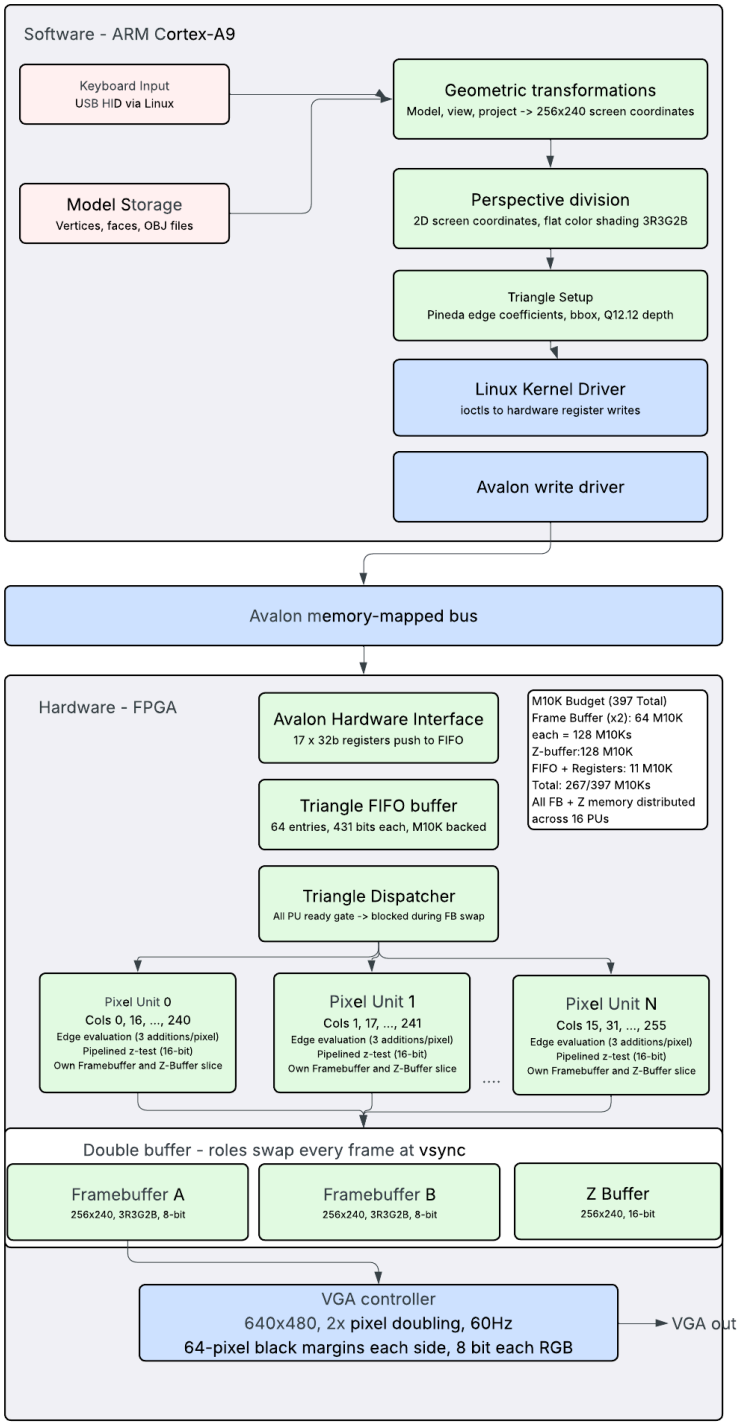
Real-time 3D graphics rendering is one of the most computation-intensive workloads in modern computing, requiring the rapid transformation, projection, and rasterization of thousands of triangles per frame. This project addresses that challenge by designing and implementing a hardware-accelerated 3D triangle rasterizer on the DE1-SoC FPGA platform, capable of rendering rotating 3D geometry on a VGA monitor at interactive frame rates. The goal is to demonstrate that a carefully designed hardware pipeline can achieve real-time rendering performance within the tight resource constraints of an embedded FPGA system.

The system is partitioned between two tasks. Computation that occurs once per triangle belongs in software, while computation that occurs once per pixel belongs in dedicated hardware. The ARM processor handles user input and performs the 3D mathematics such as applying rotation and projection matrices to transform vertex coordinates from 3D world space into 2D screen coordinates. It also performs triangle setup, generating bounding boxes, edge-function coefficients, initial edge values, depth interpolation terms, and color fields. These triangle packets are then passed to the FPGA over the Avalon memory-mapped bus. The hardware rasterizer takes over from there, determining which pixels fall inside each triangle, interpolating depth, performing a Z-buffer test, and writing visible pixels into a framebuffer that drives the VGA display.

The core rasterization algorithm is the Pineda edge equation method, introduced in the SIGGRAPH paper “A Parallel Algorithm for Polygon Rasterization.” For each triangle, a linear edge function is defined for each of its three edges. A pixel is interior to the triangle if and only if it lies on the positive side of all three edge functions simultaneously. Since these functions are linear, their values change by a constant amount with each unit step in the x or y direction. This means that after software computes the initial edge values, the hardware can walk through pixels using only additions and sign checks. Depth interpolation works the same way: because the triangle is planar, depth changes by a constant amount in x and y, so the hardware updates depth with additions rather than recomputing a full equation per pixel. This makes the algorithm well-suited to hardware implementation, where adders are cheap and dividers are expensive.

This approach represents a significant architectural improvement over prior scanline-based implementations. The Pineda method eliminates per-pixel division entirely, replacing that pipeline with a 16-pixel-unit systolic rasterizer whose inner loop consists of adders, sign-bit checks, and Z comparisons. In the final design, the FPGA datapath uses no DSP blocks and is not limited by arithmetic complexity. Instead, the main hardware resource cost is the distributed framebuffer and Z-buffer storage, while the measured end-to-end performance is dominated by software triangle setup and MMIO packet submission.

2. System Block Diagram



Top Level System Diagram

The top level system diagram shows the system divided into a software component running on the ARM Cortex-A9 and a hardware component implemented on the FPGA, communicating over the Avalon memory-mapped bus.

On the software side, keyboard input is read via the Linux USB HID driver and used to update rotation angles each frame. The 3D model is stored in memory as a list of vertices and faces. Each frame, the ARM processor applies model, view, and projection transforms to every vertex, producing 2D screen coordinates and a depth value per vertex. Perspective division is performed in software to complete the projection. For each triangle, software computes a flat shade color and runs triangle setup, computing the 3 edge function coefficients per edge, the bounding box, initial edge function values at the bounding box corner, and depth step values. This ensures hardware will only perform per-pixel work.

The Avalon write driver submits triangle packets by writing 17 consecutive 32-bit words to the FIFO write port, then strobes the commit register. Submission is interrupt driven to avoid busy waiting.

The triangle FIFO buffers up to 64 entries in inferred M10K BRAM. The triangle dispatcher pops one entry and broadcasts it to all 16 pixel units simultaneously, but only after all 16 units signal idle, which ensures no unit receives a new triangle while still finishing the previous one. Each pixel unit owns its own interleaved columns. Each PU handles columns N, N+16, N+32, ..., across the 256 pixel width. Each unit uses its own M10K slices for their own double-buffered framebuffer and a single buffered Z-buffer, eliminating memory contention.

The two frame buffers are 256x240, 3R3G2B, and use 4 M10K blocks each (for a total of 128 M10Ks for 2 framebuffers for 16 pixel units each). One is always read by the VGA and the other is written by the pixel units. On frame completion, their roles swap. The VGA controller reads the front buffer and generates a 512x480 by doubling each pixel in both dimensions at 60Hz. To fill up the 640x480 dimensions, the VGA controller leaves 64 black pixels on each side.

3. Algorithms

3.1 Software

3.1.1 Overview

The ARM Cortex-A9 processor on the DE1-SoC runs a C program that performs all per-frame and per-triangle computation. Its job is to take a 3D model stored as vertices and triangle faces, transform them into 2D screen coordinates, compute lighting and color for each face, set up the Pineda edge equation parameters for each triangle, and submit the resulting triangle packets to the FPGA hardware over the Avalon memory-mapped bus.

All floating-point computation happens in software. The hardware receives only fixed-point integer values and performs only addition and comparison in the pixel-processing loop. This division of labor puts expensive math (matrix multiplies, trigonometric functions, division) on the ARM, and puts the high-volume per-pixel work (edge function evaluation, depth testing) on the FPGA where it can be parallelized across 16 pixel units.

3.1.2 Pipeline Flow

The software pipeline executes the following steps in order, once per frame:

1. **Read user input.** Read keyboard state via the Linux input subsystem. Arrow keys adjust the object's rotation angles by a fixed increment per frame (e.g., 2 degrees). Up/down arrows rotate around the X axis, left/right around the Y axis.
2. **Build transformation matrix.** Construct the model-view-projection (MVP) matrix by composing rotation, view (camera translation), and perspective projection matrices. This single 4×4 matrix encapsulates all geometric transformations.
3. **Transform and project vertices.** Multiply each model vertex by the MVP matrix to produce clip coordinates. Perform perspective division (divide by w) to get normalized device coordinates. Apply the viewport transform to convert from NDC [-1, 1] to screen coordinates. The result is a 2D screen position and a depth value for each vertex.
4. **Per-face lighting.** For each triangle, compute the face normal from the cross product of two edge vectors. Transform the normal to world space. Take the dot product with a fixed light direction vector to determine brightness. Encode the shaded color as an 8-bit RGB332 value.
5. **Triangle setup.** For each triangle, compute the Pineda edge equation coefficients (a, b for each of the three edges), the bounding box clamped to screen boundaries, the initial edge

function values at the bounding box origin, and the depth interpolation step values. Convert all values from floating-point to Q12.12 fixed-point. Pack everything into a triangle packet struct.

6. **Submit to hardware FIFO.** Write the triangle packet fields to the Avalon register interface. The final write to the COMMIT register pushes the assembled packet into the hardware FIFO. Check the FIFO status register before each submission to avoid overflow.

3.1.3 Data Structures

The following C structs define the data types used throughout the software pipeline.

3D vertex (vec3f_t):

```
C/C++

typedef struct {

    float x, y, z;

} vec3f_t;
```

4D homogeneous vector (vec4f_t):

```
C/C++

typedef struct {

    float x, y, z, w;

} vec4f_t;
```

4×4 transformation matrix (mat4f_t):

```
C/C++

typedef struct {

    float m[4][4];

} mat4f_t;
```

Triangle face (face_t):

```C/C++
typedef struct {
    int v[3];  // indices into vertex array
} face_t;
```

Screen-space vertex (screen_vertex_t):

```C/C++
typedef struct {
    float sx, sy;  // screen coords [0,319] x [0,239]
    float sz;      // normalized depth [0, 1]
} screen_vertex_t;
```

Triangle packet (to FIFO) (triangle_packet_t):

```C/C++
typedef struct {
    int32_t  a0, b0, c0;              /* edge 0 coefficients (c0 unused by HW) */
    int32_t  a1, b1, c1;              /* edge 1 coefficients (c1 unused by HW) */
    int32_t  a2, b2, c2;              /* edge 2 coefficients (c2 unused by HW) */
    int32_t  e0_init, e1_init, e2_init;  /* initial edge values at bbox origin */
    int32_t  z_origin, z_step_x, z_step_y; /* depth interpolation Q12.12 */
    uint32_t bbox_packed;             /* [31:24]=ymax [23:16]=ymin [15:8]=xmax [7:0]=xmin */
    uint32_t flags_color;             /* bit [8]=front_facing, bits [7:0]=RGB332 color */
} triangle_packet_t;
```

c0, c1, c2 are accepted by the hardware staging registers but dropped before the FIFO. The ex_init fields already encode the constant term evaluated at the bounding box origin, so c is never needed downstream.

3.1.4 Fixed-Point Format

All values sent to hardware use Q12.12 signed fixed-point representation stored in 32-bit integers.

Property	Value
Format	Q12.12 (1 sign + 11 integer + 12 fractional)
Storage	int32_t (32 bits)
Range	−524288.0 to +524287.9998
Precision	1/4096 ≈ 0.000244
Conversion	fixed = (int32_t)(float_val * 4096.0 + 0.5)

Conversion function:

```
C/C++

int32_t to_fixed(float v) {

    return (int32_t)(v * 4096.0f + (v >= 0 ? 0.5f : -0.5f));

}
```

3.1.5 Matrix Transforms

The MVP matrix is constructed by multiplying: MVP = Projection × View × Rotation_Z × Rotation_Y × Rotation_X.

Rotation matrices are standard 4×4 rotations. For example, rotation around Y by angle θ uses cosθ and sinθ in the [0,0], [0,2], [2,0], [2,2] positions. The view matrix is a translation along negative Z by the camera distance (default 4.0). The projection matrix uses a 60° field of view, aspect ratio 256/240 ≈ 1.067, near plane 0.1, and far plane 100.0.

3.1.10 Vertex Projection

For each vertex: (1) multiply by MVP to get clip coordinates, (2) perspective divide: ndc_x = clip.x/clip.w, ndc_y = clip.y/clip.w, ndc_z = clip.z/clip.w, (3) viewport transform:

```
C/C++

screen_x = (ndc_x + 1.0) * 0.5 * 320;

screen_y = (1.0 - ndc_y) * 0.5 * 240;   // Y flipped

screen_z = clamp((ndc_z + 1.0) * 0.5, 0.0, 1.0);
```

3.1.7 Per-Face Lighting

For each triangle face: (1) compute face normal from cross product of two edge vectors, normalized, (2) transform normal to world space using model matrix (w=0), (3) diffuse intensity: ndotl = max(0, dot(world_normal, light_dir)), (4) final intensity with ambient 0.25: intensity = clamp(0.25 + 0.75 * ndotl, 0, 1), (5) encode as RGB332:

```
C/C++

R = clamp(base_r * intensity * 7 + 0.5, 0, 7);   // 3 bits

G = clamp(base_g * intensity * 7 + 0.5, 0, 7);   // 3 bits

B = clamp(base_b * intensity * 3 + 0.5, 0, 3);   // 2 bits

color = (R << 5) | (G << 2) | B;
```

Light direction: normalize(0.4, 0.7, 0.5). Base color: R=0.2, G=0.7, B=1.0 (teal/cyan).

3.1.8 Triangle Setup - Pineda Edge Equations

This function converts three screen-space vertices into the precomputed coefficients that the FPGA uses. Edge equation coefficients for each edge E(x,y) = a*x + b*y + c:

```
C/C++

Edge 0 (v1->v2):  a0 = v1.sy - v2.sy    b0 = v2.sx - v1.sx

Edge 1 (v2->v0):  a1 = v2.sy - v0.sy    b1 = v0.sx - v2.sx

Edge 2 (v0->v1):  a2 = v0.sy - v1.sy    b2 = v1.sx - v0.sx
```

The a coefficient is the x-step (hardware adds a per pixel right). The b coefficient is the y-step (hardware adds b per pixel down). No multiplication or division in the inner loop.

Winding normalization. The signed area is computed as E0 evaluated at v0: area = a0 · v0.sx + b0 · v0.sy + c0.

- If $|\text{area}| \leq 10^{-6}$: degenerate triangle (collinear vertices), return −1 and skip.
- If area < 0: winding is inconsistent. Negate all six coefficients (a0, b0, c0, a1, b1, c1, a2, b2, c2) to normalize to a consistent orientation. The triangle is still submitted with front_facing = 1. True occlusion is resolved by the depth buffer in hardware — back-facing surfaces of solid objects are occluded by closer front-facing surfaces and discarded at the Z-test stage.

Bounding box: Clamped to screen: bbox_xmin = max(0, floor(min(v0.sx, v1.sx, v2.sx))), bbox_xmax = min(319, ceil(max(...))), etc.

```
C/C++

bbox_xmin = max(0,   floor(min(v0.sx, v1.sx, v2.sx)))

bbox_xmax = min(255, ceil (max(v0.sx, v1.sx, v2.sx)))
```

After clamping, bbox_xmin is snapped down to the nearest multiple of 16:

```
C/C++

bbox_xmin = bbox_xmin & ~15;
```

Initial edge values: Evaluated at the pixel centre of the bounding box origin (px, py) = (bbox_xmin + 0.5, bbox_ymin + 0.5):

```
C/C++

e0_init = a0*px + b0*py + c0

e1_init = a1*px + b1*py + c1

e2_init = a2*px + b2*py + c2
```

Depth interpolation: Vertex depths are scaled from [0, 1] to [0, 65535] before interpolation: zs_i = v_i.sz × 65535. The depth plane through the three vertices is then:

```
C/C++

z_step_x = (a0 * zs0 + a1 * zs1 + a2 * zs2) / area

z_step_y = (b0 * zs0 + b1 * zs1 + b2 * zs2) / area

z_origin = (e0_init * zs0 + e1_init * zs1 + e2_init * zs2) / area
```

All values are converted to Q12.12 fixed-point before packing into the packet.

3.1.9 Complete Function List

Function	Input	Output	Description
keyboard_init	—	status	Open USB HID device via libusb
keyboard_close	—	—	Release USB interface and handle
poll_keys	USB HID report	g_keys[], g_key_pressed[]	Build held/edge tables from scan codes
rotation_x/y/z	angle (degrees)	mat4_t	4×4 rotation matrix
translation	tx, ty, tz	mat4_t	4×4 translation matrix
perspective	fov, aspect, near, far	mat4_t	Perspective projection matrix
mat4_mul	mat4_t a, b	mat4_t	Matrix product a × b
mat4_mul_vec4	mat4_t, vec4_t	vec4_t	Matrix-vector product
project_vertices	model, MVP	screen_vertex_t[]	Project all vertices to screen space
shade_face	normal, light, color	uint8_t	RGB332 color via N·L lighting
to_fixed	float	int32_t	Float to Q12.12 fixed-point
setup_triangle	3 screen verts, color	triangle_packet_t	Edge coefficients, bbox, depth pack
submit_triangle	fd, packet	(writes to HW)	Write 17-word packet via mmap or ioctl

3.2 Hardware Rasterization

3.2.1 Rasterizer Overview

The hardware rasterizer implements a 16-lane systolic triangle rasterization pipeline. Each triangle is prepared in software and submitted to hardware as a packet containing its bounding box, edge-function increments, initial edge values, interpolated depth values, and color. The hardware then walks the triangle bounding box, evaluates edge functions at each pixel, performs a Z-depth test, and writes visible pixels into the frame buffer.

The design is split into four main hardware blocks:

1. A triangle FIFO stores incoming triangle packets from the software side
2. A triangle dispatcher pops one triangle at a time and broadcasts its constants to the pixel units
3. A 16-pixel-unit systolic chain rasterizes the triangle
4. A VGA frame buffer reader scans the completed front buffer to the VGA output

This design distributes the frame buffer across 16 pixel units. Each pixel unit owns one column bank of the 256x240 internal render target. Specifically, pixel unit PU_ID owns all columns whose lower 4 bits equal PU_ID. For example, PU0 owns columns 0, 16, 32, …, 240, while PU1 owns columns 1, 17, 33, …, 241. This lets all 16 pixel units work on different columns of the same triangle in parallel. The internal frame buffer is 256x240 and is displayed through VGA by scaling each pixel 2x. This produces a 512x480 image centered horizontally inside the 640x480 VGA output region

3.2.2 Edge-function Rasterization

The rasterizer uses the parallel algorithm from Pineda. Each directed edge of a triangle defines a linear function E(x,y) that is positive on one side of the edge and negative on the other. For an edge from vertex (x_0,y_0) to (x_1,y_1):

$$E(x, y) = a \cdot x + b \cdot y + c$$

Where $a = (y_1 - y_0)$, $b = (x_0 - x_1)$, and c is the constant that makes $E(x_0,y_0) = 0$. A pixel lies inside the triangle if and only if all three edge functions E_0, E_1, E_2 are all non-negative. Software guarantees this by ordering triangle vertices consistently before submission.

Because E is linear, incrementing x by one adds a to the edge value and incrementing y by one adds b. Every per-pixel update is therefore a signed add and a sign check. Depth z is updated the same way. One add per pixel, using z_step_x when walking in x and z_step_y when stepping to the next row. As a result, the per-pixel datapath does not need multipliers. Each pixel unit only needs signed adders, sign-bit checks, and a Z comparison.

In the current RTL, the inside test is implemented by checking the sign bits of the three edge accumulators:

```
None

inside_now = (state == S_ACTIVE)

         && (e0[31] == 1'b0)

         && (e1[31] == 1'b0)

         && (e2[31] == 1'b0);
```

This is cheaper than synthesizing three full signed comparators.

3.2.3 Triangle Packets and Dispatch

Software performs the per-triangle setup before submitting work to the rasterizer. Each submitted triangle packet contains:

```
None

       bbox_xmin, bbox_xmax

       bbox_ymin, bbox_ymax

       a0, b0, a1, b1, a2, b2

       e0_init, e1_init, e2_init

       z_at_origin

       z_step_x, z_step_y
```

```
color
```

The a values are the edge-function increments for moving one pixel in x, while the b values are the increments for moving one pixel in y. The initial edge values and initial depth correspond to the first pixel of the triangle bounding box.

Triangle packets are stored in a hardware FIFO. The dispatcher only issues a new triangle when all 16 pixel units report ready. This guarantees that the previous triangle has fully drained through the systolic chain before the next triangle begins.

When a packet is dispatched, the triangle constants are broadcast to all pixel units, but only PU0 receives the initial seed directly. The rest of the pixel units receive their seeds from the previous pixel unit in the chain.

3.2.4 Systolic Seed Handoff

The systolic chain is used to stagger the start of each pixel unit by one column. PU0 starts at the first column of the aligned bounding box. On the same clock edge that it accepts its seed, it forwards a new seed to PU1:

```
None
```

```
seed_e0_out = seed_e0_in + a0

seed_e1_out = seed_e1_in + a1

seed_e2_out = seed_e2_in + a2

seed_z_out = seed_z_in + z_step_x
```

This gives the next pixel unit the correct edge and depth values for the column one pixel to the right. The same process repeats down the chain until all 16 pixel units have been seeded.

This structure avoids a long combinational adder chain. Each pixel unit performs only one registered +a step before passing the seed to the next unit. Therefore, the chain has one 32-bit add between registers per stage, rather than a 16-stage combinational dependency.

The dispatcher assumes that the bounding box x-min value is aligned to a 16-pixel boundary. With that invariant, PU_ID always matches the lower 4 bits of the starting column assigned to that pixel unit.

3.2.5 Pixel Unit Algorithm

Each pixel unit owns every 16th column of the screen. After receiving its initial seed, it walks down one column of the triangle bounding box. When it reaches the last row, it jumps 16 columns to the right and repeats. This continues until the next 16-column jump would pass the triangle's bbox_xmax. The first column is seeded by the systolic chain. Later columns are generated locally from saved column-top edge values plus 16*a. This is necessary because the live edge and depth accumulators have already been advanced by +b while walking down the previous column.

Each pixel unit has four main states:

```
None

S_INIT_CLEAR  clears local memories after reset

S_IDLE        waits for a triangle seed or frame-clear request

S_ACTIVE      rasterizes the current triangle

S_CLEAR       clears the Z-buffer and current back buffer between frames
```

The pixel unit behavior can be summarized as:

```
None

On reset:

   clear local memories, then enter IDLE

In IDLE:
```

```
   if z_clear_start:
       enter CLEAR
   else if seed_valid_in:
       latch triangle constants
       latch starting edge/depth seed
       save column-top edge/depth values
       forward seed + a to the next PU
       start at row_base
       enter ACTIVE
In ACTIVE:
   form local address from {col_base[7:4], cur_row}
   test inside = sign(e0) == 0 && sign(e1) == 0 && sign(e2) == 0
   queue the candidate pixel for the next-cycle Z-test
   issue a synchronous Z-buffer read
   advance one row:
       e0 += b0
       e1 += b1
       e2 += b2
       z  += z_step_y
```

```
        cur_row++

    if cur_row reached last_row:

        if col_base + 16 would pass last_col:

            return to IDLE

        else:

            jump 16 columns right

            restore e0/e1/e2/z from column-top + 16*x_step

            reset cur_row to row_base

            keep rasterizing

One cycle after each queued candidate:

    if candidate was inside and q_z <= z_rd:

        write q_z into the local Z-buffer

        write q_color into the selected back framebuffer
```

This keeps the per-pixel loop simple: one candidate pixel is queued per active cycle, the edge and depth values advance by +b, and the Z-test/write happens one cycle later through the M10K-backed read pipeline. The design can test one candidate per active PU per cycle after the pipeline fills.

3.2.6 Local Addressing and Column Banking

Each pixel unit contains its own local memories. A local memory address is formed as:

addr_now = {col_base[7:4], cur_row}

The lower 4 bits of the screen x-coordinate are not stored in the local address, because they are implied by the pixel unit ID. Therefore:

```
None

screen x[3:0]  selects the pixel unit


screen x[7:4]  selects the local column bank inside that pixel unit


screen y[7:0]  selects the row
```

This gives a 12-bit local address:

4 bits column group + 8 bits row = 12 bits

Each pixel unit therefore has 4096 local addresses. The visible framebuffer uses rows 0 through 239, so the 4096-address space is slightly larger than the 256x240 active image, but the power-of-two addressing keeps the hardware simple.

3.2.7 Z-buffer and Depth Test

Each pixel unit has a local 16-bit Z-buffer stored in M10K memory. The internal depth value is maintained as a signed 32-bit fixed-point value, and the stored Z-buffer value is a 16-bit slice:

q_z <= z[Z_MSB:Z_LSB];

In the current configuration, the slice is:

z[27:12]

The Z-buffer is initialized to 16'hFFFF, which represents the farthest depth. The rasterizer uses a smaller-depth-wins convention:

z_pass = q_valid && q_inside && (q_z <= z_rd)

The <= comparison allows equal-depth pixels to write. If two pixels have the same depth, the later one wins. This is visually harmless for coincident depths.

Because M10K reads are synchronous, the Z-test is pipelined over two cycles:

```
None

Cycle N:

   issue read at addr_now

   queue candidate pixel information into q_* registers

Cycle N+1:

   z_rd contains the old stored depth

   q_* contains the candidate pixel

   compare q_z against z_rd

   conditionally write new depth and color
```

The queued candidate registers include q_valid, q_inside, q_addr, q_col, q_row, q_color, and q_z.

This lets the pixel unit keep walking one new candidate pixel per cycle while the previous candidate is being depth-tested. Since the design uses separate read and write behavior for the M10K-backed memory, the pipeline can sustain one candidate per active pixel unit per cycle after filling.

3.2.8 Color Frame buffers and VGA Output

Each pixel unit contains two local 8-bit color framebuffers fb_a_mem and fb_b_mem. The design uses double buffering. While the rasterizer writes into one framebuffer, VGA reads from the other. The signal fb_write_sel chooses the current render target:

fb_write_sel = 0: rasterizer writes FB_A

fb_write_sel = 1: rasterizer writes FB_B

The VGA side reads the opposite buffer:

vga_r_buf_sel = ~fb_write_sel

The VGA reader computes the internal framebuffer coordinate from the current VGA scan position. The 256x240 internal image is scaled 2x to 512x480 and horizontally centered in the 640x480 display. Pixels outside this centered region are black.

For each visible VGA pixel:

fb_x = (hcount - 64) / 2

fb_y = vcount / 2

Then:

vga_r_addr = {fb_x[7:4], fb_y}

PU select = fb_x[3:0]

All 16 pixel units receive the same vga_r_addr. Each one returns the color stored at that local address. The lower nibble of fb_x selects which pixel unit's returned data is sent to the VGA output. Distributing framebuffer storage across the pixel units eliminates shared framebuffer arbitration and allows all rasterizers to perform concurrent writes without memory contention. Because the framebuffer BRAM read path has one cycle latency, the VGA module delays the pixel unit select signal by one clock cycle so that the selected PU output aligns with the returned framebuffer data.

The framebuffer stores color in RGB332 format:

bits [7:5] = red

bits [4:2] = green

bits [1:0] = blue

The VGA module expands this 8-bit color into 8-bit VGA red, green, and blue outputs using bit replication.

3.2.9 Frame Swap and Clear

At the end of a rendered frame, software requests a present operation. The hardware waits until it is safe to swap the front and back buffers. The swap logic waits for:

1. A completed VGA frame
2. An empty triangle FIFO
3. All pixel units idle

Once these conditions are true, the hardware toggles fb_write_sel. This makes the previously rendered back buffer become the new front buffer for VGA scanout. This guarantees that the VGA never scans a partially rendered framebuffer and prevents visible tearing artifacts while displaying.

After the swap, each pixel unit receives a one-cycle z_clear_start pulse. The pixel units then clear:

z_mem -> 16'hFFFF

back color framebuffer -> 8'h00

Only the new back buffer is cleared. The front buffer is not cleared because VGA is actively scanning it out.

During the swap and clear operation, the dispatcher is blocked from issuing new triangles. This guarantees no pixel unit begins rasterizing into a framebuffer while its local memories are still being cleared. This prevents new rasterization work from starting while the pixel units are clearing their local memories. Each PU clears 4096 local addresses, but all 16 PUs clear in parallel, so the wall-clock clear time is about 4096 cycles. The top-level uses a 4100-cycle clear window, or about 82 us at 50 MHz. Since the clear latency is much smaller than a 16.67 ms VGA period, the clear pass does not meaningfully reduce rendering throughput or display

4. Resource Allocation

The DE1-SoC uses a Cyclone V 5CSEMA5F31C6N FPGA with approximately 32K ALMs and 3,970 Kbits of M10K block RAM. The design runs on the 50 MHz FPGA fabric and drives a standard 640x480 VGA output at 60 Hz.

The rasterizer renders into a 256x240 internal RGB332 framebuffer. The VGA controller scales this image 2x to 512x480 and centers it in the 640x480 display, leaving 64-pixel black bars on the left and right. The current architecture uses 16 systolic pixel units. Each PU owns one column bank of the screen: PU i stores and rasterizes columns whose lower 4 bits equal i.

The memory is distributed across the pixel units. Each PU contains two local color framebuffers and one local Z-buffer:

fb_a_mem: 4096 x 8-bit

fb_b_mem: 4096 x 8-bit

z_mem: 4096 x 16-bit

The local address is 12 bits:

local_addr = {x[7:4], y[7:0]}

The lower 4 bits of x select which PU owns the pixel. The upper 4 bits of x select the local column index inside that PU, and y[7:0] selects the row. Each PU actively uses 16 x 240 = 3,840 entries, rounded up to 4,096 entries for simple power-of-two addressing.

4.1 Expected M10K Budget

The current RTL uses Quartus M10K attributes to guide memory inference. Each 4096x8 color buffer maps to four M10Ks per PU, and each 4096x16 Z-buffer maps to eight M10Ks per PU. Therefore, each PU uses about 16 M10Ks for local image storage, and the 16-PU chain uses about 256 M10Ks before counting the triangle FIFO.

Component	Per-PU logical size	Per-PU M10Ks	Across 16 PUs	Raw storage
Framebuffer A	4096 x 8	4	64	524,288 bits / 64 KiB
Framebuffer B	4096 x 8	4	64	524,288 bits / 64 KiB
Z-buffer	4096 x 16	8	128	1,048,576 bits / 128 KiB
PU-local total	—	16	256	2,097,152 bits / 256 KiB
Triangle FIFO	64 x 431-bit	—	—	27,584 bits / 3.37 KiB
Total raw memory	—	—	266	2,124,736 bits / 259.37 KiB

The DE1-SoC provides roughly 397 M10K blocks, so the PU-local memories alone use about 256 / 397 = 64.5% of the available M10K blocks. The FIFO adds a small number of additional M10Ks, but the exact number depends on how Quartus packs the wide 64-entry packet memory.

This makes the design clearly BRAM-dominated. The pixel-unit logic is mostly adders, registers, counters, sign checks, and control state. The rasterizer datapath does not require per-pixel multipliers. Edge and depth updates are additions, and the 16-column jump is implemented as a left shift by 4 followed by an add.

4.2 Framebuffer and Z-buffer Choices

Three decisions drive the memory budget.

First, color is stored as 8-bit RGB332. This keeps each color buffer to one byte per pixel. Moving to 16-bit color would double the color framebuffer cost and would make double buffering much more expensive.

Second, the render target is 256x240 rather than full VGA resolution. A full 640x480 double framebuffer and 16-bit Z-buffer would not fit comfortably in the available on-chip memory. The VGA controller instead scales the 256x240 image to 512x480 and centers it on screen.

Third, the Z-buffer is local to each PU. Since each PU owns a disjoint set of columns, each PU can perform its own read-modify-write Z-test without fighting for a centralized memory port. The Z-buffer is initialized to 16'hFFFF, and the rasterizer uses a smaller-depth-wins comparison.

4.3 Frame Clear Cost

On reset, each PU clears its Z-buffer and both color buffers. On a normal frame swap, each PU clears its Z-buffer and only the new back color buffer. The front buffer is not cleared because VGA is scanning it out.

Each PU clears 4096 local addresses. Since all 16 PUs clear in parallel, the clear cost is about 4096 cycles total, not 4096 cycles times 16. The top-level uses a 4100-cycle clear window:

4100 cycles / 50 MHz = 82 us

Compared with one 60 Hz VGA frame:

82 us / 16.67 ms = 0.49%

So the clear pass is not a meaningful performance bottleneck.

4.3 Resource Summary

The final fitted numbers should be taken from the Quartus Fitter report. The RTL-derived expectation and the post Quartus utilizations are:

Resource	Utilization
Internal render target	256 x 240
VGA output	640 x 480 @ 60 Hz
Displayed active image	512 x 480 centered
Pixel units	16
PU-local memory	256 KiB raw
Triangle FIFO raw memory	3.37 KiB
Total raw memory	259.37 KiB
ALMs	8,933 (28% utilization)
Block Memory Bits	2,123,520 (52% utilization)
M10K Block (registers and buffers)	267 (67% utilization)
DSP usage	0
Pins	362 (79% utilization)
PLLs	0
DLLs	1 (25% utilization)

The measured post-fit M10K usage closely matches the analytical estimate. The distributed framebuffer and Z-buffer architecture accounts for 256 of the final 267 M10K blocks reported by Quartus, while the remaining ~11 M10Ks are used by the triangle FIFO and smaller inferred memory/register structures. This confirms that the design is BRAM-dominated, as expected from the design architecture.

In contrast, logic utilization remains moderate at 8,933 ALMs. Most of the data path consists of adders, comparators, counters, address generation, and control FSMs rather than large arithmetic units. The rasterizer does not require DSP blocks because edge stepping and depth interpolation

are implemented using incremental additions instead of multipliers. Post-fit results reported 0 DSP block usage.

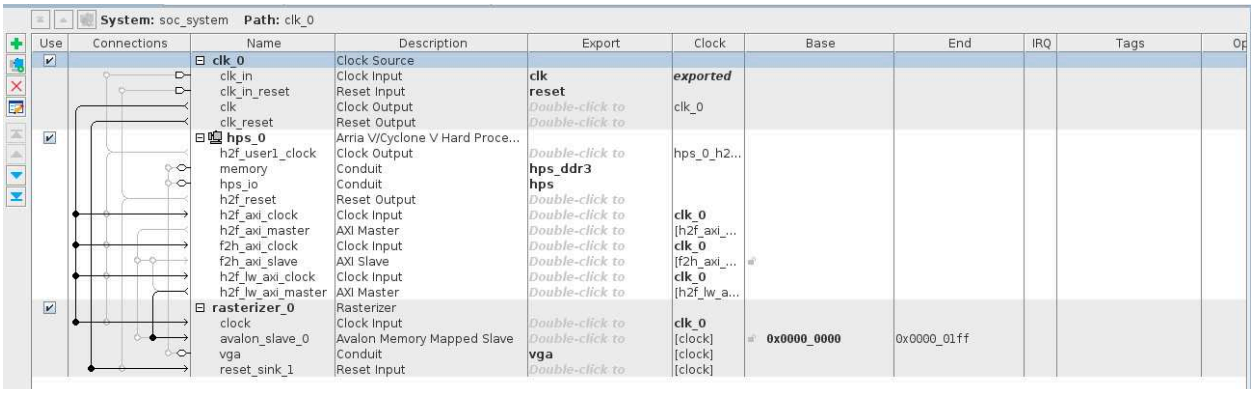
Overall, the implementation fits comfortably within the Cyclone V resource limits while still leaving headroom for additional control logic, shading features, or larger FIFOs.

5. Hardware/Software Interface

The ARM processor and FPGA fabric communicate over the Avalon memory-mapped bus. Triangle generation happens entirely on the ARM core in software; only the essential rasterization parameters are sent to hardware:

- Bounding box (xmin, xmax, ymin, ymax)
- Edge steps (a, b, c per edge) and pre-evaluated edge values (e_init per edge)
- Depth interpolation constants (z at origin, Δz per pixel in x and y)
- Color and back-face culling flag

Below is the Platform Designer interconnect view:



Below is the register info:

Note: All the hardware registers are 32 bit.

One triangle packet would be 15 regs and 2 extra regs for control and status transmission

Register	Access	Format / Description	
bbox_x	W	[xmin(10)|xmax(10) | pad(12)]	X bounds, pixel coords 0–319
bbox_y	W	[ymin(9) | ymax(9) | pad(14)]	Y bounds, pixel coords 0–239
a0	W	32-bit	Edge 0: coefficient a
b0	W	32-bit	Edge 0: coefficient b
a1	W	32-bit	Edge 1: coefficient a
b1	W	32-bit	Edge 1: coefficient b

a2	W	32-bit	Edge 2: coefficient a
b2	W	32-bit	Edge 2: coefficient b
e0_init	W	32-bit	Edge 0 value at bbox origin (c rolled in)
e1_init	W	32-bit	Edge 1 value at bbox origin (c rolled in)
e2_init	W	32-bit	Edge 2 value at bbox origin (c rolled in)
z_at_origin	W	32-bit	Depth at bbox origin (0.5 offset)
z_step_x	W	32-bit	Δz per pixel in x direction
z_step_y	W	32-bit	Δz per pixel in y direction
color_cull	W	[color(8)|front_facing(1)|pad(23)]	RGB332 color + back-face cull flag
commit	W	Any value triggers push	
status	R	[fifo_full(1)|fifo_level(6)|pad(25)]	Hardware FIFO state: occupancy + full flag

This approach minimizes the FPGA memory footprint by offloading expensive floating-point math (triangle setup) to the CPU.

5.1 HW->SW Datapath

The HW–SW interface builds on the lab3 ioctl design but replaces the per-triangle kernel crossing with a direct memory-mapped write path. The initial implementation used iowrite32() to submit each of the 17 packet words from kernel space, requiring two syscalls per triangle (one status read, one submit) and a privilege level switch on every submission. Profiling revealed this kernel-crossing overhead was the primary bottleneck, the hardware FIFO never filled (fifo_max = 0 across frames), confirming the CPU was spending more time in syscall overhead than the hardware spent rasterizing.

To eliminate this, the kernel driver exposes a rasterizer_mmap handler that uses remap_pfn_range to map the physical MMIO page (0xFF200000) directly into the userspace process's virtual address space. Userspace holds a pointer into this region and writes all 17 packet words with plain C stores, no kernel crossing per triangle.

The page is mapped with **pgprot_writecombine** rather than the default **pgprot_noncached**. This changes the ARM memory attribute from strongly-ordered Device memory (one AXI transaction per store) to allowing the ARM write buffer to coalesce all 17 stores into a single AXI burst to the FPGA. A **dsb sy** (Data Synchronisation Barrier) is issued after the 17 stores and before the COMMIT write to force the write buffer to drain, ensuring the FPGA's staging

registers hold the complete packet before the push is triggered and the writes are not happening in an out of order fashion.

Status polling is also done via the mmap'd region, reading g_regs[STATUS] directly without a syscall.

Data flow:

1. Software generates a triangle packet on the ARM core in setup_triangle()
2. Every 32 triangles, userspace reads g_regs[STATUS] and spins if fifo_full is set
3. Userspace writes all 17 words to g_regs[0..16] directly (writecombine AXI burst)
4. dsb sy barrier drains the write buffer
5. Userspace writes g_regs[COMMIT] to trigger the hardware FIFO push
6. Hardware receives the packet into its internal FIFO and feeds it to the rasterizer asynchronously

This design allows hardware and software to run asynchronously, with the hardware FIFO absorbing bursts from the CPU.

5.2 Kernel Driver Design

The driver is structured identically to Lab 3 with two primary ioctl commands retained as a fallback path, plus the rasterizer_mmap handler that enables the fast path:

- RASTERIZER_SUBMIT — fallback ioctl path: uses copy_from_user() to read the triangle struct, then 17 × iowrite32() to staging registers, then writes the commit register
- RASTERIZER_STATUS — fallback ioctl path: reads the hardware status register via ioread32() and returns FIFO level and full flag
- rasterizer_mmap — maps the physical MMIO page into userspace with pgprot_writecombine; once mapped, all triangle writes and status reads bypass the kernel entirely

The ioctl fallback is used only if mmap() fails at startup. In normal operation all triangle submissions go through the mmap fast path.

5.3 HW-SW Handshake

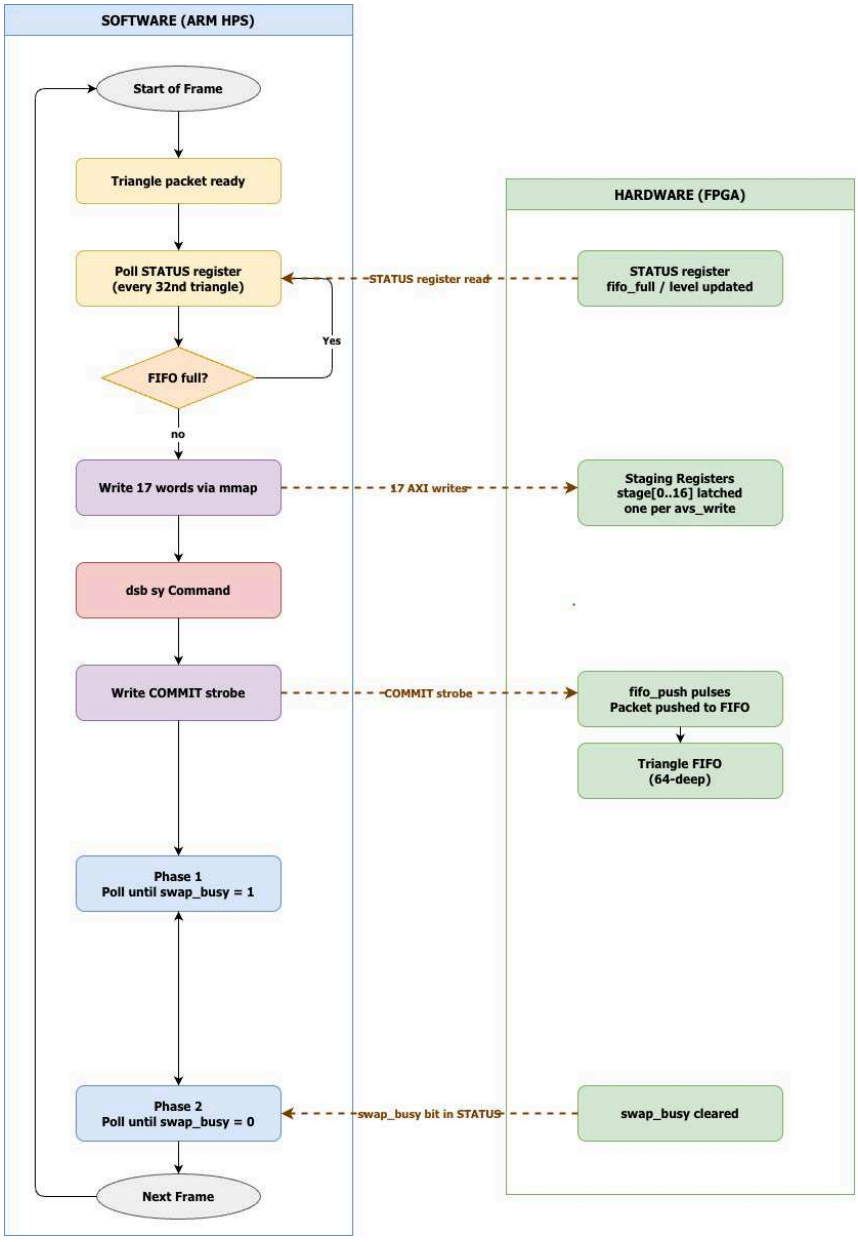
Triangle submission:

1. Software generates a triangle packet in setup_triangle() on the ARM core
2. Every 32 triangles, userspace reads the STATUS register via g_regs[STATUS] and spins if fifo_full is asserted
3. Userspace writes all 17 packet words to g_regs[0..16] and the ARM write buffer coalesces these into a single AXI burst to the FPGA staging registers
4. DSB SY barrier forces the burst to complete before the next store
5. Userspace writes g_regs[COMMIT] on which the hardware atomically pushes the staging register contents into its internal FIFO
6. The Triangle Dispatcher pulls packets from the FIFO and dispatches them to idle Pixel Units

Frame swap handshake:

When all triangles for a frame are submitted, userspace writes to the PRESENT register. Hardware raises swap_busy and the swap FSM waits for the FIFO to drain and the current rasterization pass to complete before swapping the VGA framebuffer. Userspace polls swap_busy via g_regs[STATUS] by first waiting for it to assert (closing the race between the PRESENT write and the FSM responding), then waiting for it to deassert before beginning the next frame. The framebuffer swap is hardware-controlled; software only fires the one-cycle PRESENT pulse and observes the completion via the status bit.

Below is a graphical representation of the overall HW SW interface


SOFTWARE (ARM HPS)
Start of Frame
Triangle packet ready
Poll STATUS register
(every 32nd triangle)
FIFO full?
Yes
no
Write 17 words via mmap
dsb sy Command
Write COMMIT strobe
Phase 1
Poll until swap_busy = 1
Phase 2
Poll until swap_busy = 0
Next Frame
HARDWARE (FPGA)
STATUS register read
STATUS register
fifo_full / level updated
17 AXI writes
Staging Registers
stage[0..16] latched
one per avs_write
COMMIT strobe
fifo_push pulses
Packet pushed to FIFO
Triangle FIFO
(64-deep)
swap_busy bit in STATUS
swap_busy cleared

6. Results and Performance Analysis

The final demo frame rate decreases as model complexity increases: the 15K triangle model runs at about 30 FPS, the 50K model runs at about 12 FPS, and the 160K model runs at about 4.5 FPS. To understand why performance scales this way, we used a software bottleneck probe that separates the frame into setup, submission, and present phases. The probe is not intended to be a perfectly optimized FPS measurement, since it uses clock_gettime() around operations. Instead, it is used to identify which parts of the pipeline dominate runtime.

6.1 Measurement Methodology

The probe measures three main timing categories:

Metric	What it measures
setup_ms	Time spent in setup_triangle() across the frame
submit_ms	Time spent submitting successfully built triangle packets to the FPGA
present_ms	Time from PRESENT request until swap/clear completes

setup_ms is measured by calling clock_gettime() immediately before and after each call to setup_triangle(), then accumulating the time across the frame. This isolates CPU-side triangle packet construction: Pineda edge setup, bounding-box clipping/snapping, depth interpolation setup, and packet packing.

submit_ms is measured similarly around submit_triangle(), but only for triangles that are successfully built and submitted. In the current MMIO path, each submitted triangle requires one STATUS read, seventeen packet-word writes, and one COMMIT write, for a total of 19 MMIO operations per submitted triangle.

present_ms is measured once per frame. The probe timestamps before issuing the present request, waits for swap_busy to assert and then deassert, and timestamps again after the wait completes. This means present_ms is not pure rasterization time. It is the post-submit frame handoff latency, including the buffer swap and clear sequence.

The measurements were averaged over steady-state portions of the traces.

6.2 ARM Triangle Setup

The clearest bottleneck is CPU-side triangle setup. setup_ms scales almost linearly with the number of input faces:

Model	Input faces	Setup time	Setup cost
15K	15,848	22.97 ms	1.45 µs / input triangle
50K	52,302	76.23 ms	1.46 µs / input triangle
160K	159,698	247.84 ms	1.55 µs / input triangle

This is a very consistent cost. Each input triangle costs about 1.5 µs on the ARM before the FPGA sees it. As model size increases, this per-triangle setup cost grows almost directly with the number of faces. This means the software front end is doing a large amount of work before the hardware rasterizer can begin processing the triangle.

6.3 ARM-to-FPGA Submission Overhead

The second bottleneck is packet submission. Once a triangle packet is built, software must send it to the rasterizer through the Avalon/MMIO path. The cost per triangle is also very consistent:

Model	Submitted packets	Submit time	Submit cost
15K	9,093	12.68 ms	1.40 µs / submitted triangle
50K	28,782	40.24 ms	1.40 µs / submitted triangle

160K	137,854	194.13 ms	1.41 µs / submitted triangle

The consistency shows that submission is mostly a fixed software/MMIO cost per triangle. In the probe path, each submitted triangle performs 19 MMIO operations, which produces a large number of MMIO operations per frame. This explains why submit_ms becomes large for the 160K model. Even if the FPGA rasterizer can consume work quickly, the ARM still has to push millions of MMIO operations per frame through the software interface.

6.4 Projection Cost

Projection is measurable, but it is not the main source of the slowdown. The measured project_ms values are:

Model	Projection time
15K	1.12 ms
50K	3.76 ms
160K	14.79 ms

Across the tested models, projection accounts for only about 1.7-1.8% of render_ms. Therefore, the performance drop is not primarily caused by vertex projection. The larger costs are triangle setup and packet submission.

6.5 FPGA Input Queue Behavior

The bottleneck probe does not include true on-chip hardware utilization counters, so we cannot claim an exact FPGA utilization percentage from this tool. However, the FIFO-related counters provide useful indirect evidence. Across the tested workloads:

```
None

fifo_max = 0


fifo_full_polls = 0


eagain = 0
```

This means the software never observed the input queue filling, never had to wait for FIFO space, and never received a submit retry. If the FPGA rasterizer were the immediate bottleneck, the first expected symptom would be queue buildup or FIFO-full polling. That did not happen.

The careful conclusion is that the FPGA input side was not visibly saturated in these runs. The software path was spending more time generating and submitting work than the hardware was spending accepting it.

6.6 Present Time

present_ms measures the post-submit handoff from PRESENT until the buffer swap and clear sequence completes. It does not scale upward with model size:

Model	Present time
15K	16.26 ms
50K	11.95 ms
160K	6.44 ms

This shows that present/swap/clear latency is not the main limiter. The largest growth with model size comes from setup_ms and submit_ms, not present_ms.

One reason present_ms does not grow with triangle count is that the average triangle gets smaller on screen as model complexity increases. The measured average bounding-box areas are 149.7px for the 15K model, 80.0px for 50K, and 32.8px for 160K. So while the 160K model has many

more triangles, each triangle covers fewer candidate pixels on average. This increases software work sharply, but it does not increase raster work per triangle at the same rate.

6.7 Combined Setup and Submit Cost

The clearest summary is the combined cost of setup and submission. Together, these two stages account for about half of the measured render time in all three runs:

Model	Setup + submit	Render time	Share of render time
15K	35.65 ms	67.66 ms	52.7%
50K	116.47 ms	223.14 ms	52.2%
160K	441.97 ms	820.25 ms	53.9%

This is a conservative view of the software bottleneck, because other CPU-side work outside setup_triangle() and submit_triangle() is not included in this combined number. The main scaling cost is therefore not “software” in a vague sense, but specifically:

1. triangle packet construction on the ARM, and
2. register-based packet submission to the FPGA.

7. Work Division

Gianna: Hardware algorithm (from FIFO to pixel units), resource budgeting/allocation, performance analysis

Srika: Resource allocation, pixel unit-framebuffer interface, framebuffer-VGA controller interface, performance analysis, flickering/triangle debugging

Shlok: HW/SW interface, kernel module development, register mapping for the Avalon bus interface, avalon_interface for the hardware end to accept packets

Sathvik - geometry calculations of coordinate transformation, golden reference, triangle packet setup, models in c and loading obj models

Aarush - geometry calculations of coordinate transformation

8. Lessons Learnt and Advice

Our initial design proposed one to four pixel units dividing work by partitioning screen regions. Memory allocation was not given much thought at that stage. Building a performant embedded system requires efficient use of limited resources and deliberate design decisions within tight hardware budgets.

During the design review with Prof. Edwards, we recognized that Pineda's edge function algorithm offers two key advantages:

1. It requires only simple arithmetic operations (additions and sign checks)
2. It parallelizes and pipelines really good

The first instinct in hardware development should be to identify what can be parallelized for time speedup and what can be pipelined for throughput improvement. Acting on that instinct, we restructured our implementation into a systolic chain of 16 pixel units, where each unit receives its seed from the previous one and immediately begins processing its assigned columns. This achieves 16 pixels per cycle throughput in steady state, a direct 4x improvement over the screen-division approach with 4 pixel units.

Our bottleneck analysis confirmed that software is the limiting factor. The following changes would improve frame rate:

1. Go to office hours and try implementing what the Prof. Edwards says, it will yield you good results
2. Use both cores of the DE1-SoC ARM processor to separate triangle dispatch from 3D geometry processing
3. Reduce the overhead of writes to the memory-mapped bus (we made significant progress here, but OS-level processing time remains a factor)
4. Remove all print statements. Every UART write is a blocking call and directly degrades software throughput
5. Schedule tasks across both cores with care. Analyze worst-case execution times for each thread and prioritize hardware write tasks over diagnostic tasks

Appendix A. Module Interfaces and Hierarchy

This appendix documents the Verilog modules in the custom peripheral, their hierarchy, and the protocols connecting them.

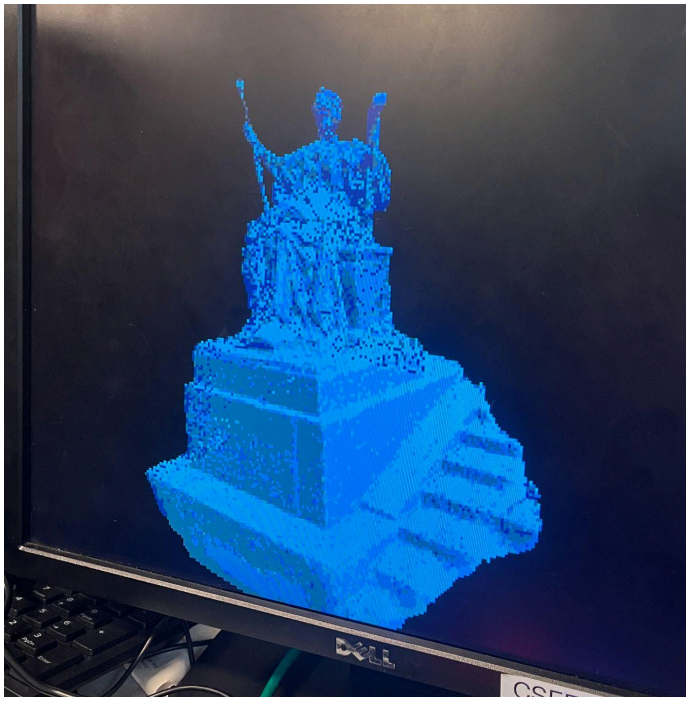
A.1 Module Hierarchy

```
None
rasterizer_top
├── avalon_slave           (×1)   HPS Avalon-MM interface, register staging
├── triangle_fifo          (×1)   64-entry packet FIFO, inferred M10K
├── triangle_dispatcher    (×1)   broadcasts packets to all pixel units
├── pixel_unit             (×2)   interleaved column rasterizer + local buffers
└── vga_framebuffer        (×1)  VGA scanout, framebuffer read selection, double-buffer ctrl
    └── vga_counters       (×1)   hcount/vcount generation, HS/VS timing
```

A.2 Inter-module Protocols

Connection	Protocol
HPS ↔ avalon_interface	Avalon-MM, 32-bit single-cycle reads/writes, no waitrequest
avalon_interface → triangle_fifo	One cycle push pulse on commit write; ignored when FIFO full
triangle_fifo → triangle_dispatcher	Pop handshake: dispatcher asserts pop when ready and FIFO non-empty
triangle_dispatcher → pixel_unit[0]	Starts systolic chain with valid_out[0] and latched packet fields
pixel_unit[i] → pixel_unit[i+1]	Systolic seed forwarding; valid, edge values, and depth
rasterizer_top → pixel_unit[i]	Broadcast triangle constants, framebuffer select, clear pulse
pixel_unit[i] → vga_framebuffer	Each PU returns local framebuffer read data
vga_framebuffer → rasterizer_top	frame_done pulse used by the swap FSM

A.3 Sample Images



160K Triangle Render of Columbia's Alma Matter (~4.5FPS)

10K Triangle Render of Teapot (~60FPS)

Source Code:

```
================================================================
FILE: hardware/triangle_packet.svh
================================================================

// triangle_packet.svh

`ifndef TRIANGLE_PACKET_SVH
`define TRIANGLE_PACKET_SVH

typedef struct packed {
    logic [9:0] bbox_xmin;
    logic [9:0] bbox_xmax;
    logic [8:0] bbox_ymin;
    logic [8:0] bbox_ymax;
    logic signed [31:0] a0, b0;
    logic signed [31:0] a1, b1;
    logic signed [31:0] a2, b2;
    logic signed [31:0] e0_init;
    logic signed [31:0] e1_init;
    logic signed [31:0] e2_init;
    logic signed [31:0] z_at_origin;
    logic signed [31:0] z_step_x;
    logic signed [31:0] z_step_y;
    logic [7:0] color;
    logic front_facing;
} triangle_packet_t;

`endif


================================================================
FILE: hardware/avalon_interface.sv
================================================================

// avalon_interface.sv
```

```
`include "triangle_packet.svh"


module avalon_interface (
    input logic clk, rst,
    input logic [6:0] avalon_address,
    input logic avalon_write,
    input logic [31:0] avalon_writedata,
    output logic [31:0] avalon_readdata,

    input logic pop,
    output logic pop_available,
    output triangle_packet_t pop_data,
    input logic pop_ACK,
    output logic fifo_full,
    output logic fifo_empty,
    output logic [5:0] fifo_level,
    output logic present_req,
    input logic swap_busy
);


    localparam logic [6:0] ADDR_PACKET_LO = 7'h00;
    localparam logic [6:0] ADDR_PACKET_HI = 7'h10;
    localparam logic [6:0] ADDR_COMMIT = 7'h11;
    localparam logic [6:0] ADDR_STATUS = 7'h12;
    localparam logic [6:0] ADDR_CONTROL = 7'h13;
    localparam logic [6:0] ADDR_PRESENT = 7'h14;


    logic [31:0] stage [17];


    logic [31:0] control_reg;


    logic fifo_push;
    triangle_packet_t fifo_push_data;
```

```
always_comb begin
    fifo_push_data = '0;


    fifo_push_data.bbox_xmin = {2'b00, stage[15][ 7: 0]};
    fifo_push_data.bbox_xmax = {2'b00, stage[15][15: 8]};
    fifo_push_data.bbox_ymin = {1'b0, stage[15][23:16]};
    fifo_push_data.bbox_ymax = {1'b0, stage[15][31:24]};


    fifo_push_data.a0 = stage[0];
    fifo_push_data.b0 = stage[1];
    fifo_push_data.a1 = stage[3];
    fifo_push_data.b1 = stage[4];
    fifo_push_data.a2 = stage[6];
    fifo_push_data.b2 = stage[7];


    fifo_push_data.e0_init = stage[ 9];
    fifo_push_data.e1_init = stage[10];
    fifo_push_data.e2_init = stage[11];


    fifo_push_data.z_at_origin = stage[12];
    fifo_push_data.z_step_x = stage[13];
    fifo_push_data.z_step_y = stage[14];


    fifo_push_data.color = stage[16][ 7:0];
    fifo_push_data.front_facing = stage[16][ 8];
end


always_ff @(posedge clk) begin

    fifo_push <= 1'b0;
    present_req<=1'b0;
```

```
    if (rst) begin
        for (int i = 0; i < 17; i++) stage[i] <= '0;
        control_reg<='0;
        present_req<=1'b0;
    end
    else if (avalon_write) begin
        if (avalon_address >= ADDR_PACKET_LO &&
            avalon_address<=ADDR_PACKET_HI) begin
            stage[avalon_address[4:0]]<=avalon_writedata;
        end
        else if (avalon_address == ADDR_COMMIT) begin


            if (!fifo_full)
                fifo_push<=1'b1;
        end
        else if (avalon_address == ADDR_CONTROL) begin


            control_reg<=avalon_writedata;
        end
        else if (avalon_address == ADDR_PRESENT) begin


            present_req<=1'b1;
        end
    end
end


always_comb begin
    avalon_readdata = 32'd0;
    case (avalon_address)
        ADDR_STATUS: begin
            avalon_readdata = {23'd0, swap_busy, fifo_full, fifo_empty, fifo_level};
        end
        ADDR_CONTROL: begin
            avalon_readdata = control_reg;
        end
```

```
            default: avalon_readdata = 32'd0;
        endcase
    end


    triangle_packet_t fifo_head;

    triangle_fifo #(.DEPTH(64)) u_fifo (
        .clk (clk), .rst (rst), .push (fifo_push), .push_data (fifo_push_data),
        .pop (pop_ACK), .pop_data (fifo_head), .full (fifo_full), .empty (fifo_empty),
        .level (fifo_level)
    );


    assign pop_available = !fifo_empty;
    assign pop_data = fifo_head;


    logic _pop_unused;
    assign _pop_unused = pop;

endmodule


================================================================
FILE: hardware/triangle_fifo.sv
================================================================

// triangle_fifo.sv

`include "triangle_packet.svh"


module triangle_fifo #(parameter int DEPTH = 64) (
    input logic clk, rst, push,
    input triangle_packet_t push_data,
    input logic pop,
    output triangle_packet_t pop_data,
```

```
    output logic full, empty,

    output logic [5:0] level
);
    localparam int ADDR_W =$clog2(DEPTH);
    localparam int CNT_W = $clog2(DEPTH + 1);
    triangle_packet_t mem [DEPTH];

    logic [ADDR_W-1:0] wr_ptr, rd_ptr;
    logic [CNT_W-1:0] count;

    assign empty = (count == '0);
    assign full =(count == DEPTH[CNT_W-1:0]);
    assign level = count[5:0];
    assign pop_data = mem[rd_ptr];

    always_ff @(posedge clk) begin
        if (rst) begin
            wr_ptr<= '0;
            rd_ptr<= '0;
            count<= '0;
        end
        else begin
            unique case ({push && !full, pop && !empty})
                2'b10: begin
                    mem[wr_ptr]<= push_data;
                    wr_ptr<= wr_ptr + 1'b1;
                    count <= count + 1'b1;
                end
                2'b01: begin
                    rd_ptr<= rd_ptr + 1'b1;
                    count<= count - 1'b1;
                end
                2'b11: begin
                    mem[wr_ptr]<= push_data;
                    wr_ptr <= wr_ptr + 1'b1;
                    rd_ptr <= rd_ptr + 1'b1;
                end
                2'b00: begin
```

```
            end
        endcase

        end
    end
endmodule
```

```
================================================================
FILE: hardware/triangle_dispatcher.sv
================================================================
```

```
// triangle_dispatcher.sv

`include "triangle_packet.svh"


module triangle_dispatcher #( parameter int N_PU = 16) (
    input logic clk, rst,
    output logic pop,
    input logic pop_available,
    input triangle_packet_t pop_data,
    output logic pop_ACK,

    output logic [N_PU-1:0] valid_out,
    output triangle_packet_t packet_out,
    input logic [N_PU-1:0] ready_in,
    input logic block_dispatch
);

    typedef enum logic [1:0] {
        WAIT,
        BCAST,
        COOLDOWN
    } state_t;

    state_t state;
    triangle_packet_t latched;
    logic all_ready;
```

```
  assign all_ready =&ready_in;
  assign packet_out =latched;

  always_ff @(posedge clk) begin
    pop <= 1'b0;
    pop_ACK <= 1'b0;
    valid_out<= '0;

    if (rst) begin
      state <= WAIT;
      latched <= '0;
    end else begin
      case (state)
        WAIT: begin
          if (all_ready && !block_dispatch) pop <= 1'b1;

          if (pop && pop_available && !block_dispatch)
            begin
            latched <= pop_data;
            state <= BCAST;
            end
        end
        BCAST: begin
          pop_ACK <= 1'b1;
          valid_out<='1;
          state <= COOLDOWN;
        end
        COOLDOWN: begin
          state<= WAIT;
        end

        default: state <= WAIT;
      endcase
    end
  end
endmodule
```

==========

FILE: hardware/pixel_unit.sv

==========

```
// pixel_unit.sv

`include "triangle_packet.svh"


module pixel_unit #(
    parameter int  PU_ID = 0,
    parameter bit  IS_LAST_PU = 1'b0,
    parameter int  FB_DEPTH = 4096,
    parameter int  Z_DEPTH = 4096,
    parameter int  Z_MSB = 27,
    parameter int  Z_LSB = 12,


    parameter bit  DO_INIT_CLEAR  = 1'b1
) (
    input  logic clk,
    input  logic rst,


    output logic ready,


    input  logic z_clear_start,


    input  logic seed_valid_in,
    input  logic signed [31:0] seed_e0_in,
    input  logic signed [31:0] seed_e1_in,
    input  logic signed [31:0] seed_e2_in,
    input  logic signed [31:0] seed_z_in,


    input  logic signed [31:0] a0_in, a1_in, a2_in, z_step_x_in,
    input  logic signed [31:0] b0_in, b1_in, b2_in, z_step_y_in,
    input  logic [7:0] color_in,
```

```
    input  logic [7:0] col_base_in,
    input  logic [7:0] row_base_in,
    input  logic [7:0] last_row_in,
    input  logic [7:0] last_col_in,


    output logic seed_valid_out,
    output logic signed [31:0] seed_e0_out,
    output logic signed [31:0] seed_e1_out,
    output logic signed [31:0] seed_e2_out,
    output logic signed [31:0] seed_z_out,


    output logic p_write,
    output logic [7:0] p_col,
    output logic [7:0] p_row,
    output logic [7:0] p_color,
    output logic [15:0] p_depth,


    input  logic [11:0] vga_r_addr,
    input  logic vga_r_buf_sel,
    output logic [7:0] vga_r_data,


    input  logic fb_write_sel
);

    typedef enum logic [1:0] {
       S_INIT_CLEAR, S_IDLE, S_ACTIVE,  S_CLEAR
    } state_t;
    state_t state;


    logic [11:0] clear_addr;
    logic clear_done;
    assign clear_done = (clear_addr == 12'(Z_DEPTH - 1));
```

```
logic signed [31:0] a0_q, a1_q, a2_q, z_step_x_q;
logic signed [31:0] b0_q, b1_q, b2_q, z_step_y_q;
logic [7:0] color_q, col_base_q, row_base_q, last_row_q;
logic [7:0] last_col_q;


logic signed [31:0] e0, e1, e2, z;
logic [7:0] cur_row;


logic signed [31:0] e0_col_top, e1_col_top, e2_col_top, z_col_top;

assign ready = (state == S_IDLE);


logic inside_now;
assign inside_now = (state == S_ACTIVE)  && (e0[31] == 1'b0) && (e1[31] == 1'b0) && (e2[31] == 1'b0);


logic [11:0] addr_now;
assign addr_now = {col_base_q[7:4], cur_row};


logic q_valid;
logic q_inside;
logic [11:0] q_addr;
logic [7:0]  q_col, q_row, q_color;
logic [15:0] q_z;


(* ramstyle = "M10K", max_depth = 512 *)
logic [15:0] z_mem [Z_DEPTH];

logic [15:0] z_rd;


(* ramstyle = "M10K", max_depth = 1024 *)
logic [7:0]  fb_a_mem [FB_DEPTH];
(* ramstyle = "M10K", max_depth = 1024 *)
```

```
logic [7:0]  fb_b_mem [FB_DEPTH];

logic [7:0]  fb_a_rd, fb_b_rd;


logic z_pass;
assign z_pass = q_valid && q_inside && (q_z <= z_rd);

always_ff @(posedge clk) begin

   seed_valid_out <= 1'b0;
   p_write <= 1'b0;
   q_valid <= 1'b0;

   if (rst) begin
      state <= DO_INIT_CLEAR ? S_INIT_CLEAR : S_IDLE;
      cur_row <= '0;
      clear_addr <= '0;
   end else begin
      unique case (state)
         S_INIT_CLEAR: begin


            if (clear_done) begin
               state <= S_IDLE;
               clear_addr <= '0;
            end else begin
               clear_addr <= clear_addr + 12'd1;
            end
         end

         S_IDLE: begin
            if (z_clear_start) begin
               state <= S_CLEAR;
               clear_addr <= '0;
            end else if (seed_valid_in) begin

               e0 <= seed_e0_in;
               e1 <= seed_e1_in;
```

```
        e2 <= seed_e2_in;
        z  <= seed_z_in;


        e0_col_top <= seed_e0_in;
        e1_col_top <= seed_e1_in;
        e2_col_top <= seed_e2_in;
        z_col_top  <= seed_z_in;


        a0_q <= a0_in;
        a1_q <= a1_in;
        a2_q <= a2_in;
        z_step_x_q <= z_step_x_in;
        b0_q <= b0_in;
        b1_q <= b1_in;
        b2_q <= b2_in;
        z_step_y_q <= z_step_y_in;
        color_q <= color_in;
        col_base_q <= col_base_in;
        row_base_q <= row_base_in;
        last_row_q <= last_row_in;
        last_col_q <= last_col_in;

        cur_row <= row_base_in;
        state <= S_ACTIVE;


        if (!IS_LAST_PU) begin
           seed_valid_out <= 1'b1;
           seed_e0_out <= seed_e0_in + a0_in;
           seed_e1_out <= seed_e1_in + a1_in;
           seed_e2_out <= seed_e2_in + a2_in;
           seed_z_out <= seed_z_in  + z_step_x_in;
        end
     end
end

S_ACTIVE: begin
```

```
    q_valid  <= 1'b1;
    q_inside <= inside_now;
    q_addr <= addr_now;
    q_col <= col_base_q;
    q_row <= cur_row;
    q_color  <= color_q;
    q_z <= z[Z_MSB:Z_LSB];


    e0 <= e0 + b0_q;
    e1 <= e1 + b1_q;
    e2 <= e2 + b2_q;
    z <= z  + z_step_y_q;
    cur_row <= cur_row + 8'd1;

    if (cur_row == last_row_q) begin


       if (({1'b0, col_base_q} + 9'd16) > {1'b0, last_col_q}) begin
          state <= S_IDLE;
       end else begin


          e0 <= e0_col_top + (a0_q <<< 4);
          e1 <= e1_col_top + (a1_q <<< 4);
          e2 <= e2_col_top + (a2_q <<< 4);
          z <= z_col_top  + (z_step_x_q <<< 4);
          e0_col_top <= e0_col_top + (a0_q <<< 4);
          e1_col_top <= e1_col_top + (a1_q <<< 4);
          e2_col_top <= e2_col_top + (a2_q <<< 4);
          z_col_top  <= z_col_top  + (z_step_x_q <<< 4);

          col_base_q <= col_base_q + 8'd16;
          cur_row <= row_base_q;

       end
    end
end
```

```
      S_CLEAR: begin


         if (clear_done) begin
            state   <= S_IDLE;
            clear_addr <= '0;
         end else begin
            clear_addr <= clear_addr + 12'd1;
         end
      end

      default: state <= S_IDLE;
   endcase
end


z_rd <= z_mem[addr_now];


if (state == S_INIT_CLEAR) begin
   z_mem   [clear_addr] <= 16'hFFFF;
   fb_a_mem[clear_addr] <= 8'h00;
   fb_b_mem[clear_addr] <= 8'h00;
end else if (state == S_CLEAR) begin
   z_mem[clear_addr] <= 16'hFFFF;
   if (fb_write_sel) fb_b_mem[clear_addr] <= 8'h00;
   else              fb_a_mem[clear_addr] <= 8'h00;
end else if (z_pass) begin
   z_mem[q_addr] <= q_z;
   if (fb_write_sel) fb_b_mem[q_addr] <= q_color;
   else              fb_a_mem[q_addr] <= q_color;
   p_write <= 1'b1;
   p_col   <= q_col;
   p_row   <= q_row;
   p_color <= q_color;
   p_depth <= q_z;
end
```

```
        fb_a_rd <= fb_a_mem[vga_r_addr];
        fb_b_rd <= fb_b_mem[vga_r_addr];
    end

    assign vga_r_data = vga_r_buf_sel ? fb_b_rd : fb_a_rd;


    `ifndef SYNTHESIS
    always @(posedge clk) begin
        if (!rst && state == S_IDLE && seed_valid_in) begin
            assert (col_base_in[3:0] == PU_ID[3:0])
                else $error("pixel_unit PU_ID=%0d got col_base=%0d (low nibble %0d, expected %0d)",
                        PU_ID, col_base_in, col_base_in[3:0], PU_ID[3:0]);
        end
    end
    `endif

endmodule


================================================================
FILE: hardware/rasterizer_top.sv
================================================================

// rasterizer_top.sv

`include "triangle_packet.svh"


module rasterizer_top (
    input logic clk, rst,
    input logic [6:0] avs_address,
    input logic avs_write,
    input logic [31:0] avs_writedata,
    output logic [31:0] avs_readdata,
    output logic avs_waitrequest,
```

```
    output logic [7:0] VGA_R, VGA_G, VGA_B,
    output logic VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n
);


    assign avs_waitrequest=1'b0;

    localparam int N_PU=16;


    logic disp_pop, disp_pop_available;
    triangle_packet_t disp_pop_data;
    logic disp_pop_ACK, fifo_full, fifo_empty;
    logic [5:0] fifo_level;
    logic present_req, present_pending;


    logic swap_busy;


    avalon_interface u_avalon (
        .clk (clk), .rst (rst), .avalon_address (avs_address), .avalon_write (avs_write),
        .avalon_writedata (avs_writedata), .avalon_readdata (avs_readdata), .pop (disp_pop),
        .pop_available (disp_pop_available), .pop_data (disp_pop_data), .pop_ACK (disp_pop_ACK),
        .fifo_full (fifo_full), .fifo_empty (fifo_empty), .fifo_level (fifo_level),
        .present_req (present_req), .swap_busy (swap_busy)
    );

    logic [N_PU-1:0] pu_valid_seed;
    triangle_packet_t pu_packet;
    logic [N_PU-1:0] pu_ready;


    logic block_dispatch;

    triangle_dispatcher #(.N_PU(N_PU)) u_dispatcher (
        .clk (clk), .rst (rst), .pop (disp_pop), .pop_available (disp_pop_available),
        .pop_data (disp_pop_data), .pop_ACK (disp_pop_ACK), .valid_out (pu_valid_seed),
        .packet_out (pu_packet), .ready_in (pu_ready), .block_dispatch(block_dispatch)
```

```
    );


    logic chain_idle;
    assign chain_idle = &pu_ready;


    logic chain_seed_valid [N_PU];
    logic signed [31:0] chain_seed_e0 [N_PU], chain_seed_e1 [N_PU], chain_seed_e2 [N_PU];
logic signed [31:0] chain_seed_z [N_PU];


    assign chain_seed_valid[0] = pu_valid_seed[0];
    assign chain_seed_e0[0] = pu_packet.e0_init;
    assign chain_seed_e1[0] = pu_packet.e1_init;
    assign chain_seed_e2[0] = pu_packet.e2_init;
    assign chain_seed_z [0] = pu_packet.z_at_origin;


    logic [11:0] vga_r_addr;
    logic vga_r_buf_sel;
    logic fb_write_sel;
    logic [7:0] pu_vga_rdata [N_PU];


    logic z_clear_start;


    logic [N_PU-1:0] pu_p_write;
    logic [7:0] pu_p_col [N_PU],  pu_p_row [N_PU], pu_p_color [N_PU];
    logic [15:0] pu_p_depth [N_PU];

    genvar gi;
    generate
        for (gi = 0; gi < N_PU; gi++) begin : g_pu

            logic sv_out;
            logic signed [31:0] se0_out, se1_out, se2_out, sz_out;
```

```
        logic [7:0] col_base_for_pu;
        assign col_base_for_pu=pu_packet.bbox_xmin[7:0] + 8'(gi);

        pixel_unit #(.PU_ID (gi), .IS_LAST_PU ((gi == N_PU - 1) ? 1'b1 : 1'b0))
        u_pu (
            .clk (clk), .rst (rst), .ready (pu_ready[gi]), .z_clear_start (z_clear_start),
 .seed_valid_in (chain_seed_valid[gi]),              .seed_e0_in (chain_seed_e0[gi]), .seed_e1_in (chain_seed_e1[gi]),
 .seed_e2_in (chain_seed_e2[gi]), .seed_z_in (chain_seed_z[gi]),  .a0_in (pu_packet.a0), .a1_in (pu_packet.a1),
 .a2_in (pu_packet.a2), .z_step_x_in (pu_packet.z_step_x), .b0_in (pu_packet.b0), .b1_in (pu_packet.b1),
.b2_in (pu_packet.b2), .z_step_y_in (pu_packet.z_step_y), .color_in (pu_packet.color),
.col_base_in (col_base_for_pu), .row_base_in (pu_packet.bbox_ymin[7:0]), .last_row_in
(pu_packet.bbox_ymax[7:0]), .last_col_in (pu_packet.bbox_xmax[7:0]),
            .seed_valid_out (sv_out), .seed_e0_out (se0_out), .seed_e1_out (se1_out), .seed_e2_out (se2_out),
.seed_z_out (sz_out),
            .p_write (pu_p_write[gi]), .p_col (pu_p_col[gi]), .p_row (pu_p_row[gi]), .p_color (pu_p_color[gi]),
.p_depth (pu_p_depth[gi]),
            .vga_r_addr (vga_r_addr), .vga_r_buf_sel (vga_r_buf_sel), .vga_r_data (pu_vga_rdata[gi]), .fb_write_sel
 (fb_write_sel)
        );


        if (gi == N_PU-1) begin : g_chain_tail


        end else begin : g_chain
            assign chain_seed_valid[gi + 1]=sv_out;
            assign chain_seed_e0 [gi + 1]=se0_out;
            assign chain_seed_e1 [gi + 1]=se1_out;
            assign chain_seed_e2 [gi + 1]=se2_out;
            assign chain_seed_z [gi + 1]=sz_out;
        end
      end
   endgenerate


   localparam int CLEAR_CYCLES = 4100;

   typedef enum logic [1:0] {
```

```
    SW_IDLE,
    SW_ARM,
    SW_CLEARING
} swap_state_t;

swap_state_t sw_state;
logic [12:0] clear_cnt;
logic frame_done;


assign block_dispatch=(sw_state != SW_IDLE);


assign swap_busy=(sw_state != SW_IDLE) || present_pending;

always_ff @(posedge clk) begin
    z_clear_start<=1'b0;

    if (rst) begin
        sw_state <= SW_IDLE;
        fb_write_sel <= 1'b0;
        clear_cnt <= '0;
        present_pending<=1'b0;
    end

    else begin
        if (present_req)
            present_pending<=1'b1;
        unique case (sw_state)
            SW_IDLE: begin
                if (present_pending && frame_done && fifo_empty) sw_state <= SW_ARM;
            end

            SW_ARM: begin
                if (chain_idle) begin
                    fb_write_sel <= ~fb_write_sel;
                    z_clear_start<=1'b1;
                    clear_cnt <= 13'(CLEAR_CYCLES);
                    present_pending<=1'b0;
```

```
                sw_state <= SW_CLEARING;
            end
        end

        SW_CLEARING: begin
            if (clear_cnt == 0) sw_state <= SW_IDLE;
            else clear_cnt <= clear_cnt-13'd1;
        end

        default: sw_state<=SW_IDLE;
      endcase
    end
  end

  vga_framebuffer u_vga (
    .clk (clk), .reset (rst), .pu_vga_data (pu_vga_rdata), .vga_r_addr (vga_r_addr), .vga_r_buf_sel(vga_r_buf_sel),
    .fb_write_sel (fb_write_sel), .frame_done (frame_done), .VGA_R (VGA_R), .VGA_G (VGA_G), .VGA_B
(VGA_B),
    .VGA_CLK (VGA_CLK), .VGA_HS (VGA_HS), .VGA_VS (VGA_VS), .VGA_BLANK_n
(VGA_BLANK_n), .VGA_SYNC_n (VGA_SYNC_n)
  );

endmodule
```

```
================================================================
FILE: hardware/vga_framebuffer.sv
================================================================
```

```
// vga_framebuffer.sv

module vga_framebuffer (
  input logic clk, reset,
  input logic [7:0] pu_vga_data [0:15],
  output logic [11:0] vga_r_addr,
  output logic vga_r_buf_sel,
  input logic fb_write_sel,
  output logic frame_done,
  output logic [7:0] VGA_R, VGA_G, VGA_B,
```

```
    output logic VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n);

    logic [10:0] hcount;
    logic [9:0] vcount;

    vga_counters counters (.clk50(clk), .reset(reset), .hcount(hcount), .vcount(vcount), .frame_done(frame_done),
.VGA_CLK(VGA_CLK),
                           .VGA_HS(VGA_HS), .VGA_VS(VGA_VS), .VGA_BLANK_n(VGA_BLANK_n),
.VGA_SYNC_n(VGA_SYNC_n));

    logic [10:0] fb_x_tmp;
    logic [7:0] fb_x;
    logic [7:0] fb_y;
    logic in_fb_region;

    assign in_fb_region = (hcount >= 11'd64)  &&
                          (hcount <  11'd576) &&
                         (vcount <  10'd480);

    assign fb_x_tmp = (hcount - 11'd64)>> 1;
    assign fb_x = fb_x_tmp[7:0];

    assign fb_y = vcount[8:1];
    assign vga_r_addr = {fb_x[7:4], fb_y};



    assign vga_r_buf_sel = ~fb_write_sel;

    logic [3:0] pu_sel_q;
    logic in_fb_region_q;

    always_ff @(posedge clk or posedge reset) begin
        if (reset) begin
            pu_sel_q <= 4'd0;
            in_fb_region_q <= 1'b0;
        end
        else begin
```

```
        pu_sel_q <= fb_x[3:0];
        in_fb_region_q<= in_fb_region;
        end
    end

    logic [7:0] pixelcolor;
    assign pixelcolor =pu_vga_data[pu_sel_q];

    always_comb begin
        VGA_R = 8'h00;
        VGA_G=8'h00;
        VGA_B = 8'h00;

        if (VGA_BLANK_n && in_fb_region_q) begin
            VGA_R ={pixelcolor[7:5], pixelcolor[7:5], pixelcolor[7:6]};
            VGA_G= {pixelcolor[4:2], pixelcolor[4:2], pixelcolor[4:3]};
            VGA_B = {pixelcolor[1:0], pixelcolor[1:0], pixelcolor[1:0], pixelcolor[1:0]};
      end
 end
endmodule


module vga_counters (
    input logic clk50, reset,
    output logic [10:0] hcount,
    output logic [9:0]  vcount,
    output logic frame_done, VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n
);

    logic pixel_tick;

    always_ff @(posedge clk50 or posedge reset) begin
        if (reset) begin
            pixel_tick <= 1'b0;
            hcount <= 11'd0;
            vcount <= 10'd0;
            frame_done <= 1'b0;
        end
        else begin
```

```
        pixel_tick <= ~pixel_tick;
        frame_done <= 1'b0;

        if (pixel_tick) begin
          if (hcount == 11'd799) begin
            hcount <= 11'd0;

            if (vcount == 10'd524) begin
              vcount <= 10'd0;
              frame_done <= 1'b1;
            end

            else begin
              vcount <= vcount + 10'd1;
            end
          end
          else begin
            hcount <= hcount + 11'd1;
          end
      end
    end
   end

assign VGA_CLK = pixel_tick;
assign VGA_HS = ~((hcount >= 11'd656) && (hcount < 11'd752));
assign VGA_VS = ~((vcount >= 10'd490) && (vcount < 10'd492));
assign VGA_BLANK_n = (hcount < 11'd640) && (vcount < 10'd480);
assign VGA_SYNC_n = 1'b0;

endmodule
```

```
================================================================
FILE: sim/avalon_interface_tb.sv
================================================================

// avalon_interface_tb.sv

`timescale 1ns/1ps
```

```
`include "triangle_packet.svh"


module avalon_interface_tb;

    localparam int NUM_PACKETS = 5;

    localparam logic [6:0] ADDR_PACKET_LO = 7'h00;
    localparam logic [6:0] ADDR_COMMIT    = 7'h11;
    localparam logic [6:0] ADDR_STATUS    = 7'h12;
    localparam logic [6:0] ADDR_CONTROL   = 7'h13;


    logic           clk;
    logic           rst;

    logic [6:0]     avalon_address;
    logic           avalon_write;
    logic [31:0]    avalon_writedata;
    logic [31:0]    avalon_readdata;

    logic           pop;
    logic           pop_available;
    triangle_packet_t pop_data;
    logic           pop_ACK;

    logic           fifo_full;
    logic           fifo_empty;
    logic [5:0]     fifo_level;

    avalon_interface dut (
        .clk            (clk),
        .rst            (rst),
        .avalon_address   (avalon_address),
        .avalon_write     (avalon_write),
        .avalon_writedata (avalon_writedata),
        .avalon_readdata  (avalon_readdata),
        .pop            (pop),
        .pop_available    (pop_available),
```

```
    .pop_data        (pop_data),
    .pop_ACK         (pop_ACK),
    .fifo_full       (fifo_full),
    .fifo_empty      (fifo_empty),
    .fifo_level      (fifo_level)
);

initial begin
    clk = 0;
    forever #5 clk = ~clk;
end


int cycle;
always_ff @(posedge clk) begin
    if (rst) cycle <= 0;
    else     cycle <= cycle + 1;
end

int errors = 0;
task automatic check(input bit cond, input string msg);
    if (!cond) begin
        $error("FAIL: %s", msg);
        errors++;
    end
endtask


always_ff @(posedge clk) begin
    if (!rst && avalon_write) begin
        if (avalon_address >= ADDR_PACKET_LO && avalon_address <= 7'h10) begin
            $display("[cyc=%0d t=%0t] BUS WR  stage[%0d] <= %h",
                    cycle, $time, avalon_address[4:0], avalon_writedata);
        end
        else if (avalon_address == ADDR_COMMIT) begin
            $display("[cyc=%0d t=%0t] BUS WR  COMMIT  (data=%h, fifo_level=%0d, full=%b)",
                    cycle, $time, avalon_writedata, fifo_level, fifo_full);
        end
        else if (avalon_address == ADDR_CONTROL) begin
```

```
        $display("[cyc=%0d t=%0t] BUS WR  CONTROL <= %h",
                cycle, $time, avalon_writedata);
      end
      else begin
        $display("[cyc=%0d t=%0t] BUS WR  addr=%h data=%h (UNMAPPED)",
                cycle, $time, avalon_address, avalon_writedata);
      end
    end
end


always_ff @(posedge clk) begin
    if (!rst && dut.fifo_push) begin
      $display("[cyc=%0d t=%0t] FIFO PUSH",
            cycle, $time);
      $display("              a0=%h  b0=%h", dut.fifo_push_data.a0, dut.fifo_push_data.b0);
      $display("              a1=%h  b1=%h", dut.fifo_push_data.a1, dut.fifo_push_data.b1);
      $display("              a2=%h  b2=%h", dut.fifo_push_data.a2, dut.fifo_push_data.b2);
      $display("              e0_init=%h  e1_init=%h  e2_init=%h",
            dut.fifo_push_data.e0_init, dut.fifo_push_data.e1_init, dut.fifo_push_data.e2_init);
      $display("              z_at_origin=%h  z_step_x=%h  z_step_y=%h",
            dut.fifo_push_data.z_at_origin, dut.fifo_push_data.z_step_x, dut.fifo_push_data.z_step_y);
      $display("              bbox xmin=%0d xmax=%0d ymin=%0d ymax=%0d",
            dut.fifo_push_data.bbox_xmin, dut.fifo_push_data.bbox_xmax,
            dut.fifo_push_data.bbox_ymin, dut.fifo_push_data.bbox_ymax);
      $display("              color=%h front_facing=%b",
            dut.fifo_push_data.color, dut.fifo_push_data.front_facing);
    end
end


always_ff @(posedge clk) begin
    if (!rst && pop_ACK) begin
      $display("[cyc=%0d t=%0t] FIFO POP (level was %0d)",
            cycle, $time, fifo_level);
      $display("              a0=%h color=%h bbox_xmin=%0d ymin=%0d",
            pop_data.a0, pop_data.color, pop_data.bbox_xmin, pop_data.bbox_ymin);
    end
end
```

```
logic [5:0] prev_level;
always_ff @(posedge clk) begin
   if (rst) begin
      prev_level <= '0;
   end else begin
      if (fifo_level != prev_level) begin
         $display("[cyc=%0d t=%0t] LEVEL %0d -> %0d   (full=%b empty=%b)",
                cycle, $time, prev_level, fifo_level, fifo_full, fifo_empty);
      end
      prev_level <= fifo_level;
   end
end


task automatic avalon_write_word(input logic [6:0] addr,
                          input logic [31:0] data);
   avalon_address  <= addr;
   avalon_writedata <= data;
   avalon_write    <= 1'b1;
   @(posedge clk);
   avalon_write    <= 1'b0;
   avalon_address  <= '0;
   avalon_writedata <= '0;
endtask

task automatic avalon_read_word(input logic [6:0] addr,
                          output logic [31:0] data);
   avalon_address <= addr;
   avalon_write   <= 1'b0;
   @(posedge clk);
   data = avalon_readdata;
   $display("[cyc=%0d t=%0t] BUS RD  addr=%h -> %h",
          cycle, $time, addr, data);
   avalon_address <= '0;
endtask
```

```
function automatic void make_packet_words(input int idx,
                                          output logic [31:0] words [17]);
    words[ 0] = 32'(idx * 32'h1000 + 32'h0001);
    words[ 1] = 32'(idx * 32'h1000 + 32'h0002);
    words[ 2] = 32'(idx * 32'h1000 + 32'h0003);
    words[ 3] = 32'(idx * 32'h1000 + 32'h0004);
    words[ 4] = 32'(idx * 32'h1000 + 32'h0005);
    words[ 5] = 32'(idx * 32'h1000 + 32'h0006);
    words[ 6] = 32'(idx * 32'h1000 + 32'h0007);
    words[ 7] = 32'(idx * 32'h1000 + 32'h0008);
    words[ 8] = 32'(idx * 32'h1000 + 32'h0009);
    words[ 9] = 32'(idx * 32'h1000 + 32'h000A);
    words[10] = 32'(idx * 32'h1000 + 32'h000B);
    words[11] = 32'(idx * 32'h1000 + 32'h000C);
    words[12] = 32'(idx * 32'h1000 + 32'h000D);
    words[13] = 32'(idx * 32'h1000 + 32'h000E);
    words[14] = 32'(idx * 32'h1000 + 32'h000F);
    words[15] = {8'(idx + 8'd40),
                 8'(idx + 8'd10),
                 8'(idx + 8'd50),
                 8'(idx + 8'd20)};
    words[16] = {23'd0, 1'b1, 8'(idx + 8'h80)};
endfunction

function automatic triangle_packet_t expected_packet(input int idx);
    triangle_packet_t p;
    p             = '0;
    p.bbox_xmin   = 10'(idx + 20);
    p.bbox_xmax   = 10'(idx + 50);
    p.bbox_ymin   = 9'(idx + 10);
    p.bbox_ymax   = 9'(idx + 40);
    p.a0          = 32'(idx * 32'h1000 + 32'h0001);
    p.b0          = 32'(idx * 32'h1000 + 32'h0002);
    p.a1          = 32'(idx * 32'h1000 + 32'h0004);
    p.b1          = 32'(idx * 32'h1000 + 32'h0005);
    p.a2          = 32'(idx * 32'h1000 + 32'h0007);
    p.b2          = 32'(idx * 32'h1000 + 32'h0008);
    p.e0_init     = 32'(idx * 32'h1000 + 32'h000A);
    p.e1_init     = 32'(idx * 32'h1000 + 32'h000B);
```

```
        p.e2_init      = 32'(idx * 32'h1000 + 32'h000C);
        p.z_at_origin  = 32'(idx * 32'h1000 + 32'h000D);
        p.z_step_x     = 32'(idx * 32'h1000 + 32'h000E);
        p.z_step_y     = 32'(idx * 32'h1000 + 32'h000F);
        p.color        = 8'(idx + 8'h80);
        p.front_facing = 1'b1;
        return p;
    endfunction


    task automatic submit_packet(input int idx);
        logic [31:0] words [17];
        $display("============================================================");
        $display("[cyc=%0d t=%0t] >>> SUBMIT PACKET idx=%0d", cycle, $time, idx);
        $display("============================================================");
        make_packet_words(idx, words);
        for (int i = 0; i < 17; i++) begin
            avalon_write_word(7'(i), words[i]);
        end
        avalon_write_word(ADDR_COMMIT, 32'h1);
        $display("[cyc=%0d t=%0t] <<< SUBMIT PACKET idx=%0d done\n",
                 cycle, $time, idx);
    endtask

task automatic pop_one(output triangle_packet_t got);
    $display("============================================================");
    $display("[cyc=%0d t=%0t] >>> POP_ONE  pop_available=%b empty=%b level=%0d",
             cycle, $time, pop_available, fifo_empty, fifo_level);
    pop <= 1'b1;
    while (!pop_available) @(posedge clk);


    #1;
    got = pop_data;
    $display("[cyc=%0d t=%0t] POP captured: a0=%h color=%h xmin=%0d ymin=%0d",
             cycle, $time, got.a0, got.color, got.bbox_xmin, got.bbox_ymin);
    pop_ACK <= 1'b1;
    @(posedge clk);
    pop_ACK <= 1'b0;
```

```
    pop     <= 1'b0;
    $display("[cyc=%0d t=%0t] <<< POP_ONE done (now level=%0d empty=%b)\n",
             cycle, $time, fifo_level, fifo_empty);
endtask


    task automatic diff_packet(input triangle_packet_t got,
                               input triangle_packet_t exp,
                               input int idx);
        $display("---- PACKET %0d DIFF ----", idx);
        if (got.a0         !== exp.a0)         $display("  a0         got=%h exp=%h", got.a0, exp.a0);
        if (got.b0         !== exp.b0)         $display("  b0         got=%h exp=%h", got.b0, exp.b0);
        if (got.a1         !== exp.a1)         $display("  a1         got=%h exp=%h", got.a1, exp.a1);
        if (got.b1         !== exp.b1)         $display("  b1         got=%h exp=%h", got.b1, exp.b1);
        if (got.a2         !== exp.a2)         $display("  a2         got=%h exp=%h", got.a2, exp.a2);
        if (got.b2         !== exp.b2)         $display("  b2         got=%h exp=%h", got.b2, exp.b2);
        if (got.e0_init    !== exp.e0_init)    $display("  e0_init    got=%h exp=%h", got.e0_init, exp.e0_init);
        if (got.e1_init    !== exp.e1_init)    $display("  e1_init    got=%h exp=%h", got.e1_init, exp.e1_init);
        if (got.e2_init    !== exp.e2_init)    $display("  e2_init    got=%h exp=%h", got.e2_init, exp.e2_init);
        if (got.z_at_origin !== exp.z_at_origin) $display("  z_at_origin  got=%h exp=%h", got.z_at_origin,
exp.z_at_origin);
        if (got.z_step_x   !== exp.z_step_x)   $display("  z_step_x    got=%h exp=%h", got.z_step_x, exp.z_step_x);
        if (got.z_step_y   !== exp.z_step_y)   $display("  z_step_y    got=%h exp=%h", got.z_step_y, exp.z_step_y);
        if (got.bbox_xmin  !== exp.bbox_xmin)  $display("  bbox_xmin   got=%0d exp=%0d", got.bbox_xmin,
exp.bbox_xmin);
        if (got.bbox_xmax  !== exp.bbox_xmax)  $display("  bbox_xmax   got=%0d exp=%0d", got.bbox_xmax,
exp.bbox_xmax);
        if (got.bbox_ymin  !== exp.bbox_ymin)  $display("  bbox_ymin   got=%0d exp=%0d", got.bbox_ymin,
exp.bbox_ymin);
        if (got.bbox_ymax  !== exp.bbox_ymax)  $display("  bbox_ymax   got=%0d exp=%0d", got.bbox_ymax,
exp.bbox_ymax);
        if (got.color      !== exp.color)      $display("  color       got=%h exp=%h", got.color, exp.color);
        if (got.front_facing !== exp.front_facing) $display("  front_facing got=%b exp=%b", got.front_facing,
exp.front_facing);
        $display("------------------------");
    endtask


    triangle_packet_t got_pkts [NUM_PACKETS];
```

```
triangle_packet_t exp_pkts [NUM_PACKETS];
logic [31:0]      rdata;

initial begin
   rst             = 1;
   avalon_address   = '0;
   avalon_write     = 1'b0;
   avalon_writedata = '0;
   pop              = 1'b0;
   pop_ACK          = 1'b0;
   repeat (4) @(posedge clk);
   rst = 0;
   @(posedge clk);

   $display("\n#### TEST 1 - status after reset ####");
   avalon_read_word(ADDR_STATUS, rdata);
   check(rdata[5:0] == 6'd0, "level 0 after reset");
   check(rdata[6]   == 1'b1, "empty bit set after reset");
   check(rdata[7]   == 1'b0, "full bit clear after reset");
   check(fifo_empty,         "fifo_empty output high after reset");
   check(!fifo_full,         "fifo_full output low after reset");

   $display("\n#### TEST 2 - push one packet ####");
   submit_packet(0);
   @(posedge clk);
   avalon_read_word(ADDR_STATUS, rdata);
   check(rdata[5:0] == 6'd1, "level 1 after one commit");
   check(rdata[6]   == 1'b0, "empty clear after commit");
   check(pop_available,      "pop_available high after commit");

   $display("\n#### TEST 3 - pop the single packet ####");
   pop_one(got_pkts[0]);
   exp_pkts[0] = expected_packet(0);
   if (got_pkts[0] !== exp_pkts[0]) diff_packet(got_pkts[0], exp_pkts[0], 0);
   check(got_pkts[0] === exp_pkts[0], "packet 0 round-trip content matches");

   @(posedge clk);
   avalon_read_word(ADDR_STATUS, rdata);
   check(rdata[5:0] == 6'd0, "level back to 0 after pop");
```

```
    check(rdata[6]   == 1'b1, "empty bit set after pop");

    $display("\n#### TEST 4 - push %0d packets back-to-back ####", NUM_PACKETS);
    for (int i = 0; i < NUM_PACKETS; i++) begin
        submit_packet(i);
    end
    @(posedge clk);
    avalon_read_word(ADDR_STATUS, rdata);
    check(rdata[5:0] == 6'(NUM_PACKETS),
          $sformatf("level %0d after %0d commits", rdata[5:0], NUM_PACKETS));

    $display("\n#### TEST 5 - drain and verify order ####");
    for (int i = 0; i < NUM_PACKETS; i++) begin
        pop_one(got_pkts[i]);
        exp_pkts[i] = expected_packet(i);
        if (got_pkts[i] !== exp_pkts[i]) diff_packet(got_pkts[i], exp_pkts[i], i);
        check(got_pkts[i] === exp_pkts[i],
              $sformatf("packet %0d order/content mismatch", i));
    end

    @(posedge clk);
    avalon_read_word(ADDR_STATUS, rdata);
    check(rdata[5:0] == 6'd0, "level 0 after full drain");
    check(rdata[6]   == 1'b1, "empty after full drain");

    $display("\n#### TEST 6 - control register roundtrip ####");
    avalon_write_word(ADDR_CONTROL, 32'hDEAD_BEEF);
    @(posedge clk);
    avalon_read_word(ADDR_CONTROL, rdata);
    check(rdata == 32'hDEAD_BEEF, "control register roundtrip");

    $display("\n============================================================");
    if (errors == 0)
        $display("PASS avalon_interface_tb: %0d packets verified", NUM_PACKETS);
    else
        $display("FAIL avalon_interface_tb: %0d errors", errors);
    $display("============================================================");
    $finish;
end
```

```
    initial begin
        #100000;
        $error("avalon_interface_tb timeout");
        $finish;
    end

endmodule
```

```
================================================================
FILE: sim/triangle_fifo_tb.sv
================================================================
```

```
// triangle_fifo_tb.sv

`timescale 1ns/1ps
`include "triangle_packet.svh"

module triangle_fifo_tb;

    localparam int DEPTH       = 64;
    localparam int NUM_PACKETS = 5;

    logic            clk;
    logic            rst;
    logic            push;
    triangle_packet_t push_data;
    logic            pop;
    triangle_packet_t pop_data;
    logic            full, empty;
    logic [5:0]      level;

    triangle_fifo #(.DEPTH(DEPTH)) dut (
        .clk(clk),
        .rst(rst),
        .push(push),
        .push_data(push_data),
        .pop(pop),
```

```
    .pop_data(pop_data),
    .full(full),
    .empty(empty),
    .level(level)
);

initial begin
    clk = 0;
    forever #5 clk = ~clk;
end

int errors = 0;
task automatic check(input bit cond, input string msg);
    if (!cond) begin
        $error("FAIL: %s", msg);
        errors++;
    end
endtask


int              seen_count;
triangle_packet_t seen [NUM_PACKETS];

always_ff @(posedge clk) begin
    if (rst) begin
        seen_count <= 0;
    end else if (pop && !empty) begin
        seen[seen_count] <= pop_data;
        seen_count       <= seen_count + 1;
    end
end

triangle_packet_t sent [NUM_PACKETS];

initial begin
    rst      = 1;
    push     = 0;
    pop      = 0;
    push_data = '0;
```

```
repeat (4) @(posedge clk);
rst = 0;
@(posedge clk);

check(empty, "empty after reset");
check(!full, "not full after reset");
check(level == 6'd0, "level 0 after reset");


for (int i = 0; i < NUM_PACKETS; i++) begin
   sent[i]       = '0;
   sent[i].color  = 8'(i + 8'hA0);
   sent[i].bbox_xmin = 10'(i);
   push      <= 1'b1;
   push_data <= sent[i];
   @(posedge clk);
end
push <= 1'b0;
@(posedge clk);

check(level == 6'(NUM_PACKETS), "level matches push count");
check(!empty, "not empty after pushes");


pop <= 1'b1;
repeat (NUM_PACKETS) @(posedge clk);
pop <= 1'b0;
@(posedge clk);

check(empty, "empty after draining all");
check(seen_count == NUM_PACKETS, "captured NUM_PACKETS");

for (int i = 0; i < NUM_PACKETS; i++) begin
   check(seen[i] === sent[i], $sformatf("order/content mismatch at %0d", i));
end

if (errors == 0) $display("PASS triangle_fifo_tb");
else             $display("FAIL triangle_fifo_tb: %0d errors", errors);
$finish;
```

```
    end

    initial begin
        #10000;
        $error("triangle_fifo_tb timeout");
        $finish;
    end

endmodule
```

```
================================================================
FILE: sim/triangle_dispatcher_tb.sv
================================================================
```

```
// triangle_dispatcher_tb.sv

`timescale 1ns/1ps
`include "triangle_packet.svh"


module triangle_dispatcher_tb;

    localparam int N            = 16;
    localparam int NUM_PACKETS  = 5;
    localparam int BUSY_CYCLES  = 10;

    logic clk;
    logic rst;


    logic           fifo_push;
    triangle_packet_t fifo_push_data;
    logic           fifo_pop;
    triangle_packet_t fifo_pop_data;
    logic           fifo_full;
    logic           fifo_empty;
    logic [5:0]       fifo_level;
```

```
logic            pop;
logic            pop_available;
triangle_packet_t pop_data;
logic            pop_ACK;


logic [N-1:0]     valid_out;
triangle_packet_t packet_out;
logic [N-1:0]     ready_in;

triangle_fifo u_fifo (
    .clk(clk),
    .rst(rst),
    .push(fifo_push),
    .push_data(fifo_push_data),
    .pop(fifo_pop),
    .pop_data(fifo_pop_data),
    .full(fifo_full),
    .empty(fifo_empty),
    .level(fifo_level)
);


assign pop_available = !fifo_empty;
assign pop_data      = fifo_pop_data;
assign fifo_pop      = pop_ACK;

triangle_dispatcher #(.N_PU(N)) u_dispatcher (
    .clk(clk),
    .rst(rst),
    .pop(pop),
    .pop_available(pop_available),
    .pop_data(pop_data),
    .pop_ACK(pop_ACK),
    .valid_out(valid_out),
    .packet_out(packet_out),
    .ready_in(ready_in),
    .block_dispatch(1'b0)
```

```
);

initial begin
   clk = 0;
   forever #5 clk = ~clk;
end


int             busy_cnt [N];
int             accepted [N];
triangle_packet_t got      [N][NUM_PACKETS];

genvar gi;
generate
   for (gi = 0; gi < N; gi++) begin : g_fake_unit
      always_ff @(posedge clk) begin
         if (rst) begin
            busy_cnt[gi] <= 0;
            accepted[gi] <= 0;
         end else if (valid_out[gi] && ready_in[gi]) begin
            got[gi][accepted[gi]] <= packet_out;
            accepted[gi]          <= accepted[gi] + 1;
            busy_cnt[gi]          <= BUSY_CYCLES;
         end else if (busy_cnt[gi] > 0) begin
            busy_cnt[gi] <= busy_cnt[gi] - 1;
         end
      end
      assign ready_in[gi] = (busy_cnt[gi] == 0);
   end
endgenerate


triangle_packet_t sent [NUM_PACKETS];

function automatic triangle_packet_t make_packet(input int idx);
   triangle_packet_t p;
   p             = '0;
   p.bbox_xmin    = 10'(idx * 10);
   p.bbox_xmax    = 10'(idx * 10 + 4);
```

```
    p.bbox_ymin    = 9'(idx);
    p.bbox_ymax    = 9'(idx + 2);
    p.color        = 8'(idx + 8'h40);
    p.front_facing = 1'b1;
    return p;
endfunction

int errors = 0;
task automatic check(input bit cond, input string msg);
    if (!cond) begin
        $error("FAIL: %s", msg);
        errors++;
    end
endtask

initial begin
    int timeout;

    rst           = 1;
    fifo_push     = 0;
    fifo_push_data = '0;
    repeat (4) @(posedge clk);
    rst = 0;
    @(posedge clk);

    check(fifo_empty, "fifo empty after reset");
    for (int u = 0; u < N; u++)
        check(accepted[u] == 0, $sformatf("unit %0d accepted nothing at reset", u));


    for (int i = 0; i < NUM_PACKETS; i++) begin
        sent[i]       = make_packet(i);
        fifo_push     <= 1'b1;
        fifo_push_data <= sent[i];
        @(posedge clk);
    end
    fifo_push     <= 1'b0;
    fifo_push_data <= '0;
```

```
    timeout = 0;
    begin
        bit all_done;
        do begin
            all_done = 1'b1;
            for (int u = 0; u < N; u++)
                if (accepted[u] < NUM_PACKETS) all_done = 1'b0;
            if (all_done) break;
            @(posedge clk);
            timeout++;
        end while (timeout < 5000);
    end

    for (int u = 0; u < N; u++)
        check(accepted[u] == NUM_PACKETS,
              $sformatf("unit %0d accepted %0d/%0d",
                        u, accepted[u], NUM_PACKETS));
    check(fifo_empty, "fifo should drain fully");


    for (int i = 0; i < NUM_PACKETS; i++) begin
        for (int u = 0; u < N; u++) begin
            check(got[u][i] === sent[i],
                  $sformatf("unit %0d packet %0d mismatch", u, i));
            check(got[u][i] === got[0][i],
                  $sformatf("unit %0d disagreed with unit 0 on packet %0d",
                            u, i));
        end
    end

    if (errors == 0)
        $display("PASS triangle_dispatcher_tb: %0d packets to %0d units", NUM_PACKETS, N);
    else
        $display("FAIL triangle_dispatcher_tb: %0d errors", errors);
    $finish;
end

initial begin
```

```
        #50000;
        $error("triangle_dispatcher_tb timeout");
        $finish;
    end

endmodule
```

```
================================================================
FILE: sim/pixel_unit_tb.sv
================================================================
```

```
// pixel_unit_tb.sv

`timescale 1ns/1ps
`include "triangle_packet.svh"

module pixel_unit_tb;

    localparam int PU_ID    = 4'd3;
    localparam int FB_DEPTH = 4096;
    localparam int Z_DEPTH  = 4096;

    logic clk, rst;

    logic seed_valid_in;
    logic signed [31:0] seed_e0_in, seed_e1_in, seed_e2_in, seed_z_in;
    logic signed [31:0] a0_in, a1_in, a2_in, z_step_x_in;
    logic signed [31:0] b0_in, b1_in, b2_in, z_step_y_in;
    logic [7:0] color_in, col_base_in, row_base_in, last_row_in, last_col_in;

    logic ready;
    logic seed_valid_out;
    logic signed [31:0] seed_e0_out, seed_e1_out, seed_e2_out, seed_z_out;

    logic p_write;
    logic [7:0] p_col, p_row, p_color;
    logic [15:0] p_depth;
```

```
logic [11:0] vga_r_addr;
logic vga_r_buf_sel, fb_write_sel;
logic [7:0] vga_r_data;

pixel_unit #(
    .PU_ID(PU_ID),
    .IS_LAST_PU(1'b0),
    .FB_DEPTH(FB_DEPTH),
    .Z_DEPTH(Z_DEPTH),


    .DO_INIT_CLEAR(1'b0)
) dut (
    .clk(clk),
    .rst(rst),
    .ready(ready),
    .z_clear_start(1'b0),

    .seed_valid_in(seed_valid_in),
    .seed_e0_in(seed_e0_in),
    .seed_e1_in(seed_e1_in),
    .seed_e2_in(seed_e2_in),
    .seed_z_in(seed_z_in),

    .a0_in(a0_in),
    .a1_in(a1_in),
    .a2_in(a2_in),
    .z_step_x_in(z_step_x_in),

    .b0_in(b0_in),
    .b1_in(b1_in),
    .b2_in(b2_in),
    .z_step_y_in(z_step_y_in),

    .color_in(color_in),
    .col_base_in(col_base_in),
    .row_base_in(row_base_in),
    .last_row_in(last_row_in),
    .last_col_in(last_col_in),
```

```
    .seed_valid_out(seed_valid_out),
    .seed_e0_out(seed_e0_out),
    .seed_e1_out(seed_e1_out),
    .seed_e2_out(seed_e2_out),
    .seed_z_out(seed_z_out),

    .p_write(p_write),
    .p_col(p_col),
    .p_row(p_row),
    .p_color(p_color),
    .p_depth(p_depth),

    .vga_r_addr(vga_r_addr),
    .vga_r_buf_sel(vga_r_buf_sel),
    .vga_r_data(vga_r_data),
    .fb_write_sel(fb_write_sel)
);

logic last_seed_valid_in;
logic last_seed_valid_out;
logic [31:0] last_seed_e0_out;

pixel_unit #(
    .PU_ID(PU_ID),
    .IS_LAST_PU(1'b1),
    .FB_DEPTH(FB_DEPTH),
    .Z_DEPTH(Z_DEPTH),
    .DO_INIT_CLEAR(1'b0)
) dut_last (
    .clk(clk),
    .rst(rst),
    .ready(),
    .z_clear_start(1'b0),

    .seed_valid_in(last_seed_valid_in),
    .seed_e0_in(seed_e0_in),
    .seed_e1_in(seed_e1_in),
    .seed_e2_in(seed_e2_in),
```

```
    .seed_z_in(seed_z_in),

    .a0_in(a0_in),
    .a1_in(a1_in),
    .a2_in(a2_in),
    .z_step_x_in(z_step_x_in),

    .b0_in(b0_in),
    .b1_in(b1_in),
    .b2_in(b2_in),
    .z_step_y_in(z_step_y_in),

    .color_in(color_in),
    .col_base_in(col_base_in),
    .row_base_in(row_base_in),
    .last_row_in(last_row_in),
    .last_col_in(last_col_in),

    .seed_valid_out(last_seed_valid_out),
    .seed_e0_out(last_seed_e0_out),
    .seed_e1_out(),
    .seed_e2_out(),
    .seed_z_out(),

    .p_write(),
    .p_col(),
    .p_row(),
    .p_color(),
    .p_depth(),

    .vga_r_addr(12'h000),
    .vga_r_buf_sel(1'b0),
    .vga_r_data(),
    .fb_write_sel(1'b0)
);

initial clk = 1'b0;
always #5 clk = ~clk;
```

```
int errors = 0;

task automatic check(input bit cond, input string msg);
   if (!cond) begin
      $error("FAIL: %s", msg);
      errors++;
   end
endtask

int pix_n;
logic [7:0]  pix_col   [0:511];
logic [7:0]  pix_row   [0:511];
logic [7:0]  pix_color [0:511];
logic [15:0] pix_depth [0:511];

always_ff @(posedge clk) begin
   if (rst) begin
      pix_n <= 0;
   end else if (p_write) begin
      $display("[%0t] p_write: col=%0d row=%0d color=%h depth=%h idx=%0d",
            $time, p_col, p_row, p_color, p_depth, pix_n);
      pix_col[pix_n]   <= p_col;
      pix_row[pix_n]   <= p_row;
      pix_color[pix_n] <= p_color;
      pix_depth[pix_n] <= p_depth;
      pix_n            <= pix_n + 1;
   end
end

task automatic z_clear_to_far();
   for (int i = 0; i < Z_DEPTH; i++) begin
      dut.z_mem[i] = 16'hFFFF;
   end
endtask

task automatic z_set_all(input logic [15:0] val);
   for (int i = 0; i < Z_DEPTH; i++) begin
      dut.z_mem[i] = val;
   end
```

```
endtask

task automatic fb_clear();
   for (int i = 0; i < FB_DEPTH; i++) begin
      dut.fb_a_mem[i] = 8'h00;
      dut.fb_b_mem[i] = 8'h00;
   end
endtask

task automatic clear_collector();
   begin
      pix_n = 0;
      #1;
   end
endtask

task automatic drive_idle();
   begin
      seed_valid_in      = 1'b0;
      last_seed_valid_in = 1'b0;

      seed_e0_in = '0;
      seed_e1_in = '0;
      seed_e2_in = '0;
      seed_z_in  = '0;

      a0_in = '0;
      a1_in = '0;
      a2_in = '0;
      z_step_x_in = '0;

      b0_in = '0;
      b1_in = '0;
      b2_in = '0;
      z_step_y_in = '0;

      color_in    = '0;
      col_base_in = '0;
      row_base_in = '0;
```

```
        last_row_in = '0;
        last_col_in = '0;

        vga_r_addr    = '0;
        vga_r_buf_sel = 1'b0;
        fb_write_sel  = 1'b0;
    end
endtask

task automatic launch_main();
    begin
        @(negedge clk);
        seed_valid_in = 1'b1;

        @(posedge clk);
        #1;

        seed_valid_in = 1'b0;

        check(ready === 1'b0, "ready low immediately after seed accepted");
    end
endtask

task automatic launch_last();
    begin
        @(negedge clk);
        last_seed_valid_in = 1'b1;

        @(posedge clk);
        #1;

        last_seed_valid_in = 1'b0;
    end
endtask

task automatic wait_done();
    begin
        wait (ready === 1'b1);
        repeat (4) @(posedge clk);
```

```
        #1;
    end
endtask

initial begin
    drive_idle();
    fb_clear();
    z_clear_to_far();

    rst = 1'b1;
    repeat (4) @(posedge clk);
    #1;
    rst = 1'b0;
    repeat (2) @(posedge clk);
    #1;

    $display("--- Test 1: reset ---");
    check(ready === 1'b1, "ready high after reset");
    check(p_write === 1'b0, "p_write low at idle");
    check(seed_valid_out === 1'b0, "seed_valid_out low at idle");


    $display("--- Test 2: 5-row column, all-positive edges ---");
    wait_done();
    z_clear_to_far();
    clear_collector();

    seed_e0_in = 32'sd1000;
    seed_e1_in = 32'sd2000;
    seed_e2_in = 32'sd3000;
    seed_z_in  = 32'sh0100;

    a0_in = 32'sd10;
    a1_in = 32'sd20;
    a2_in = 32'sd30;
    z_step_x_in = 32'sd1;

    b0_in = 32'sd0;
    b1_in = 32'sd0;
```

```
b2_in = 32'sd0;
z_step_y_in = 32'sd0;

color_in    = 8'hA5;
col_base_in = 8'h13;
row_base_in = 8'd0;
last_row_in = 8'd4;
last_col_in = 8'h13;

launch_main();

check(seed_valid_out === 1'b1, "seed_valid_out high on accept cycle");
check(seed_e0_out === 32'sd1010, $sformatf("seed_e0_out=%0d expected 1010", seed_e0_out));
check(seed_e1_out === 32'sd2020, $sformatf("seed_e1_out=%0d expected 2020", seed_e1_out));
check(seed_e2_out === 32'sd3030, $sformatf("seed_e2_out=%0d expected 3030", seed_e2_out));
check(seed_z_out  === 32'sh0101, $sformatf("seed_z_out=%h expected 0101", seed_z_out));

@(posedge clk);
#1;
check(seed_valid_out === 1'b0, "seed_valid_out one-cycle pulse");

wait_done();

check(pix_n == 5, $sformatf("expected 5 pixels, got %0d", pix_n));
for (int i = 0; i < 5 && i < pix_n; i++) begin
   check(pix_col[i]   === 8'h13, $sformatf("pix[%0d].col=%h expected 13", i, pix_col[i]));
   check(pix_row[i]   === 8'(i), $sformatf("pix[%0d].row=%0d expected %0d", i, pix_row[i], i));
   check(pix_color[i] === 8'hA5, $sformatf("pix[%0d].color=%h expected A5", i, pix_color[i]));
   check(pix_depth[i] === 16'h0100, $sformatf("pix[%0d].depth=%h expected 0100", i, pix_depth[i]));
end


$display("--- Test 3: negative edge culls all pixels ---");
wait_done();
z_clear_to_far();
clear_collector();

seed_e0_in = -32'sd1;
seed_e1_in = 32'sd1;
```

```
seed_e2_in = 32'sd1;
seed_z_in  = 32'sh0100;

a0_in = 32'sd0;
a1_in = 32'sd0;
a2_in = 32'sd0;
z_step_x_in = 32'sd0;

b0_in = 32'sd0;
b1_in = 32'sd0;
b2_in = 32'sd0;
z_step_y_in = 32'sd0;

color_in    = 8'h55;
col_base_in = 8'h03;
row_base_in = 8'd0;
last_row_in = 8'd3;
last_col_in = 8'h03;

launch_main();
wait_done();

check(pix_n == 0, $sformatf("inside cull: expected 0 pixels, got %0d", pix_n));



$display("--- Test 4: z-test culls farther fragment ---");
wait_done();
z_set_all(16'h0050);
clear_collector();

seed_e0_in = 32'sd1;
seed_e1_in = 32'sd1;
seed_e2_in = 32'sd1;
seed_z_in  = 32'sh0100;

a0_in = 32'sd0;
a1_in = 32'sd0;
a2_in = 32'sd0;
z_step_x_in = 32'sd0;
```

```
b0_in = 32'sd0;
b1_in = 32'sd0;
b2_in = 32'sd0;
z_step_y_in = 32'sd0;

color_in    = 8'h66;
col_base_in = 8'h03;
row_base_in = 8'd0;
last_row_in = 8'd3;
last_col_in = 8'h03;

launch_main();
wait_done();

check(pix_n == 0, $sformatf("z-test reject: expected 0 pixels, got %0d", pix_n));


$display("--- Test 4b: z-test admits closer fragment ---");
wait_done();
z_set_all(16'h0050);
clear_collector();

seed_e0_in = 32'sd1;
seed_e1_in = 32'sd1;
seed_e2_in = 32'sd1;
seed_z_in  = 32'sh0010;

color_in    = 8'h77;
col_base_in = 8'h03;
row_base_in = 8'd0;
last_row_in = 8'd3;
last_col_in = 8'h03;

launch_main();
wait_done();

check(pix_n == 4, $sformatf("z-test admit: expected 4 pixels, got %0d", pix_n));
```

```
$display("--- Test 5: IS_LAST_PU instance does not forward ---");
wait_done();

seed_e0_in = 32'sd1;
seed_e1_in = 32'sd1;
seed_e2_in = 32'sd1;
seed_z_in  = 32'sh0100;

a0_in = 32'sd5;
a1_in = 32'sd5;
a2_in = 32'sd5;
z_step_x_in = 32'sd5;

b0_in = 32'sd0;
b1_in = 32'sd0;
b2_in = 32'sd0;
z_step_y_in = 32'sd0;

color_in    = 8'h88;
col_base_in = 8'h03;
row_base_in = 8'd0;
last_row_in = 8'd2;
last_col_in = 8'h03;

launch_last();

check(last_seed_valid_out === 1'b0,
    "IS_LAST_PU: seed_valid_out stays low on accept cycle");

repeat (10) @(posedge clk);
#1;


$display("--- Test 6: back-to-back ---");
wait_done();
z_clear_to_far();
clear_collector();
```

```
seed_e0_in = 32'sd1;
seed_e1_in = 32'sd1;
seed_e2_in = 32'sd1;
seed_z_in  = 32'sh0080;

a0_in = 32'sd5;
a1_in = 32'sd5;
a2_in = 32'sd5;
z_step_x_in = 32'sd0;

b0_in = 32'sd0;
b1_in = 32'sd0;
b2_in = 32'sd0;
z_step_y_in = 32'sd0;

color_in    = 8'h11;
col_base_in = 8'h03;
row_base_in = 8'd0;
last_row_in = 8'd2;
last_col_in = 8'h03;

launch_main();
wait_done();

color_in    = 8'h22;
col_base_in = 8'h13;
row_base_in = 8'd100;
last_row_in = 8'd103;
last_col_in = 8'h13;
seed_z_in   = 32'sh0040;

launch_main();
wait_done();

check(pix_n == 7, $sformatf("back-to-back: expected 7 pixels total, got %0d", pix_n));

for (int i = 0; i < 3 && i < pix_n; i++) begin
   check(pix_color[i] === 8'h11, $sformatf("first-tri pix[%0d] color", i));
   check(pix_col[i]   === 8'h03, $sformatf("first-tri pix[%0d] col", i));
```

```
    check(pix_row[i]   === 8'(i), $sformatf("first-tri pix[%0d] row", i));
end

for (int i = 0; i < 4 && (i + 3) < pix_n; i++) begin
    check(pix_color[i+3] === 8'h22, $sformatf("second-tri pix[%0d] color", i));
    check(pix_col[i+3]   === 8'h13, $sformatf("second-tri pix[%0d] col", i));
    check(pix_row[i+3]   === 8'(100 + i), $sformatf("second-tri pix[%0d] row", i));
end


$display("--- Test 7: multi-column iteration (3 cols at stride 16) ---");
wait_done();
z_clear_to_far();
clear_collector();

seed_e0_in = 32'sd100;
seed_e1_in = 32'sd200;
seed_e2_in = 32'sd300;
seed_z_in  = 32'sh0040;

a0_in = 32'sd2;
a1_in = 32'sd3;
a2_in = 32'sd5;
z_step_x_in = 32'sd0;

b0_in = 32'sd0;
b1_in = 32'sd0;
b2_in = 32'sd0;
z_step_y_in = 32'sd0;

color_in    = 8'h7C;
col_base_in = 8'h03;
row_base_in = 8'd0;
last_row_in = 8'd4;
last_col_in = 8'd47;

launch_main();
wait_done();
```

```
$display("Test 7 result: pix_n=%0d col_base_q=%0d e0_col_top=%0d e1_col_top=%0d e2_col_top=%0d",
        pix_n, dut.col_base_q, dut.e0_col_top, dut.e1_col_top, dut.e2_col_top);

for (int i = 0; i < pix_n && i < 20; i++) begin
   $display("  pix[%0d]: col=%0d row=%0d color=%h depth=%h",
          i, pix_col[i], pix_row[i], pix_color[i], pix_depth[i]);
end

check(pix_n == 15, $sformatf("multi-col: expected 15 pixels, got %0d", pix_n));

begin
   logic [7:0] expected_cols [0:2];
   expected_cols[0] = 8'd3;
   expected_cols[1] = 8'd19;
   expected_cols[2] = 8'd35;

   for (int c = 0; c < 3; c++) begin
      for (int r = 0; r < 5; r++) begin
         int idx;
         idx = c * 5 + r;

         if (idx < pix_n) begin
            check(pix_col[idx] === expected_cols[c],
                $sformatf("multi-col pix[%0d] col=%0d expected %0d",
                      idx, pix_col[idx], expected_cols[c]));
            check(pix_row[idx] === 8'(r),
                $sformatf("multi-col pix[%0d] row=%0d expected %0d",
                      idx, pix_row[idx], r));
            check(pix_color[idx] === 8'h7C,
                $sformatf("multi-col pix[%0d] color expected 7C", idx));
         end
      end
   end
end

check(dut.col_base_q === 8'd35,
     $sformatf("multi-col: dut.col_base_q=%0d expected 35", dut.col_base_q));

check(dut.e0_col_top === 32'sd100 + 32'sd32 * 32'sd2,
```

```
            $sformatf("multi-col: e0_col_top=%0d expected %0d",
                dut.e0_col_top, 100 + 32*2));

        check(dut.e1_col_top === 32'sd200 + 32'sd32 * 32'sd3,
            $sformatf("multi-col: e1_col_top=%0d expected %0d",
                dut.e1_col_top, 200 + 32*3));

        check(dut.e2_col_top === 32'sd300 + 32'sd32 * 32'sd5,
            $sformatf("multi-col: e2_col_top=%0d expected %0d",
                dut.e2_col_top, 300 + 32*5));

        if (errors == 0)
          $display("PASS pixel_unit_tb");
        else
          $display("FAIL pixel_unit_tb: %0d errors", errors);

        $finish;
    end

    initial begin
        #200000;
        $error("pixel_unit_tb timed out");
        $finish;
    end

endmodule


================================================================
FILE: sim/systolic_chain_tb.sv
================================================================

// systolic_chain_tb.sv

`timescale 1ns/1ps
`include "triangle_packet.svh"

module systolic_chain_tb;
```

```
localparam int N_PU           = 4;
localparam int N_ROWS         = 5;
localparam int N_COLS_PER_PU  = 3;
localparam int LAST_COL       = 35;
localparam int FB_DEPTH       = 4096;
localparam int Z_DEPTH        = 4096;

logic clk, rst;


logic              chain_seed_valid [N_PU];
logic signed [31:0] chain_seed_e0    [N_PU];
logic signed [31:0] chain_seed_e1    [N_PU];
logic signed [31:0] chain_seed_e2    [N_PU];
logic signed [31:0] chain_seed_z     [N_PU];

logic              sv_out  [N_PU];
logic signed [31:0] se0_out [N_PU];
logic signed [31:0] se1_out [N_PU];
logic signed [31:0] se2_out [N_PU];
logic signed [31:0] sz_out  [N_PU];


logic signed [31:0] a0_b, a1_b, a2_b, z_step_x_b;
logic signed [31:0] b0_b, b1_b, b2_b, z_step_y_b;
logic [7:0]         color_b, row_base_b, last_row_b, last_col_b;
logic [7:0]         col_base_for_pu [N_PU];


logic              tb_seed_valid;
logic signed [31:0] tb_seed_e0, tb_seed_e1, tb_seed_e2, tb_seed_z;

assign chain_seed_valid[0] = tb_seed_valid;
assign chain_seed_e0[0]    = tb_seed_e0;
assign chain_seed_e1[0]    = tb_seed_e1;
assign chain_seed_e2[0]    = tb_seed_e2;
assign chain_seed_z[0]     = tb_seed_z;
```

```
logic [N_PU-1:0] pu_ready;
logic [N_PU-1:0] p_write;
logic [7:0]      p_col   [N_PU];
logic [7:0]      p_row   [N_PU];
logic [7:0]      p_color [N_PU];
logic [15:0]     p_depth [N_PU];


logic [11:0] vga_r_addr;
logic        vga_r_buf_sel, fb_write_sel;
assign vga_r_addr     = '0;
assign vga_r_buf_sel  = 1'b0;
assign fb_write_sel   = 1'b0;

genvar gi;
generate
   for (gi = 0; gi < N_PU; gi++) begin : g_pu

      assign col_base_for_pu[gi] = 8'(gi);

      pixel_unit #(
         .PU_ID      (gi),
         .IS_LAST_PU ((gi == N_PU - 1) ? 1'b1 : 1'b0),
         .FB_DEPTH   (FB_DEPTH),
         .Z_DEPTH    (Z_DEPTH),

         .DO_INIT_CLEAR(1'b0)
      ) u_pu (
         .clk           (clk),
         .rst           (rst),
         .ready         (pu_ready[gi]),
         .z_clear_start  (1'b0),
         .seed_valid_in  (chain_seed_valid[gi]),
         .seed_e0_in     (chain_seed_e0[gi]),
         .seed_e1_in     (chain_seed_e1[gi]),
         .seed_e2_in     (chain_seed_e2[gi]),
         .seed_z_in      (chain_seed_z[gi]),
         .a0_in          (a0_b),
         .a1_in          (a1_b),
```

```
            .a2_in          (a2_b),
            .z_step_x_in    (z_step_x_b),
            .b0_in          (b0_b),
            .b1_in          (b1_b),
            .b2_in          (b2_b),
            .z_step_y_in    (z_step_y_b),
            .color_in       (color_b),
            .col_base_in    (col_base_for_pu[gi]),
            .row_base_in    (row_base_b),
            .last_row_in    (last_row_b),
            .last_col_in    (last_col_b),
            .seed_valid_out (sv_out[gi]),
            .seed_e0_out    (se0_out[gi]),
            .seed_e1_out    (se1_out[gi]),
            .seed_e2_out    (se2_out[gi]),
            .seed_z_out     (sz_out[gi]),
            .p_write        (p_write[gi]),
            .p_col          (p_col[gi]),
            .p_row          (p_row[gi]),
            .p_color        (p_color[gi]),
            .p_depth        (p_depth[gi]),
            .vga_r_addr     (vga_r_addr),
            .vga_r_buf_sel  (vga_r_buf_sel),
            .vga_r_data     (),
            .fb_write_sel   (fb_write_sel)
        );

        if (gi < N_PU - 1) begin : g_chain
            assign chain_seed_valid[gi + 1] = sv_out[gi];
            assign chain_seed_e0   [gi + 1] = se0_out[gi];
            assign chain_seed_e1   [gi + 1] = se1_out[gi];
            assign chain_seed_e2   [gi + 1] = se2_out[gi];
            assign chain_seed_z    [gi + 1] = sz_out[gi];
        end
    end
endgenerate


initial clk = 0;
```

```
always #5 clk = ~clk;

int errors = 0;
task automatic check(input bit cond, input string msg);
   if (!cond) begin
      $error("FAIL: %s", msg);
      errors++;
   end
endtask


int cycle_num;
int pix_count       [N_PU];
int pu_first_cycle  [N_PU];
int pu_last_cycle   [N_PU];

initial begin
   cycle_num = 0;
   for (int u = 0; u < N_PU; u++) begin
      pix_count[u]      = 0;
      pu_first_cycle[u] = -1;
      pu_last_cycle[u]  = -1;
   end
end

always @(posedge clk) begin
   if (rst) begin
      cycle_num <= 0;
   end else begin
      cycle_num <= cycle_num + 1;
      for (int u = 0; u < N_PU; u++) begin
         if (p_write[u]) begin
            if (pu_first_cycle[u] == -1)
               pu_first_cycle[u] <= cycle_num;
            pu_last_cycle[u] <= cycle_num;
            pix_count[u] <= pix_count[u] + 1;
         end
      end
   end
```

```
end


bit diagonal_checked;
initial diagonal_checked = 1'b0;

always @(posedge clk) begin
   if (!rst && p_write[N_PU - 1] && !diagonal_checked) begin
      diagonal_checked <= 1'b1;
      for (int u = 0; u < N_PU; u++) begin
         check(p_write[u] === 1'b1,
            $sformatf("diagonal: PU%0d not writing on fill cycle", u));
         check(p_row[u] === 8'(N_PU - 1 - u),
            $sformatf("diagonal: PU%0d row=%0d expected %0d",
                  u, p_row[u], N_PU - 1 - u));
         check(p_col[u] === 8'(u),
            $sformatf("diagonal: PU%0d col=%0d expected %0d",
                  u, p_col[u], u));
      end
   end
end


always @(posedge clk) begin
   if (!rst && sv_out[N_PU - 1])
      $error("FAIL: PU(N-1) asserted seed_valid_out (IS_LAST_PU broken)");
end


task automatic z_clear_all_pus_to_far();
   for (int i = 0; i < Z_DEPTH; i++) begin
      g_pu[0].u_pu.z_mem[i] = 16'hFFFF;
      g_pu[1].u_pu.z_mem[i] = 16'hFFFF;
      g_pu[2].u_pu.z_mem[i] = 16'hFFFF;
      g_pu[3].u_pu.z_mem[i] = 16'hFFFF;
   end
endtask

task automatic drive_idle();
```

```
    tb_seed_valid = 1'b0;
    tb_seed_e0    = '0;
    tb_seed_e1    = '0;
    tb_seed_e2    = '0;
    tb_seed_z     = '0;
    a0_b          = '0;
    a1_b          = '0;
    a2_b          = '0;
    z_step_x_b    = '0;
    b0_b          = '0;
    b1_b          = '0;
    b2_b          = '0;
    z_step_y_b    = '0;
    color_b       = '0;
    row_base_b    = '0;
    last_row_b    = '0;
    last_col_b    = '0;
endtask

initial begin
    drive_idle();
    rst = 1'b1;
    repeat (4) @(posedge clk);
    rst = 1'b0;
    @(posedge clk);

    z_clear_all_pus_to_far();


    a0_b       <= 32'sd1;
    a1_b       <= 32'sd1;
    a2_b       <= 32'sd1;
    z_step_x_b <= 32'sd0;
    b0_b       <= 32'sd0;
    b1_b       <= 32'sd0;
    b2_b       <= 32'sd0;
    z_step_y_b <= 32'sd0;
    color_b    <= 8'h7E;
    row_base_b <= 8'd0;
```

```
last_row_b <= 8'(N_ROWS - 1);
last_col_b <= 8'(LAST_COL);

tb_seed_e0 <= 32'sd100;
tb_seed_e1 <= 32'sd200;
tb_seed_e2 <= 32'sd300;
tb_seed_z  <= 32'sh0040;
@(posedge clk);


tb_seed_valid <= 1'b1;
@(posedge clk);
tb_seed_valid <= 1'b0;


repeat (N_PU + N_COLS_PER_PU * N_ROWS + 16) @(posedge clk);

for (int u = 0; u < N_PU; u++) begin
   check(pix_count[u] == N_COLS_PER_PU * N_ROWS,
        $sformatf("PU%0d emitted %0d pixels, expected %0d",
                u, pix_count[u], N_COLS_PER_PU * N_ROWS));
   check(pu_ready[u] === 1'b1,
        $sformatf("PU%0d not back to ready after drain", u));
end


for (int u = 1; u < N_PU; u++) begin
   check(pu_first_cycle[u] - pu_first_cycle[u-1] == 1,
        $sformatf("stagger PU%0d-PU%0d: first cycles %0d, %0d",
                u-1, u, pu_first_cycle[u-1], pu_first_cycle[u]));
   check(pu_last_cycle[u] - pu_last_cycle[u-1] == 1,
        $sformatf("drain PU%0d-PU%0d: last cycles %0d, %0d",
                u-1, u, pu_last_cycle[u-1], pu_last_cycle[u]));
end

check(diagonal_checked,
     "diagonal cycle witness never fired (chain may not have filled)");

if (errors == 0)
```

```
        $display("PASS systolic_chain_tb: %0d-PU chain, %0d cols x %0d rows each",
                 N_PU, N_COLS_PER_PU, N_ROWS);
      else
        $display("FAIL systolic_chain_tb: %0d errors", errors);
      $finish;
    end

    initial begin
      #200000;
      $error("systolic_chain_tb timed out");
      $finish;
    end

endmodule
```

```
================================================================
FILE: sim/rasterizer_top_tb.sv
================================================================
```

```
// rasterizer_top_tb.sv

`timescale 1ns/1ps


module rasterizer_top_tb;

    logic        clk;
    logic        rst;
    logic [6:0]  avs_address;
    logic        avs_write;
    logic [31:0] avs_writedata;
    logic [31:0] avs_readdata;
    logic        avs_waitrequest;
    logic [7:0]  VGA_R, VGA_G, VGA_B;
    logic        VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n;

    rasterizer_top dut (
        .clk             (clk),
```

```
    .rst              (rst),
    .avs_address      (avs_address),
    .avs_write        (avs_write),
    .avs_writedata    (avs_writedata),
    .avs_readdata     (avs_readdata),
    .avs_waitrequest  (avs_waitrequest),
    .VGA_R            (VGA_R),
    .VGA_G            (VGA_G),
    .VGA_B            (VGA_B),
    .VGA_CLK          (VGA_CLK),
    .VGA_HS           (VGA_HS),
    .VGA_VS           (VGA_VS),
    .VGA_BLANK_n      (VGA_BLANK_n),
    .VGA_SYNC_n       (VGA_SYNC_n)
);

initial begin
    clk = 0;
    forever #5 clk = ~clk;
end

initial begin
    rst          = 1;
    avs_address   = '0;
    avs_write     = 1'b0;
    avs_writedata = '0;
    repeat (5) @(posedge clk);
    rst = 0;

    repeat (2000) @(posedge clk);
    $display("PASS rasterizer_top_tb elaborated + idled");
    $finish;
end

initial begin
    #100000;
    $error("rasterizer_top_tb timeout");
    $finish;
end
```

```
endmodule
```

```
================================================================
FILE: sim/vga_framebuffer_tb.sv
================================================================
```

```
// vga_framebuffer_tb.sv

`timescale 1ns/1ps

module vga_framebuffer_tb;

    logic clk, reset;

    logic [7:0] pu_vga_data [0:15];

    logic [11:0] vga_r_addr;
    logic        vga_r_buf_sel;
    logic        fb_write_sel;
    logic        frame_done;

    logic [7:0] VGA_R, VGA_G, VGA_B;
    logic VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n;

    vga_framebuffer dut (
        .clk(clk),
        .reset(reset),

        .pu_vga_data(pu_vga_data),

        .vga_r_addr(vga_r_addr),
        .vga_r_buf_sel(vga_r_buf_sel),

        .fb_write_sel(fb_write_sel),
        .frame_done(frame_done),

        .VGA_R(VGA_R),
```

```
    .VGA_G(VGA_G),
    .VGA_B(VGA_B),
    .VGA_CLK(VGA_CLK),
    .VGA_HS(VGA_HS),
    .VGA_VS(VGA_VS),
    .VGA_BLANK_n(VGA_BLANK_n),
    .VGA_SYNC_n(VGA_SYNC_n)
);


initial clk = 1'b0;
always #10 clk = ~clk;

int errors = 0;

task automatic check(input bit cond, input string msg);
    if (!cond) begin
        $error("FAIL: %s", msg);
        errors++;
    end
endtask

function automatic [23:0] rgb332_to_rgb888(input logic [7:0] c);
    rgb332_to_rgb888 = {
        {c[7:5], c[7:5], c[7:6]},
        {c[4:2], c[4:2], c[4:3]},
        {c[1:0], c[1:0], c[1:0], c[1:0]}
    };
endfunction

task automatic drive_pu_pattern();
    for (int i = 0; i < 16; i++) begin

        pu_vga_data[i] = {i[3:0], i[3:0]};
    end
endtask

task automatic wait_hv(input int h, input int v);
    begin
```

```
        wait (dut.hcount == h && dut.vcount == v);
        #1;
    end
endtask

task automatic check_pixel(input int fb_x, input int fb_y);
    int screen_x;
    int h_target;
    int v_target;
    int expected_pu;
    logic [11:0] expected_addr;
    logic [7:0]  expected_color;
    logic [23:0] expected_rgb;

    begin


        screen_x = 64 + (2 * fb_x);


        h_target = screen_x;


        v_target = 2 * fb_y;

        wait_hv(h_target, v_target);

        expected_pu    = fb_x[3:0];
        expected_addr  = {fb_x[7:4], fb_y[7:0]};
        expected_color = {expected_pu[3:0], expected_pu[3:0]};
        expected_rgb   = rgb332_to_rgb888(expected_color);

        check(vga_r_addr === expected_addr,
            $sformatf("addr fb_x=%0d fb_y=%0d got=%h expected=%h",
                    fb_x, fb_y, vga_r_addr, expected_addr));


        @(posedge clk);
        #1;
```

```
        check({VGA_R, VGA_G, VGA_B} === expected_rgb,
            $sformatf("RGB fb_x=%0d fb_y=%0d PU=%0d got=%h expected=%h",
                fb_x, fb_y, expected_pu,
                {VGA_R, VGA_G, VGA_B}, expected_rgb));

        $display("PASS pixel fb_x=%0d fb_y=%0d PU=%0d addr=%h RGB=%h",
              fb_x, fb_y, expected_pu, expected_addr, expected_rgb);
    end
endtask

task automatic check_black(input int screen_x, input int screen_y);
    begin
        wait_hv(screen_x, screen_y);

        @(posedge clk);
        #1;

        check({VGA_R, VGA_G, VGA_B} === 24'h000000,
            $sformatf("black region screen_x=%0d screen_y=%0d got RGB=%h",
                screen_x, screen_y, {VGA_R, VGA_G, VGA_B}));

        $display("PASS black screen_x=%0d screen_y=%0d", screen_x, screen_y);
    end
endtask

initial begin
    drive_pu_pattern();

    fb_write_sel = 1'b0;

    reset = 1'b1;
    repeat (5) @(posedge clk);
    #1;
    reset = 1'b0;

    repeat (10) @(posedge clk);
    #1;
```

```
$display("--- Test 1: buffer select ---");
check(vga_r_buf_sel === 1'b1,
    "vga_r_buf_sel should equal ~fb_write_sel when fb_write_sel=0");

fb_write_sel = 1'b1;
#1;
check(vga_r_buf_sel === 1'b0,
    "vga_r_buf_sel should equal ~fb_write_sel when fb_write_sel=1");

fb_write_sel = 1'b0;
#1;

$display("--- Test 2: black bars ---");
check_black(10, 20);
check_black(600, 20);

$display("--- Test 3: active framebuffer pixels ---");
check_pixel(0,   0);
check_pixel(1,   0);
check_pixel(15,  0);
check_pixel(16,  0);
check_pixel(37,  5);
check_pixel(255, 239);

$display("--- Test 4: frame_done pulse ---");
wait (frame_done === 1'b1);
#1;
check(frame_done === 1'b1, "frame_done should pulse high at end of frame");

@(posedge clk);
#1;
check(frame_done === 1'b0, "frame_done should be one-cycle pulse");

if (errors == 0)
   $display("PASS vga_framebuffer_tb");
else
   $display("FAIL vga_framebuffer_tb: %0d errors", errors);

$finish;
```

```
    end

    initial begin
        #50_000_000;
        $error("vga_framebuffer_tb timed out");
        $finish;
    end

endmodule
```

```
================================================================
FILE: software/kernel/rasterizer.h
================================================================
```

```
// rasterizer.h

#ifndef RASTERIZER_H
#define RASTERIZER_H


#endif
```

```
================================================================
FILE: software/kernel/avalon_kernel.h
================================================================
```

```
// avalon_kernel.h

#ifndef _RASTERIZER_H
#define _RASTERIZER_H

#include <linux/ioctl.h>
#include <linux/types.h>

typedef struct {
    __s32 a0, b0, c0;
    __s32 a1, b1, c1;
```

```
    __s32 a2, b2, c2;
    __s32 e0_init;
    __s32 e1_init;
    __s32 e2_init;
    __s32 z_origin;
    __s32 z_step_x;
    __s32 z_step_y;
    __u32 bbox_packed;
    __u32 flags_color;
} triangle_packet_t;


typedef struct {
    __u32 fifo_level;
    __u32 fifo_empty;
    __u32 fifo_full;
    __u32 swap_busy;
} rasterizer_status_t;


typedef struct {
    __u32 irq_enable;
    __u32 low_watermark;
} rasterizer_control_t;


typedef union {
    triangle_packet_t  packet;
    rasterizer_status_t  status;
    rasterizer_control_t control;
} rasterizer_arg_t;


#define RAST_PACKET_WORD_BASE   0x00
#define RAST_PACKET_NUM_WORDS   17
#define RAST_COMMIT_OFFSET      0x44
#define RAST_STATUS_OFFSET      0x48
#define RAST_CONTROL_OFFSET     0x4C
#define RAST_PRESENT_OFFSET     0x50
```

```
#define RAST_STATUS_LEVEL_MASK      0x3F
#define RAST_STATUS_EMPTY_BIT       (1<<6)
#define RAST_STATUS_FULL_BIT        (1<<7)
#define RAST_STATUS_SWAP_BUSY_BIT   (1<<8)


#define RAST_CTRL_IRQ_EN_BIT    (1<<0)
#define RAST_CTRL_WMARK_SHIFT   1
#define RAST_CTRL_WMARK_MASK    0x7F


#define RASTERIZER_MAGIC        'R'


#define RASTERIZER_SUBMIT       _IOW(RASTERIZER_MAGIC, 1, rasterizer_arg_t *)
#define RASTERIZER_STATUS       _IOR(RASTERIZER_MAGIC, 2, rasterizer_arg_t *)
#define RASTERIZER_SET_CONTROL  _IOW(RASTERIZER_MAGIC, 3, rasterizer_arg_t *)
#define RASTERIZER_GET_CONTROL  _IOR(RASTERIZER_MAGIC, 4, rasterizer_arg_t *)
#define RASTERIZER_PRESENT _IO(RASTERIZER_MAGIC, 5)
#endif


================================================================
FILE: software/kernel/avalon_kernel.c
================================================================

// avalon_kernel.c

#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/io.h>
```

```
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include <linux/mm.h>
#include "avalon_kernel.h"

#define DRIVER_NAME "rasterizer"


struct rasterizer_dev {
   struct resource         res;
   void __iomem            *virtbase;
   rasterizer_control_t    control;
} dev;

static void writeTrianglePacket(const triangle_packet_t *pkt)
{

   const __u32 *words = (const __u32 *)pkt;
   int i;


   for (i = 0; i < RAST_PACKET_NUM_WORDS; i++) {
      iowrite32(words[i],
            dev.virtbase + RAST_PACKET_WORD_BASE + (i * 4));
   }

   iowrite32(1, dev.virtbase + RAST_COMMIT_OFFSET);
}

static void present_frame(void)
{


   iowrite32(1, dev.virtbase + RAST_PRESENT_OFFSET);
}
```

```
static void read_status(rasterizer_status_t *status)
{
   __u32 raw = ioread32(dev.virtbase + RAST_STATUS_OFFSET);

   status->fifo_level = raw & RAST_STATUS_LEVEL_MASK;
   status->fifo_empty = (raw & RAST_STATUS_EMPTY_BIT) ? 1 : 0;
   status->fifo_full  = (raw & RAST_STATUS_FULL_BIT) ? 1 : 0;
   status->swap_busy  = (raw & RAST_STATUS_SWAP_BUSY_BIT) ? 1 : 0;
}


static void write_control(const rasterizer_control_t *ctrl)
{
   __u32 raw = 0;

   raw |= (ctrl->irq_enable   ? RAST_CTRL_IRQ_EN_BIT : 0);
   raw |= ((ctrl->low_watermark & RAST_CTRL_WMARK_MASK) << RAST_CTRL_WMARK_SHIFT);

   iowrite32(raw, dev.virtbase + RAST_CONTROL_OFFSET);

   dev.control = *ctrl;
}


static long rasterizer_ioctl(struct file *f, unsigned int cmd,
                             unsigned long arg)
{
   rasterizer_arg_t ra;
   rasterizer_status_t current_status;

   switch (cmd) {

   case RASTERIZER_SUBMIT:
      if (copy_from_user(&ra, (rasterizer_arg_t __user *)arg,
                         sizeof(rasterizer_arg_t)))
         return -EACCES;


         read_status(&current_status);
```

```
        if (current_status.fifo_full) {
            return -EAGAIN;
        }

    writeTrianglePacket(&ra.packet);
    break;

case RASTERIZER_STATUS:
    read_status(&ra.status);

    if (copy_to_user((rasterizer_arg_t __user *)arg, &ra,
                sizeof(rasterizer_arg_t)))
        return -EACCES;
    break;

case RASTERIZER_SET_CONTROL:
    if (copy_from_user(&ra, (rasterizer_arg_t __user *)arg,
                sizeof(rasterizer_arg_t)))
        return -EACCES;

    write_control(&ra.control);
    break;

case RASTERIZER_GET_CONTROL:
    ra.control = dev.control;

    if (copy_to_user((rasterizer_arg_t __user *)arg, &ra,
                sizeof(rasterizer_arg_t)))
        return -EACCES;
    break;

case RASTERIZER_PRESENT:
present_frame();
break;

default:
    return -EINVAL;
}
```

```
    return 0;
}


static int rasterizer_mmap(struct file *f, struct vm_area_struct *vma)
{
    unsigned long size  = vma->vm_end - vma->vm_start;
    unsigned long phys  = (unsigned long)dev.res.start;
    unsigned long pfn   = phys >> PAGE_SHIFT;
    unsigned long limit = PAGE_ALIGN(resource_size(&dev.res));

    if (vma->vm_pgoff != 0)           return -EINVAL;
    if (size > limit)                 return -EINVAL;

    vma->vm_page_prot = pgprot_writecombine(vma->vm_page_prot);

    if (remap_pfn_range(vma, vma->vm_start, pfn, size, vma->vm_page_prot))
        return -EAGAIN;

    return 0;
}

static const struct file_operations rasterizer_fops = {
    .owner          = THIS_MODULE,
    .unlocked_ioctl = rasterizer_ioctl,
    .mmap           = rasterizer_mmap,
};

static struct miscdevice rasterizer_misc_device = {
    .minor = MISC_DYNAMIC_MINOR,
    .name  = DRIVER_NAME,
    .fops  = &rasterizer_fops,
};

static int __init rasterizer_probe(struct platform_device *pdev)
{
    int ret;
```

```
dev.control.irq_enable    = 0;
dev.control.low_watermark = 0;


ret = misc_register(&rasterizer_misc_device);
if (ret) {
   pr_err(DRIVER_NAME ": misc_register failed (%d)\n", ret);
   return ret;
}


ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
if (ret) {
   pr_err(DRIVER_NAME ": of_address_to_resource failed (%d)\n", ret);
   ret = -ENOENT;
   goto out_deregister;
}


if (request_mem_region(dev.res.start, resource_size(&dev.res),
                       DRIVER_NAME) == NULL) {
   pr_err(DRIVER_NAME ": request_mem_region failed\n");
   ret = -EBUSY;
   goto out_deregister;
}


dev.virtbase = of_iomap(pdev->dev.of_node, 0);
if (dev.virtbase == NULL) {
   pr_err(DRIVER_NAME ": of_iomap failed\n");
   ret = -ENOMEM;
   goto out_release_mem_region;
}


write_control(&dev.control);

pr_info(DRIVER_NAME ": probe OK, virtbase=%p phys=0x%08llx size=%llu\n",
        dev.virtbase,
```

```
        (unsigned long long)dev.res.start,
        (unsigned long long)resource_size(&dev.res));

    return 0;

out_release_mem_region:
    release_mem_region(dev.res.start, resource_size(&dev.res));
out_deregister:
    misc_deregister(&rasterizer_misc_device);
    return ret;
}


static int rasterizer_remove(struct platform_device *pdev)
{
    rasterizer_control_t off = { .irq_enable = 0, .low_watermark = 0 };
    write_control(&off);

    iounmap(dev.virtbase);
    release_mem_region(dev.res.start, resource_size(&dev.res));
    misc_deregister(&rasterizer_misc_device);
    return 0;
}

#ifdef CONFIG_OF
static const struct of_device_id rasterizer_of_match[] = {
    { .compatible = "csee4840,rasterizer-1.0" },
    {},
};
MODULE_DEVICE_TABLE(of, rasterizer_of_match);
#endif


static struct platform_driver rasterizer_driver = {
    .driver = {
        .name           = DRIVER_NAME,
        .owner          = THIS_MODULE,
        .of_match_table = of_match_ptr(rasterizer_of_match),
    },
```

```
    .remove = __exit_p(rasterizer_remove),
};


static int __init rasterizer_init(void)
{
    pr_info(DRIVER_NAME ": init\n");
    return platform_driver_probe(&rasterizer_driver, rasterizer_probe);
}

static void __exit rasterizer_exit(void)
{
    platform_driver_unregister(&rasterizer_driver);
    pr_info(DRIVER_NAME ": exit\n");
}

module_init(rasterizer_init);
module_exit(rasterizer_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("CSEE4840 Rasterizer Team, Columbia University");
MODULE_DESCRIPTION("Avalon-MM driver for hardware triangle rasterizer");


================================================================
FILE: software/kernel/rasterizer_driver.c
================================================================

// rasterizer_driver.c

#include "rasterizer.h"


================================================================
FILE: software/userspace/geometry.h
================================================================

// geometry.h
```

```
#ifndef GEOMETRY_H
#define GEOMETRY_H

#include <stdint.h>
#include "../kernel/avalon_kernel.h"


typedef struct { float x, y, z; } vec3_t;
typedef struct { float x, y, z, w; } vec4_t;
typedef struct { float m[4][4]; } mat4_t;


typedef struct { float sx, sy, sz; } screen_vertex_t;


mat4_t mat4_identity(void);
mat4_t mat4_mul(const mat4_t *a, const mat4_t *b);
vec4_t mat4_mul_vec4(const mat4_t *m, vec4_t v);

mat4_t rotation_x(float deg);
mat4_t rotation_y(float deg);
mat4_t rotation_z(float deg);
mat4_t translation(float tx, float ty, float tz);
mat4_t perspective(float fov_deg, float aspect, float near, float far);


vec3_t vec3_sub(vec3_t a, vec3_t b);
vec3_t vec3_cross(vec3_t a, vec3_t b);
vec3_t vec3_normalize(vec3_t v);
float  vec3_dot(vec3_t a, vec3_t b);


uint8_t shade_face(vec3_t normal, vec3_t light_dir,
                   float base_r, float base_g, float base_b);


int setup_triangle(const screen_vertex_t *v0,
                   const screen_vertex_t *v1,
                   const screen_vertex_t *v2,
```

```
                uint8_t color,
                triangle_packet_t *out);

#endif
```

```
================================================================
FILE: software/userspace/geometry.c
================================================================
```

```
// geometry.c

#include "geometry.h"

#include <math.h>
#include <stddef.h>
#include <string.h>


#define SCREEN_W   256
#define SCREEN_H   240
#define FRAC_BITS  12
#define FIXED_ONE  (1 << FRAC_BITS)
#define DEPTH_MAX  65535


vec3_t vec3_sub(vec3_t a, vec3_t b)
{
   return (vec3_t){ a.x - b.x, a.y - b.y, a.z - b.z };
}

vec3_t vec3_cross(vec3_t a, vec3_t b)
{
   return (vec3_t){
      a.y * b.z - a.z * b.y,
      a.z * b.x - a.x * b.z,
      a.x * b.y - a.y * b.x
   };
}
```

```
float vec3_dot(vec3_t a, vec3_t b)
{
   return a.x * b.x + a.y * b.y + a.z * b.z;
}

vec3_t vec3_normalize(vec3_t v)
{
   float l = sqrtf(v.x*v.x + v.y*v.y + v.z*v.z);
   if (l < 1e-8f) return (vec3_t){0, 0, 0};
   return (vec3_t){ v.x/l, v.y/l, v.z/l };
}



mat4_t mat4_identity(void)
{
   mat4_t r;
   memset(&r, 0, sizeof(r));
   r.m[0][0] = r.m[1][1] = r.m[2][2] = r.m[3][3] = 1.0f;
   return r;
}

mat4_t mat4_mul(const mat4_t *a, const mat4_t *b)
{
   mat4_t r;
   memset(&r, 0, sizeof(r));
   for (int i = 0; i < 4; i++)
      for (int j = 0; j < 4; j++)
         for (int k = 0; k < 4; k++)
            r.m[i][j] += a->m[i][k] * b->m[k][j];
   return r;
}

vec4_t mat4_mul_vec4(const mat4_t *m, vec4_t v)
{
   return (vec4_t){
      m->m[0][0]*v.x + m->m[0][1]*v.y + m->m[0][2]*v.z + m->m[0][3]*v.w,
      m->m[1][0]*v.x + m->m[1][1]*v.y + m->m[1][2]*v.z + m->m[1][3]*v.w,
      m->m[2][0]*v.x + m->m[2][1]*v.y + m->m[2][2]*v.z + m->m[2][3]*v.w,
```

```
        m->m[3][0]*v.x + m->m[3][1]*v.y + m->m[3][2]*v.z + m->m[3][3]*v.w
    };
}


mat4_t rotation_x(float deg)
{
    float r = deg * (float)M_PI / 180.0f;
    mat4_t m = mat4_identity();
    m.m[1][1] =  cosf(r); m.m[1][2] = -sinf(r);
    m.m[2][1] =  sinf(r); m.m[2][2] =  cosf(r);
    return m;
}

mat4_t rotation_y(float deg)
{
    float r = deg * (float)M_PI / 180.0f;
    mat4_t m = mat4_identity();
    m.m[0][0] =  cosf(r); m.m[0][2] = sinf(r);
    m.m[2][0] = -sinf(r); m.m[2][2] = cosf(r);
    return m;
}

mat4_t rotation_z(float deg)
{
    float r = deg * (float)M_PI / 180.0f;
    mat4_t m = mat4_identity();
    m.m[0][0] =  cosf(r); m.m[0][1] = -sinf(r);
    m.m[1][0] =  sinf(r); m.m[1][1] =  cosf(r);
    return m;
}

mat4_t translation(float tx, float ty, float tz)
{
    mat4_t m = mat4_identity();
    m.m[0][3] = tx; m.m[1][3] = ty; m.m[2][3] = tz;
    return m;
}
```

```
mat4_t perspective(float fov_deg, float aspect, float near, float far)
{
   float f = 1.0f / tanf(fov_deg * (float)M_PI / 360.0f);
   mat4_t m;
   memset(&m, 0, sizeof(m));
   m.m[0][0] = f / aspect;
   m.m[1][1] = f;
   m.m[2][2] = (far + near) / (near - far);
   m.m[2][3] = (2.0f * far * near) / (near - far);
   m.m[3][2] = -1.0f;
   return m;
}


static uint8_t float_to_rgb332(float r, float g, float b)
{
   int ri = (int)(r * 7.0f + 0.5f);
   int gi = (int)(g * 7.0f + 0.5f);
   int bi = (int)(b * 3.0f + 0.5f);
   if (ri < 0) ri = 0; if (ri > 7) ri = 7;
   if (gi < 0) gi = 0; if (gi > 7) gi = 7;
   if (bi < 0) bi = 0; if (bi > 3) bi = 3;
   return (uint8_t)((ri << 5) | (gi << 2) | bi);
}

uint8_t shade_face(vec3_t normal, vec3_t light_dir,
                   float base_r, float base_g, float base_b)
{
   float ndotl    = vec3_dot(normal, light_dir);
   if (ndotl < 0.0f) ndotl = 0.0f;
   float intensity = 0.25f + 0.75f * ndotl;
   intensity *= 1.2f;
   if (intensity > 1.0f) intensity = 1.0f;
   return float_to_rgb332(base_r * intensity,
                          base_g * intensity,
                          base_b * intensity);
}
```

```
static int32_t to_fixed(float v)
{
   return (int32_t)(v * FIXED_ONE + (v >= 0.0f ? 0.5f : -0.5f));
}


int setup_triangle(const screen_vertex_t *v0, const screen_vertex_t *v1,
                   const screen_vertex_t *v2, uint8_t color,
                   triangle_packet_t *out)
{
   float a0f = v1->sy - v2->sy, b0f = v2->sx - v1->sx;
   float c0f = v1->sx * v2->sy - v2->sx * v1->sy;

   float a1f = v2->sy - v0->sy, b1f = v0->sx - v2->sx;
   float c1f = v2->sx * v0->sy - v0->sx * v2->sy;

   float a2f = v0->sy - v1->sy, b2f = v1->sx - v0->sx;
   float c2f = v0->sx * v1->sy - v1->sx * v0->sy;

   float area = a0f * v0->sx + b0f * v0->sy + c0f;


   if (fabsf(area) <= 1e-6f) return -1;


   if (area < 0.0f) {
      area = -area;

      a0f = -a0f; b0f = -b0f; c0f = -c0f;
      a1f = -a1f; b1f = -b1f; c1f = -c1f;
      a2f = -a2f; b2f = -b2f; c2f = -c2f;
   }

   float xminf = fminf(fminf(v0->sx, v1->sx), v2->sx);
   float yminf = fminf(fminf(v0->sy, v1->sy), v2->sy);
   float xmaxf = fmaxf(fmaxf(v0->sx, v1->sx), v2->sx);
   float ymaxf = fmaxf(fmaxf(v0->sy, v1->sy), v2->sy);

   int bbox_xmin = (int)floorf(xminf);
```

```
int bbox_ymin = (int)floorf(yminf);
int bbox_xmax = (int)ceilf(xmaxf);
int bbox_ymax = (int)ceilf(ymaxf);

if (bbox_xmin < 0) bbox_xmin = 0;
if (bbox_ymin < 0) bbox_ymin = 0;
if (bbox_xmax > SCREEN_W - 1) bbox_xmax = SCREEN_W - 1;
if (bbox_ymax > SCREEN_H - 1) bbox_ymax = SCREEN_H - 1;

if (bbox_xmin > bbox_xmax || bbox_ymin > bbox_ymax) return -1;

bbox_xmin = bbox_xmin & ~15;

float px = (float)bbox_xmin + 0.5f;
float py = (float)bbox_ymin + 0.5f;

float e0_initf = a0f * px + b0f * py + c0f;
float e1_initf = a1f * px + b1f * py + c1f;
float e2_initf = a2f * px + b2f * py + c2f;

float zs0 = v0->sz * (float)DEPTH_MAX;
float zs1 = v1->sz * (float)DEPTH_MAX;
float zs2 = v2->sz * (float)DEPTH_MAX;

float z_step_xf = (a0f * zs0 + a1f * zs1 + a2f * zs2) / area;
float z_step_yf = (b0f * zs0 + b1f * zs1 + b2f * zs2) / area;

float z_at_originf = (e0_initf * zs0 +
                      e1_initf * zs1 +
                      e2_initf * zs2) / area;

memset(out, 0, sizeof(*out));

out->a0 = to_fixed(a0f);
out->b0 = to_fixed(b0f);
out->c0 = to_fixed(e0_initf);

out->a1 = to_fixed(a1f);
out->b1 = to_fixed(b1f);
```

```
    out->c1 = to_fixed(e1_initf);

    out->a2 = to_fixed(a2f);
    out->b2 = to_fixed(b2f);
    out->c2 = to_fixed(e2_initf);

    out->e0_init = to_fixed(e0_initf);
    out->e1_init = to_fixed(e1_initf);
    out->e2_init = to_fixed(e2_initf);

    out->z_origin = to_fixed(z_at_originf);
    out->z_step_x = to_fixed(z_step_xf);
    out->z_step_y = to_fixed(z_step_yf);

    out->bbox_packed =
        ((__u32)(bbox_ymax & 0xFF) << 24) |
        ((__u32)(bbox_ymin & 0xFF) << 16) |
        ((__u32)(bbox_xmax & 0xFF) << 8)  |
        ((__u32)(bbox_xmin & 0xFF));

    out->flags_color = ((__u32)1 << 8) | ((__u32)color & 0xFF);

    return 0;
}
```

```
================================================================
FILE: software/userspace/model.h
================================================================

// model.h

#ifndef MODEL_H
#define MODEL_H

#include "geometry.h"


#define MODEL_MAX_VERTS  131072
```

```
#define MODEL_MAX_FACES  262144

typedef struct {
   int v[3];
} face_t;

typedef struct {
   vec3_t  verts[MODEL_MAX_VERTS];
   face_t  faces[MODEL_MAX_FACES];
   int     num_verts;
   int     num_faces;
   char    name[64];
} model_t;


void model_make_cube(model_t *m);


void model_make_icosphere(model_t *m, int subdivisions);


void model_make_torus(model_t *m, int major_seg, int minor_seg,
                      float R, float r);


void model_make_teapot(model_t *m);


void model_make_saturn(model_t *m);


void model_make_lego(model_t *m);


void model_make_dna(model_t *m);


void model_make_teapot_552(model_t *m);
```

```
void model_make_rocket(model_t *m);

void model_make_minifigure(model_t *m);

int model_load_obj(model_t *m, const char *filename);

#endif
```

```
================================================================
FILE: software/userspace/model.c
================================================================
```

```
// model.c

#include "model.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>


void model_make_cube(model_t *m)
{
    memset(m, 0, sizeof(*m));
    strncpy(m->name, "cube", sizeof(m->name) - 1);

    static const vec3_t verts[8] = {
        {-1,-1,-1}, { 1,-1,-1}, { 1, 1,-1}, {-1, 1,-1},
        {-1,-1, 1}, { 1,-1, 1}, { 1, 1, 1}, {-1, 1, 1}
    };
    memcpy(m->verts, verts, sizeof(verts));
    m->num_verts = 8;

    static const face_t faces[12] = {
```

```
        {{0,1,2}}, {{0,2,3}},
        {{5,4,7}}, {{5,7,6}},
        {{4,0,3}}, {{4,3,7}},
        {{1,5,6}}, {{1,6,2}},
        {{3,2,6}}, {{3,6,7}},
        {{4,5,1}}, {{4,1,0}}
    };
    memcpy(m->faces, faces, sizeof(faces));
    m->num_faces = 12;
}


#define MIDPT_CACHE_MAX 2048

typedef struct { int a, b, mid; } midpt_entry_t;

static int          g_num_midpt;
static midpt_entry_t g_midpt_cache[MIDPT_CACHE_MAX];

static void midpt_cache_reset(void) { g_num_midpt = 0; }

static int midpt_cache_lookup(int a, int b)
{
    int lo = a < b ? a : b;
    int hi = a < b ? b : a;
    for (int i = 0; i < g_num_midpt; i++)
        if (g_midpt_cache[i].a == lo && g_midpt_cache[i].b == hi)
            return g_midpt_cache[i].mid;
    return -1;
}

static void midpt_cache_insert(int a, int b, int mid)
{
    if (g_num_midpt >= MIDPT_CACHE_MAX) return;
    int lo = a < b ? a : b;
    int hi = a < b ? b : a;
    g_midpt_cache[g_num_midpt++] = (midpt_entry_t){ lo, hi, mid };
}
```

```
static int icosphere_midpoint(model_t *m, int a, int b)
{
    int cached = midpt_cache_lookup(a, b);
    if (cached >= 0) return cached;

    if (m->num_verts >= MODEL_MAX_VERTS) return 0;

    vec3_t va  = m->verts[a];
    vec3_t vb  = m->verts[b];
    vec3_t mid = { (va.x+vb.x)*0.5f, (va.y+vb.y)*0.5f, (va.z+vb.z)*0.5f };


    float l = sqrtf(mid.x*mid.x + mid.y*mid.y + mid.z*mid.z);
    if (l > 1e-8f) { mid.x /= l; mid.y /= l; mid.z /= l; }

    int idx = m->num_verts;
    m->verts[m->num_verts++] = mid;
    midpt_cache_insert(a, b, idx);
    return idx;
}

void model_make_icosphere(model_t *m, int subdivisions)
{
    memset(m, 0, sizeof(*m));
    strncpy(m->name, "sphere", sizeof(m->name) - 1);

    float t = (1.0f + sqrtf(5.0f)) / 2.0f;

    vec3_t init_v[12] = {
        {-1, t, 0}, { 1, t, 0}, {-1,-t, 0}, { 1,-t, 0},
        { 0,-1, t}, { 0, 1, t}, { 0,-1,-t}, { 0, 1,-t},
        { t, 0,-1}, { t, 0, 1}, {-t, 0,-1}, {-t, 0, 1}
    };
    for (int i = 0; i < 12; i++) {
        float l = sqrtf(init_v[i].x*init_v[i].x +
                        init_v[i].y*init_v[i].y +
                        init_v[i].z*init_v[i].z);
        init_v[i].x /= l; init_v[i].y /= l; init_v[i].z /= l;
    }
```

```
    memcpy(m->verts, init_v, sizeof(init_v));
    m->num_verts = 12;

    face_t init_f[20] = {
        {{0,11,5}}, {{0,5,1}},  {{0,1,7}},   {{0,7,10}},  {{0,10,11}},
        {{1,5,9}},  {{5,11,4}}, {{11,10,2}},  {{10,7,6}},  {{7,1,8}},
        {{3,9,4}},  {{3,4,2}},  {{3,2,6}},    {{3,6,8}},   {{3,8,9}},
        {{4,9,5}},  {{2,4,11}}, {{6,2,10}},   {{8,6,7}},   {{9,8,1}}
    };
    memcpy(m->faces, init_f, sizeof(init_f));
    m->num_faces = 20;

    static face_t tmp[MODEL_MAX_FACES];

    for (int s = 0; s < subdivisions; s++) {
        if (m->num_faces * 4 > MODEL_MAX_FACES) break;
        midpt_cache_reset();
        int new_count = 0;

        for (int i = 0; i < m->num_faces; i++) {
            int v0 = m->faces[i].v[0];
            int v1 = m->faces[i].v[1];
            int v2 = m->faces[i].v[2];

            int a = icosphere_midpoint(m, v0, v1);
            int b = icosphere_midpoint(m, v1, v2);
            int c = icosphere_midpoint(m, v2, v0);


            tmp[new_count++] = (face_t){{ v0, a, c }};
            tmp[new_count++] = (face_t){{ v1, b, a }};
            tmp[new_count++] = (face_t){{ v2, c, b }};
            tmp[new_count++] = (face_t){{  a, b, c }};
        }

        memcpy(m->faces, tmp, new_count * sizeof(face_t));
        m->num_faces = new_count;
    }
}
```

```
void model_make_torus(model_t *m, int major_seg, int minor_seg,
                      float R, float r)
{
   memset(m, 0, sizeof(*m));
   strncpy(m->name, "torus", sizeof(m->name) - 1);

   for (int i = 0; i < major_seg; i++) {
      float theta = 2.0f * (float)M_PI * i / major_seg;
      for (int j = 0; j < minor_seg; j++) {
         float phi = 2.0f * (float)M_PI * j / minor_seg;
         if (m->num_verts >= MODEL_MAX_VERTS) break;
         m->verts[m->num_verts++] = (vec3_t){
            (R + r * cosf(phi)) * cosf(theta),
            r * sinf(phi),
            (R + r * cosf(phi)) * sinf(theta)
         };
      }
   }

   for (int i = 0; i < major_seg; i++) {
      int ni = (i + 1) % major_seg;
      for (int j = 0; j < minor_seg; j++) {
         int nj = (j + 1) % minor_seg;
         int a = i  * minor_seg + j;
         int b = ni * minor_seg + j;
         int c = ni * minor_seg + nj;
         int d = i  * minor_seg + nj;
         if (m->num_faces + 1 >= MODEL_MAX_FACES) break;
         m->faces[m->num_faces++] = (face_t){{ a, b, c }};
         m->faces[m->num_faces++] = (face_t){{ a, c, d }};
      }
   }
}


int model_load_obj(model_t *m, const char *filename)
{
```

```
FILE *f = fopen(filename, "r");
if (!f) return -1;

memset(m, 0, sizeof(*m));
strncpy(m->name, filename, sizeof(m->name) - 1);

char line[256];
while (fgets(line, sizeof(line), f)) {
    char prefix[8];
    if (sscanf(line, "%7s", prefix) != 1) continue;

    if (strcmp(prefix, "v") == 0) {
        if (m->num_verts >= MODEL_MAX_VERTS) continue;
        vec3_t v;
        if (sscanf(line + 1, "%f %f %f", &v.x, &v.y, &v.z) == 3)
            m->verts[m->num_verts++] = v;

    } else if (strcmp(prefix, "f") == 0) {
        int idx[4] = {0, 0, 0, 0};
        int n = 0;
        const char *p = line + 1;

        while (n < 4) {
            while (*p == ' ' || *p == '\t') p++;
            if (*p == '\0' || *p == '\n' || *p == '\r') break;
            int v_idx = 0;
            if (sscanf(p, "%d", &v_idx) != 1) break;
            idx[n++] = v_idx - 1;
            while (*p && *p != ' ' && *p != '\t' && *p != '\n') p++;
        }


        int valid = 1;
        for (int vi = 0; vi < n; vi++) {
            if (idx[vi] < 0 || idx[vi] >= m->num_verts) {
                valid = 0; break;
            }
        }
        if (!valid) continue;
```

```
            if (n >= 3 && m->num_faces < MODEL_MAX_FACES)
                m->faces[m->num_faces++] = (face_t){{ idx[0], idx[1], idx[2] }};
            if (n == 4 && m->num_faces < MODEL_MAX_FACES)
                m->faces[m->num_faces++] = (face_t){{ idx[0], idx[2], idx[3] }};
        }
    }

    fclose(f);
    return (m->num_verts > 0 && m->num_faces > 0) ? 0 : -1;
}


#define TEAPOT_DIV 3


static const float teapot_cp[306][3] = {
    {1.4f,2.4f,0.0f},{1.4f,2.4f,-0.784f},{0.784f,2.4f,-1.4f},
    {0.0f,2.4f,-1.4f},{1.3375f,2.53125f,0.0f},{1.3375f,2.53125f,-0.749f},
    {0.749f,2.53125f,-1.3375f},{0.0f,2.53125f,-1.3375f},{1.4375f,2.53125f,0.0f},
    {1.4375f,2.53125f,-0.805f},{0.805f,2.53125f,-1.4375f},{0.0f,2.53125f,-1.4375f},
    {1.5f,2.4f,0.0f},{1.5f,2.4f,-0.84f},{0.84f,2.4f,-1.5f},{0.0f,2.4f,-1.5f},
    {-0.784f,2.4f,-1.4f},{-1.4f,2.4f,-0.784f},{-1.4f,2.4f,0.0f},
    {-0.749f,2.53125f,-1.3375f},{-1.3375f,2.53125f,-0.749f},{-1.3375f,2.53125f,0.0f},
    {-0.805f,2.53125f,-1.4375f},{-1.4375f,2.53125f,-0.805f},{-1.4375f,2.53125f,0.0f},
    {-0.84f,2.4f,-1.5f},{-1.5f,2.4f,-0.84f},{-1.5f,2.4f,0.0f},
    {-1.4f,2.4f,0.784f},{-0.784f,2.4f,1.4f},{0.0f,2.4f,1.4f},
    {-1.3375f,2.53125f,0.749f},{-0.749f,2.53125f,1.3375f},{0.0f,2.53125f,1.3375f},
    {-1.4375f,2.53125f,0.805f},{-0.805f,2.53125f,1.4375f},{0.0f,2.53125f,1.4375f},
    {-1.5f,2.4f,0.84f},{-0.84f,2.4f,1.5f},{0.0f,2.4f,1.5f},
    {0.784f,2.4f,1.4f},{1.4f,2.4f,0.784f},{0.749f,2.53125f,1.3375f},
    {1.3375f,2.53125f,0.749f},{0.805f,2.53125f,1.4375f},{1.4375f,2.53125f,0.805f},
    {0.84f,2.4f,1.5f},{1.5f,2.4f,0.84f},{1.5f,2.25f,0.0f},
    {1.5f,2.25f,-0.84f},{0.84f,2.25f,-1.5f},{0.0f,2.25f,-1.5f},
    {1.75f,1.725f,0.0f},{1.75f,1.725f,-0.98f},{0.98f,1.725f,-1.75f},
    {0.0f,1.725f,-1.75f},{2.0f,1.2f,0.0f},{2.0f,1.2f,-1.12f},
    {1.12f,1.2f,-2.0f},{0.0f,1.2f,-2.0f},{-0.84f,2.25f,-1.5f},
    {-1.5f,2.25f,-0.84f},{-1.5f,2.25f,0.0f},{-0.98f,1.725f,-1.75f},
    {-1.75f,1.725f,-0.98f},{-1.75f,1.725f,0.0f},{-1.12f,1.2f,-2.0f},
```

{-2.0f,1.2f,-1.12f},{-2.0f,1.2f,0.0f},{-1.5f,2.25f,0.84f},
{-0.84f,2.25f,1.5f},{0.0f,2.25f,1.5f},{-1.75f,1.725f,0.98f},
{-0.98f,1.725f,1.75f},{0.0f,1.725f,1.75f},{-2.0f,1.2f,1.12f},
{-1.12f,1.2f,2.0f},{0.0f,1.2f,2.0f},{0.84f,2.25f,1.5f},
{1.5f,2.25f,0.84f},{0.98f,1.725f,1.75f},{1.75f,1.725f,0.98f},
{1.12f,1.2f,2.0f},{2.0f,1.2f,1.12f},{2.0f,0.9f,0.0f},
{2.0f,0.9f,-1.12f},{1.12f,0.9f,-2.0f},{0.0f,0.9f,-2.0f},
{2.0f,0.45f,0.0f},{2.0f,0.45f,-1.12f},{1.12f,0.45f,-2.0f},
{0.0f,0.45f,-2.0f},{1.5f,0.225f,0.0f},{1.5f,0.225f,-0.84f},
{0.84f,0.225f,-1.5f},{0.0f,0.225f,-1.5f},{1.5f,0.15f,0.0f},
{1.5f,0.15f,-0.84f},{0.84f,0.15f,-1.5f},{0.0f,0.15f,-1.5f},
{-1.12f,0.9f,-2.0f},{-2.0f,0.9f,-1.12f},{-2.0f,0.9f,0.0f},
{-1.12f,0.45f,-2.0f},{-2.0f,0.45f,-1.12f},{-2.0f,0.45f,0.0f},
{-0.84f,0.225f,-1.5f},{-1.5f,0.225f,-0.84f},{-1.5f,0.225f,0.0f},
{-0.84f,0.15f,-1.5f},{-1.5f,0.15f,-0.84f},{-1.5f,0.15f,0.0f},
{-2.0f,0.9f,1.12f},{-1.12f,0.9f,2.0f},{0.0f,0.9f,2.0f},
{-2.0f,0.45f,1.12f},{-1.12f,0.45f,2.0f},{0.0f,0.45f,2.0f},
{-1.5f,0.225f,0.84f},{-0.84f,0.225f,1.5f},{0.0f,0.225f,1.5f},
{-1.5f,0.15f,0.84f},{-0.84f,0.15f,1.5f},{0.0f,0.15f,1.5f},
{1.12f,0.9f,2.0f},{2.0f,0.9f,1.12f},{1.12f,0.45f,2.0f},
{2.0f,0.45f,1.12f},{0.84f,0.225f,1.5f},{1.5f,0.225f,0.84f},
{0.84f,0.15f,1.5f},{1.5f,0.15f,0.84f},{-1.6f,0.0f,-1.5f},
{-1.5f,0.15f,-1.5f},{-2.5f,0.0f,-1.5f},{-2.5f,0.15f,-1.5f},
{-2.5f,0.0f,-1.0f},{-2.5f,0.15f,-1.0f},{-2.5f,0.0f,0.0f},
{-2.5f,0.15f,0.0f},{-2.5f,0.0f,1.0f},{-2.5f,0.15f,1.0f},
{-2.5f,0.0f,1.5f},{-2.5f,0.15f,1.5f},{-1.5f,0.15f,1.5f},
{-1.6f,0.0f,1.5f},{-1.6f,0.0f,0.0f},{-1.6f,0.15f,1.5f},
{-1.6f,0.0f,-1.5f},{-1.6f,0.15f,-1.5f},{-1.6f,0.0f,0.0f},
{-1.6f,0.15f,0.0f},{-2.5f,0.0f,0.0f},{-2.5f,0.15f,0.0f},
{-2.5f,0.0f,-1.0f},{-2.5f,0.15f,-1.0f},{-1.6f,0.0f,-1.5f},
{-1.6f,0.15f,-1.5f},{-2.5f,0.0f,-1.5f},{-2.5f,0.15f,-1.5f},
{-2.5f,0.0f,1.0f},{-2.5f,0.15f,1.0f},{-1.6f,0.0f,1.5f},
{-1.6f,0.15f,1.5f},{-2.5f,0.0f,1.5f},{-2.5f,0.15f,1.5f},
{-2.5f,0.0f,0.0f},{-2.5f,0.15f,0.0f},{1.7f,1.425f,0.0f},
{1.7f,1.425f,-0.952f},{0.952f,1.425f,-1.7f},{0.0f,1.425f,-1.7f},
{1.7f,0.6f,0.0f},{1.7f,0.6f,-0.952f},{0.952f,0.6f,-1.7f},
{0.0f,0.6f,-1.7f},{-0.952f,1.425f,-1.7f},{-1.7f,1.425f,-0.952f},
{-1.7f,1.425f,0.0f},{-0.952f,0.6f,-1.7f},{-1.7f,0.6f,-0.952f},
{-1.7f,0.6f,0.0f},{-1.7f,1.425f,0.952f},{-0.952f,1.425f,1.7f},

```
    {0.0f,1.425f,1.7f},{-1.7f,0.6f,0.952f},{-0.952f,0.6f,1.7f},
    {0.0f,0.6f,1.7f},{0.952f,1.425f,1.7f},{1.7f,1.425f,0.952f},
    {0.952f,0.6f,1.7f},{1.7f,0.6f,0.952f},{0.0f,0.0f,0.0f},
    {1.425f,0.0f,0.0f},{1.425f,0.0f,-0.798f},{0.798f,0.0f,-1.425f},
    {0.0f,0.0f,-1.425f},{0.0f,0.0f,0.0f},{0.0f,3.15f,0.0f},
    {0.8f,3.15f,0.0f},{0.8f,3.15f,-0.45f},{0.45f,3.15f,-0.8f},
    {0.0f,3.15f,-0.8f},{0.0f,2.85f,0.0f},{0.2f,2.7f,0.0f},
    {0.2f,2.7f,-0.112f},{0.112f,2.7f,-0.2f},{0.0f,2.7f,-0.2f},
    {0.4f,2.55f,0.0f},{0.4f,2.55f,-0.224f},{0.224f,2.55f,-0.4f},
    {0.0f,2.55f,-0.4f},{1.3f,2.55f,0.0f},{1.3f,2.55f,-0.728f},
    {0.728f,2.55f,-1.3f},{0.0f,2.55f,-1.3f},{1.3f,2.4f,0.0f},
    {1.3f,2.4f,-0.728f},{0.728f,2.4f,-1.3f},{0.0f,2.4f,-1.3f},
    {-0.45f,3.15f,-0.8f},{-0.8f,3.15f,-0.45f},{-0.8f,3.15f,0.0f},
    {-0.112f,2.7f,-0.2f},{-0.2f,2.7f,-0.112f},{-0.2f,2.7f,0.0f},
    {-0.224f,2.55f,-0.4f},{-0.4f,2.55f,-0.224f},{-0.4f,2.55f,0.0f},
    {-0.728f,2.55f,-1.3f},{-1.3f,2.55f,-0.728f},{-1.3f,2.55f,0.0f},
    {-0.728f,2.4f,-1.3f},{-1.3f,2.4f,-0.728f},{-1.3f,2.4f,0.0f},
    {-0.8f,3.15f,0.45f},{-0.45f,3.15f,0.8f},{0.0f,3.15f,0.8f},
    {-0.2f,2.7f,0.112f},{-0.112f,2.7f,0.2f},{0.0f,2.7f,0.2f},
    {-0.4f,2.55f,0.224f},{-0.224f,2.55f,0.4f},{0.0f,2.55f,0.4f},
    {-1.3f,2.55f,0.728f},{-0.728f,2.55f,1.3f},{0.0f,2.55f,1.3f},
    {-1.3f,2.4f,0.728f},{-0.728f,2.4f,1.3f},{0.0f,2.4f,1.3f},
    {0.45f,3.15f,0.8f},{0.8f,3.15f,0.45f},{0.112f,2.7f,0.2f},
    {0.2f,2.7f,0.112f},{0.224f,2.55f,0.4f},{0.4f,2.55f,0.224f},
    {0.728f,2.55f,1.3f},{1.3f,2.55f,0.728f},{0.728f,2.4f,1.3f},
    {1.3f,2.4f,0.728f},{0.0f,3.15f,0.0f},{0.8f,3.15f,0.0f},
    {0.0f,0.0f,0.0f},{1.425f,0.0f,0.798f},{0.798f,0.0f,1.425f},
    {0.0f,0.0f,1.425f},{-0.798f,0.0f,1.425f},{-1.425f,0.0f,0.798f},
    {-1.425f,0.0f,0.0f},{-0.798f,0.0f,-1.425f},{0.0f,0.0f,-1.425f}
};


static const int teapot_patches[32][16] = {
    {  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16},
    {  4, 17, 18, 19,  8, 20, 21, 22, 12, 23, 24, 25, 16, 26, 27, 28},
    { 19, 29, 30, 31, 22, 32, 33, 34, 25, 35, 36, 37, 28, 38, 39, 40},
    { 31, 41, 42,  1, 34, 43, 44,  5, 37, 45, 46,  9, 40, 47, 48, 13},
    { 13, 14, 15, 16, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60},
    { 16, 26, 27, 28, 52, 61, 62, 63, 56, 64, 65, 66, 60, 67, 68, 69},
```

```
    { 28, 38, 39, 40, 63, 70, 71, 72, 66, 73, 74, 75, 69, 76, 77, 78},
    { 40, 47, 48, 13, 72, 79, 80, 49, 75, 81, 82, 53, 78, 83, 84, 57},
    { 57, 58, 59, 60, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96},
    { 60, 67, 68, 69, 88, 97, 98, 99, 92,100,101,102, 96,103,104,105},
    { 69, 76, 77, 78, 99,106,107,108,102,109,110,111,105,112,113,114},
    { 78, 83, 84, 57,108,115,116, 85,111,117,118, 89,114,119,120, 93},
    {121,122,123,124,125,126,127,128,129,130,131,132,133,134,135,136},
    {124,137,138,121,128,139,140,125,132,141,142,129,136,143,144,133},
    {133,134,135,136,145,146,147,148,149,150,151,152, 69,153,154,155},
    {136,143,144,133,148,156,157,145,152,158,159,149,155,160,161, 69},
    {162,163,164,165,166,167,168,169,170,171,172,173,174,175,176,177},
    {165,178,179,162,169,180,181,166,173,182,183,170,177,184,185,174},
    {174,175,176,177,186,187,188,189,190,191,192,193,194,195,196,197},
    {177,184,185,174,189,198,199,186,193,200,201,190,197,202,203,194},
    {204,204,204,204,207,208,209,210,211,211,211,211,212,213,214,215},
    {204,204,204,204,210,217,218,219,211,211,211,211,215,220,221,222},
    {204,204,204,204,219,224,225,226,211,211,211,211,222,227,228,229},
    {204,204,204,204,226,230,231,207,211,211,211,211,229,232,233,212},
    {212,213,214,215,234,235,236,237,238,239,240,241,242,243,244,245},
    {215,220,221,222,237,246,247,248,241,249,250,251,245,252,253,254},
    {222,227,228,229,248,255,256,257,251,258,259,260,254,261,262,263},
    {229,232,233,212,257,264,265,234,260,266,267,238,263,268,269,242},
    {270,270,270,270,279,280,281,282,275,276,277,278,271,272,273,274},
    {270,270,270,270,282,289,290,291,278,286,287,288,274,283,284,285},
    {270,270,270,270,291,298,299,300,288,295,296,297,285,292,293,294},
    {270,270,270,270,300,305,306,279,297,303,304,275,294,301,302,271},
};


static float bezier1(float p0, float p1, float p2, float p3, float t)
{
    float mt = 1.0f - t;
    return mt*mt*mt*p0 + 3.0f*mt*mt*t*p1 + 3.0f*mt*t*t*p2 + t*t*t*p3;
}


static vec3_t bezier_patch(const float cp[16][3], float u, float v)
{
    float tmp[4][3];
```

```
    for (int i = 0; i < 4; i++) {
        for (int c = 0; c < 3; c++)
            tmp[i][c] = bezier1(cp[i*4+0][c], cp[i*4+1][c],
                                cp[i*4+2][c], cp[i*4+3][c], v);
    }
    return (vec3_t){
        bezier1(tmp[0][0], tmp[1][0], tmp[2][0], tmp[3][0], u),
        bezier1(tmp[0][1], tmp[1][1], tmp[2][1], tmp[3][1], u),
        bezier1(tmp[0][2], tmp[1][2], tmp[2][2], tmp[3][2], u)
    };
}

void model_make_teapot(model_t *m)
{
    memset(m, 0, sizeof(*m));
    strncpy(m->name, "teapot", sizeof(m->name) - 1);

    int div = TEAPOT_DIV;

    for (int p = 0; p < 32; p++) {

        float cp[16][3];
        for (int k = 0; k < 16; k++) {
            int idx = teapot_patches[p][k] - 1;
            cp[k][0] = teapot_cp[idx][0];
            cp[k][1] = teapot_cp[idx][1];
            cp[k][2] = teapot_cp[idx][2];
        }


        for (int i = 0; i < div; i++) {
            float u0 = (float)i       / div;
            float u1 = (float)(i + 1) / div;
            for (int j = 0; j < div; j++) {
                float v0 = (float)j       / div;
                float v1 = (float)(j + 1) / div;

                if (m->num_verts + 4 > MODEL_MAX_VERTS) goto done;
                if (m->num_faces + 2 > MODEL_MAX_FACES) goto done;
```

```
            vec3_t q00 = bezier_patch(cp, u0, v0);
            vec3_t q10 = bezier_patch(cp, u1, v0);
            vec3_t q01 = bezier_patch(cp, u0, v1);
            vec3_t q11 = bezier_patch(cp, u1, v1);


            float scale = 0.4f;
            float oy    = -1.2f;
            q00.x *= scale; q00.y = q00.y * scale + oy; q00.z *= scale;
            q10.x *= scale; q10.y = q10.y * scale + oy; q10.z *= scale;
            q01.x *= scale; q01.y = q01.y * scale + oy; q01.z *= scale;
            q11.x *= scale; q11.y = q11.y * scale + oy; q11.z *= scale;


            float ex = q10.x - q00.x, ey = q10.y - q00.y, ez = q10.z - q00.z;
            float fx = q01.x - q00.x, fy = q01.y - q00.y, fz = q01.z - q00.z;
            float nx = ey*fz - ez*fy;
            float ny = ez*fx - ex*fz;
            float nz = ex*fy - ey*fx;

            int flip = (nx*q00.x + ny*q00.y + nz*q00.z) < 0.0f;

            int base = m->num_verts;
            m->verts[m->num_verts++] = q00;
            m->verts[m->num_verts++] = q10;
            m->verts[m->num_verts++] = q11;
            m->verts[m->num_verts++] = q01;

            if (!flip) {
               m->faces[m->num_faces++] = (face_t){{ base+0, base+1, base+2 }};
               m->faces[m->num_faces++] = (face_t){{ base+0, base+2, base+3 }};
            } else {
               m->faces[m->num_faces++] = (face_t){{ base+0, base+3, base+2 }};
               m->faces[m->num_faces++] = (face_t){{ base+0, base+2, base+1 }};
            }
         }
      }
   }
```

```
done:;
}


static const vec3_t teapot552_verts[] = {
    {-1.5f, 2.25f, 0.0f},
    {-1.525926f, 2.191666f, -0.2f},
    {-1.574074f, 2.083333f, -0.2f},
    {-1.6f, 2.025f, 0.0f},
    {-2.333333f, 2.233333f, 0.0f},
    {-2.297805f, 2.17716f, -0.2f},
    {-2.231824f, 2.072839f, -0.2f},
    {-2.196296f, 2.016667f, 0.0f},
    {-2.833333f, 2.116667f, 0.0f},
    {-2.765157f, 2.075617f, -0.2f},
    {-2.638546f, 1.999383f, -0.2f},
    {-2.57037f, 1.958333f, 0.0f},
    {-3.0f, 1.8f, 0.0f},
    {-2.922222f, 1.8f, -0.2f},
    {-2.777778f, 1.8f, -0.2f},
    {-2.7f, 1.8f, 0.0f},
    {-1.525926f, 2.191666f, 0.2f},
    {-1.574074f, 2.083333f, 0.2f},
    {-2.297805f, 2.17716f, 0.2f},
    {-2.231824f, 2.072839f, 0.2f},
    {-2.765157f, 2.075617f, 0.2f},
    {-2.638546f, 1.999383f, 0.2f},
    {-2.922222f, 1.8f, 0.2f},
    {-2.777778f, 1.8f, 0.2f},
    {-2.881481f, 1.363889f, 0.0f},
    {-2.816186f, 1.403498f, -0.2f},
    {-2.694924f, 1.477057f, -0.2f},
    {-2.629629f, 1.516667f, 0.0f},
    {-2.518518f, 0.9611111f, 0.0f},
    {-2.488751f, 1.018724f, -0.2f},
    {-2.43347f, 1.12572f, -0.2f},
    {-2.403704f, 1.183333f, 0.0f},
    {-1.9f, 0.6f, 0.0f},
    {-1.925926f, 0.6777777f, -0.2f},
```

{-1.974074f, 0.8222224f, -0.2f},
{-2.0f, 0.9f, 0.0f},
{-2.816186f, 1.403498f, 0.2f},
{-2.694924f, 1.477057f, 0.2f},
{-2.488751f, 1.018724f, 0.2f},
{-2.43347f, 1.12572f, 0.2f},
{-1.925926f, 0.6777777f, 0.2f},
{-1.974074f, 0.8222224f, 0.2f},
{2.8f, 2.4f, 0.0f},
{2.903703f, 2.4f, -0.1f},
{3.096296f, 2.4f, -0.1f},
{3.2f, 2.4f, 0.0f},
{2.84074f, 2.45f, 0.0f},
{2.982373f, 2.455401f, -0.1172839f},
{3.245404f, 2.465432f, -0.1172839f},
{3.387037f, 2.470833f, 0.0f},
{2.792593f, 2.45f, 0.0f},
{2.95775f, 2.454321f, -0.1493827f},
{3.264472f, 2.462346f, -0.1493827f},
{3.42963f, 2.466667f, 0.0f},
{2.7f, 2.4f, 0.0f},
{2.855555f, 2.4f, -0.1666667f},
{3.144444f, 2.4f, -0.1666667f},
{3.3f, 2.4f, 0.0f},
{2.903703f, 2.4f, 0.1f},
{3.096296f, 2.4f, 0.1f},
{2.982373f, 2.455401f, 0.1172839f},
{3.245404f, 2.465432f, 0.1172839f},
{2.95775f, 2.454321f, 0.1493827f},
{3.264472f, 2.462346f, 0.1493827f},
{2.855555f, 2.4f, 0.1666667f},
{3.144444f, 2.4f, 0.1666667f},
{2.462963f, 2.013889f, 0.0f},
{2.549382f, 1.962757f, -0.2375308f},
{2.709877f, 1.867798f, -0.2375308f},
{2.796296f, 1.816666f, 0.0f},
{2.27037f, 1.611111f, 0.0f},
{2.339506f, 1.47428f, -0.3691358f},
{2.467901f, 1.220165f, -0.3691358f},

{2.537037f, 1.083333f, 0.0f},
{1.7f, 1.425f, 0.0f},
{1.7f, 1.211111f, -0.44f},
{1.7f, 0.8138888f, -0.44f},
{1.7f, 0.6f, 0.0f},
{2.549382f, 1.962757f, 0.2375308f},
{2.709877f, 1.867798f, 0.2375308f},
{2.339506f, 1.47428f, 0.3691358f},
{2.467901f, 1.220165f, 0.3691358f},
{1.7f, 1.211111f, 0.44f},
{1.7f, 0.8138888f, 0.44f},
{0.0f, 3.15f, 0.0f},
{-0.3629629f, 3.066666f, 0.0f},
{-0.3142276f, 3.066666f, -0.184834f},
{-0.184834f, 3.066666f, -0.3142277f},
{0.0f, 3.066666f, -0.3629629f},
{-0.237037f, 2.883333f, 0.0f},
{-0.2051797f, 2.883333f, -0.1206475f},
{-0.1206474f, 2.883333f, -0.2051797f},
{0.0f, 2.883333f, -0.237037f},
{-0.2f, 2.7f, 0.0f},
{-0.173037f, 2.7f, -0.1016296f},
{-0.1016296f, 2.7f, -0.173037f},
{0.0f, 2.7f, -0.2f},
{0.184834f, 3.066666f, -0.3142276f},
{0.3142277f, 3.066666f, -0.184834f},
{0.3629629f, 3.066666f, 0.0f},
{0.1206475f, 2.883333f, -0.2051797f},
{0.2051797f, 2.883333f, -0.1206474f},
{0.237037f, 2.883333f, 0.0f},
{0.1016296f, 2.7f, -0.173037f},
{0.173037f, 2.7f, -0.1016296f},
{0.2f, 2.7f, 0.0f},
{0.0f, 3.066666f, 0.3629629f},
{0.184834f, 3.066666f, 0.3142276f},
{0.3142277f, 3.066666f, 0.184834f},
{0.0f, 2.883333f, 0.237037f},
{0.1206475f, 2.883333f, 0.2051797f},
{0.2051797f, 2.883333f, 0.1206474f},

{0.0f, 2.7f, 0.2f},
{0.1016296f, 2.7f, 0.173037f},
{0.173037f, 2.7f, 0.1016296f},
{-0.3142276f, 3.066666f, 0.184834f},
{-0.184834f, 3.066666f, 0.3142277f},
{-0.2051797f, 2.883333f, 0.1206475f},
{-0.1206474f, 2.883333f, 0.2051797f},
{-0.173037f, 2.7f, 0.1016296f},
{-0.1016296f, 2.7f, 0.173037f},
{-0.574074f, 2.588889f, 0.0f},
{-0.4966803f, 2.588889f, -0.2917147f},
{-0.2917146f, 2.588889f, -0.4966803f},
{0.0f, 2.588889f, -0.574074f},
{-1.059259f, 2.511111f, 0.0f},
{-0.9164553f, 2.511111f, -0.5382606f},
{-0.5382606f, 2.511111f, -0.9164554f},
{0.0f, 2.511111f, -1.059259f},
{-1.3f, 2.4f, 0.0f},
{-1.12474f, 2.4f, -0.6605926f},
{-0.6605925f, 2.4f, -1.124741f},
{0.0f, 2.4f, -1.3f},
{0.2917147f, 2.588889f, -0.4966803f},
{0.4966803f, 2.588889f, -0.2917146f},
{0.574074f, 2.588889f, 0.0f},
{0.5382606f, 2.511111f, -0.9164553f},
{0.9164554f, 2.511111f, -0.5382606f},
{1.059259f, 2.511111f, 0.0f},
{0.6605926f, 2.4f, -1.12474f},
{1.124741f, 2.4f, -0.6605925f},
{1.3f, 2.4f, 0.0f},
{0.0f, 2.588889f, 0.574074f},
{0.2917147f, 2.588889f, 0.4966803f},
{0.4966803f, 2.588889f, 0.2917146f},
{0.0f, 2.511111f, 1.059259f},
{0.5382606f, 2.511111f, 0.9164553f},
{0.9164554f, 2.511111f, 0.5382606f},
{0.0f, 2.4f, 1.3f},
{0.6605926f, 2.4f, 1.12474f},
{1.124741f, 2.4f, 0.6605925f},

{-0.4966803f, 2.588889f, 0.2917147f},
{-0.2917146f, 2.588889f, 0.4966803f},
{-0.9164553f, 2.511111f, 0.5382606f},
{-0.5382606f, 2.511111f, 0.9164554f},
{-1.12474f, 2.4f, 0.6605926f},
{-0.6605925f, 2.4f, 1.124741f},
{-1.4f, 2.4f, 0.0f},
{-1.211259f, 2.4f, -0.7114074f},
{-0.7114074f, 2.4f, -1.211259f},
{0.0f, 2.4f, -1.4f},
{-1.384259f, 2.4875f, 0.0f},
{-1.19764f, 2.487499f, -0.7034087f},
{-0.7034087f, 2.4875f, -1.19764f},
{0.0f, 2.4875f, -1.384259f},
{-1.432407f, 2.4875f, 0.0f},
{-1.239297f, 2.4875f, -0.7278751f},
{-0.7278751f, 2.4875f, -1.239298f},
{0.0f, 2.4875f, -1.432407f},
{-1.5f, 2.4f, 0.0f},
{-1.297778f, 2.4f, -0.7622222f},
{-0.7622222f, 2.4f, -1.297778f},
{0.0f, 2.4f, -1.5f},
{0.7114074f, 2.4f, -1.211259f},
{1.211259f, 2.4f, -0.7114074f},
{1.4f, 2.4f, 0.0f},
{0.7034087f, 2.487499f, -1.19764f},
{1.19764f, 2.4875f, -0.7034087f},
{1.384259f, 2.4875f, 0.0f},
{0.7278751f, 2.4875f, -1.239297f},
{1.239298f, 2.4875f, -0.7278751f},
{1.432407f, 2.4875f, 0.0f},
{0.7622222f, 2.4f, -1.297778f},
{1.297778f, 2.4f, -0.7622222f},
{1.5f, 2.4f, 0.0f},
{0.0f, 2.4f, 1.4f},
{0.7114074f, 2.4f, 1.211259f},
{1.211259f, 2.4f, 0.7114074f},
{0.0f, 2.4875f, 1.384259f},
{0.7034087f, 2.487499f, 1.19764f},

{1.19764f, 2.4875f, 0.7034087f},
{0.0f, 2.4875f, 1.432407f},
{0.7278751f, 2.4875f, 1.239297f},
{1.239298f, 2.4875f, 0.7278751f},
{0.0f, 2.4f, 1.5f},
{0.7622222f, 2.4f, 1.297778f},
{1.297778f, 2.4f, 0.7622222f},
{-1.211259f, 2.4f, 0.7114074f},
{-0.7114074f, 2.4f, 1.211259f},
{-1.19764f, 2.487499f, 0.7034087f},
{-0.7034087f, 2.4875f, 1.19764f},
{-1.239297f, 2.4875f, 0.7278751f},
{-0.7278751f, 2.4875f, 1.239298f},
{-1.297778f, 2.4f, 0.7622222f},
{-0.7622222f, 2.4f, 1.297778f},
{-1.740741f, 1.877778f, 0.0f},
{-1.506063f, 1.877777f, -0.8845539f},
{-0.884554f, 1.877778f, -1.506063f},
{0.0f, 1.877778f, -1.740741f},
{-1.925926f, 1.372222f, 0.0f},
{-1.666283f, 1.372222f, -0.9786556f},
{-0.9786556f, 1.372222f, -1.666283f},
{0.0f, 1.372222f, -1.925926f},
{-2.0f, 0.9f, 0.0f},
{-1.73037f, 0.9f, -1.016296f},
{-1.016296f, 0.9000001f, -1.730371f},
{0.0f, 0.9f, -2.0f},
{0.8845539f, 1.877777f, -1.506063f},
{1.506063f, 1.877778f, -0.884554f},
{1.740741f, 1.877778f, 0.0f},
{0.9786556f, 1.372222f, -1.666283f},
{1.666283f, 1.372222f, -0.9786556f},
{1.925926f, 1.372222f, 0.0f},
{1.016296f, 0.9f, -1.73037f},
{1.730371f, 0.9000001f, -1.016296f},
{2.0f, 0.9f, 0.0f},
{0.0f, 1.877778f, 1.740741f},
{0.8845539f, 1.877777f, 1.506063f},
{1.506063f, 1.877778f, 0.884554f},

{0.0f, 1.372222f, 1.925926f},
{0.9786556f, 1.372222f, 1.666283f},
{1.666283f, 1.372222f, 0.9786556f},
{0.0f, 0.9f, 2.0f},
{1.016296f, 0.9f, 1.73037f},
{1.730371f, 0.9000001f, 1.016296f},
{-1.506063f, 1.877777f, 0.8845539f},
{-0.884554f, 1.877778f, 1.506063f},
{-1.666283f, 1.372222f, 0.9786556f},
{-0.9786556f, 1.372222f, 1.666283f},
{-1.73037f, 0.9f, 1.016296f},
{-1.016296f, 0.9000001f, 1.730371f},
{-1.87037f, 0.5222222f, 0.0f},
{-1.618216f, 0.5222221f, -0.950425f},
{-0.950425f, 0.5222223f, -1.618217f},
{0.0f, 0.5222222f, -1.87037f},
{-1.62963f, 0.2777778f, 0.0f},
{-1.409931f, 0.2777777f, -0.8280932f},
{-0.8280932f, 0.2777778f, -1.409931f},
{0.0f, 0.2777778f, -1.62963f},
{-1.5f, 0.15f, 0.0f},
{-1.297778f, 0.15f, -0.7622222f},
{-0.7622222f, 0.15f, -1.297778f},
{0.0f, 0.15f, -1.5f},
{0.950425f, 0.5222221f, -1.618216f},
{1.618217f, 0.5222223f, -0.950425f},
{1.87037f, 0.5222222f, 0.0f},
{0.8280932f, 0.2777777f, -1.409931f},
{1.409931f, 0.2777778f, -0.8280932f},
{1.62963f, 0.2777778f, 0.0f},
{0.7622222f, 0.15f, -1.297778f},
{1.297778f, 0.15f, -0.7622222f},
{1.5f, 0.15f, 0.0f},
{0.0f, 0.5222222f, 1.87037f},
{0.950425f, 0.5222221f, 1.618216f},
{1.618217f, 0.5222223f, 0.950425f},
{0.0f, 0.2777778f, 1.62963f},
{0.8280932f, 0.2777777f, 1.409931f},
{1.409931f, 0.2777778f, 0.8280932f},

```
    {0.0f, 0.15f, 1.5f},
    {0.7622222f, 0.15f, 1.297778f},
    {1.297778f, 0.15f, 0.7622222f},
    {-1.618216f, 0.5222221f, 0.950425f},
    {-0.950425f, 0.5222223f, 1.618217f},
    {-1.409931f, 0.2777777f, 0.8280932f},
    {-0.8280932f, 0.2777778f, 1.409931f},
    {-1.297778f, 0.15f, 0.7622222f},
    {-0.7622222f, 0.15f, 1.297778f},
    {-1.427778f, 0.07777777f, 0.0f},
    {-1.235292f, 0.07777776f, -0.7255225f},
    {-0.7255225f, 0.07777776f, -1.235292f},
    {0.0f, 0.07777777f, -1.427778f},
    {-1.022222f, 0.02222222f, 0.0f},
    {-0.8844114f, 0.02222222f, -0.5194403f},
    {-0.5194403f, 0.02222222f, -0.8844114f},
    {0.0f, 0.02222222f, -1.022222f},
    {0.0f, 0.0f, 0.0f},
    {0.7255225f, 0.07777776f, -1.235292f},
    {1.235292f, 0.07777776f, -0.7255225f},
    {1.427778f, 0.07777777f, 0.0f},
    {0.5194403f, 0.02222222f, -0.8844114f},
    {0.8844114f, 0.02222222f, -0.5194403f},
    {1.022222f, 0.02222222f, 0.0f},
    {0.0f, 0.07777777f, 1.427778f},
    {0.7255225f, 0.07777776f, 1.235292f},
    {1.235292f, 0.07777776f, 0.7255225f},
    {0.0f, 0.02222222f, 1.022222f},
    {0.5194403f, 0.02222222f, 0.8844114f},
    {0.8844114f, 0.02222222f, 0.5194403f},
    {-1.235292f, 0.07777776f, 0.7255225f},
    {-0.7255225f, 0.07777776f, 1.235292f},
    {-0.8844114f, 0.02222222f, 0.5194403f},
    {-0.5194403f, 0.02222222f, 0.8844114f},
};

static const face_t teapot552_faces[] = {
    {{1, 4, 0}},
    {{4, 1, 5}},
```

{{2, 5, 1}},
{{5, 2, 6}},
{{3, 6, 2}},
{{6, 3, 7}},
{{5, 8, 4}},
{{8, 5, 9}},
{{6, 9, 5}},
{{9, 6, 10}},
{{7, 10, 6}},
{{10, 7, 11}},
{{9, 12, 8}},
{{12, 9, 13}},
{{10, 13, 9}},
{{13, 10, 14}},
{{11, 14, 10}},
{{14, 11, 15}},
{{4, 16, 0}},
{{16, 4, 18}},
{{18, 17, 16}},
{{17, 18, 19}},
{{19, 3, 17}},
{{3, 19, 7}},
{{8, 18, 4}},
{{18, 8, 20}},
{{20, 19, 18}},
{{19, 20, 21}},
{{21, 7, 19}},
{{7, 21, 11}},
{{12, 20, 8}},
{{20, 12, 22}},
{{22, 21, 20}},
{{21, 22, 23}},
{{23, 11, 21}},
{{11, 23, 15}},
{{13, 24, 12}},
{{24, 13, 25}},
{{14, 25, 13}},
{{25, 14, 26}},
{{15, 26, 14}},

{{26, 15, 27}},
{{25, 28, 24}},
{{28, 25, 29}},
{{26, 29, 25}},
{{29, 26, 30}},
{{27, 30, 26}},
{{30, 27, 31}},
{{29, 32, 28}},
{{32, 29, 33}},
{{30, 33, 29}},
{{33, 30, 34}},
{{31, 34, 30}},
{{34, 31, 35}},
{{24, 22, 12}},
{{22, 24, 36}},
{{36, 23, 22}},
{{23, 36, 37}},
{{37, 15, 23}},
{{15, 37, 27}},
{{28, 36, 24}},
{{36, 28, 38}},
{{38, 37, 36}},
{{37, 38, 39}},
{{39, 27, 37}},
{{27, 39, 31}},
{{32, 38, 28}},
{{38, 32, 40}},
{{40, 39, 38}},
{{39, 40, 41}},
{{41, 31, 39}},
{{31, 41, 35}},
{{43, 46, 42}},
{{46, 43, 47}},
{{44, 47, 43}},
{{47, 44, 48}},
{{45, 48, 44}},
{{48, 45, 49}},
{{47, 50, 46}},
{{50, 47, 51}},

{{48, 51, 47}},
{{51, 48, 52}},
{{49, 52, 48}},
{{52, 49, 53}},
{{51, 54, 50}},
{{54, 51, 55}},
{{52, 55, 51}},
{{55, 52, 56}},
{{53, 56, 52}},
{{56, 53, 57}},
{{46, 58, 42}},
{{58, 46, 60}},
{{60, 59, 58}},
{{59, 60, 61}},
{{61, 45, 59}},
{{45, 61, 49}},
{{50, 60, 46}},
{{60, 50, 62}},
{{62, 61, 60}},
{{61, 62, 63}},
{{63, 49, 61}},
{{49, 63, 53}},
{{54, 62, 50}},
{{62, 54, 64}},
{{64, 63, 62}},
{{63, 64, 65}},
{{65, 53, 63}},
{{53, 65, 57}},
{{55, 66, 54}},
{{66, 55, 67}},
{{56, 67, 55}},
{{67, 56, 68}},
{{57, 68, 56}},
{{68, 57, 69}},
{{67, 70, 66}},
{{70, 67, 71}},
{{68, 71, 67}},
{{71, 68, 72}},
{{69, 72, 68}},

{{72, 69, 73}},
{{71, 74, 70}},
{{74, 71, 75}},
{{72, 75, 71}},
{{75, 72, 76}},
{{73, 76, 72}},
{{76, 73, 77}},
{{66, 64, 54}},
{{64, 66, 78}},
{{78, 65, 64}},
{{65, 78, 79}},
{{79, 57, 65}},
{{57, 79, 69}},
{{70, 78, 66}},
{{78, 70, 80}},
{{80, 79, 78}},
{{79, 80, 81}},
{{81, 69, 79}},
{{69, 81, 73}},
{{74, 80, 70}},
{{80, 74, 82}},
{{82, 81, 80}},
{{81, 82, 83}},
{{83, 73, 81}},
{{73, 83, 77}},
{{84, 86, 85}},
{{84, 87, 86}},
{{84, 88, 87}},
{{86, 89, 85}},
{{89, 86, 90}},
{{87, 90, 86}},
{{90, 87, 91}},
{{88, 91, 87}},
{{91, 88, 92}},
{{90, 93, 89}},
{{93, 90, 94}},
{{91, 94, 90}},
{{94, 91, 95}},
{{92, 95, 91}},

{{95, 92, 96}},
{{84, 97, 88}},
{{84, 98, 97}},
{{84, 99, 98}},
{{97, 92, 88}},
{{92, 97, 100}},
{{98, 100, 97}},
{{100, 98, 101}},
{{99, 101, 98}},
{{101, 99, 102}},
{{100, 96, 92}},
{{96, 100, 103}},
{{101, 103, 100}},
{{103, 101, 104}},
{{102, 104, 101}},
{{104, 102, 105}},
{{106, 107, 84}},
{{107, 108, 84}},
{{108, 99, 84}},
{{109, 107, 106}},
{{107, 109, 110}},
{{110, 108, 107}},
{{108, 110, 111}},
{{111, 99, 108}},
{{99, 111, 102}},
{{112, 110, 109}},
{{110, 112, 113}},
{{113, 111, 110}},
{{111, 113, 114}},
{{114, 102, 111}},
{{102, 114, 105}},
{{85, 115, 84}},
{{115, 116, 84}},
{{116, 106, 84}},
{{89, 115, 85}},
{{115, 89, 117}},
{{117, 116, 115}},
{{116, 117, 118}},
{{118, 106, 116}},

{{106, 118, 109}},
{{93, 117, 89}},
{{117, 93, 119}},
{{119, 118, 117}},
{{118, 119, 120}},
{{120, 109, 118}},
{{109, 120, 112}},
{{94, 121, 93}},
{{121, 94, 122}},
{{95, 122, 94}},
{{122, 95, 123}},
{{96, 123, 95}},
{{123, 96, 124}},
{{122, 125, 121}},
{{125, 122, 126}},
{{123, 126, 122}},
{{126, 123, 127}},
{{124, 127, 123}},
{{127, 124, 128}},
{{126, 129, 125}},
{{129, 126, 130}},
{{127, 130, 126}},
{{130, 127, 131}},
{{128, 131, 127}},
{{131, 128, 132}},
{{103, 124, 96}},
{{124, 103, 133}},
{{104, 133, 103}},
{{133, 104, 134}},
{{105, 134, 104}},
{{134, 105, 135}},
{{133, 128, 124}},
{{128, 133, 136}},
{{134, 136, 133}},
{{136, 134, 137}},
{{135, 137, 134}},
{{137, 135, 138}},
{{136, 132, 128}},
{{132, 136, 139}},

{{137, 139, 136}},
{{139, 137, 140}},
{{138, 140, 137}},
{{140, 138, 141}},
{{142, 113, 112}},
{{113, 142, 143}},
{{143, 114, 113}},
{{114, 143, 144}},
{{144, 105, 114}},
{{105, 144, 135}},
{{145, 143, 142}},
{{143, 145, 146}},
{{146, 144, 143}},
{{144, 146, 147}},
{{147, 135, 144}},
{{135, 147, 138}},
{{148, 146, 145}},
{{146, 148, 149}},
{{149, 147, 146}},
{{147, 149, 150}},
{{150, 138, 147}},
{{138, 150, 141}},
{{121, 119, 93}},
{{119, 121, 151}},
{{151, 120, 119}},
{{120, 151, 152}},
{{152, 112, 120}},
{{112, 152, 142}},
{{125, 151, 121}},
{{151, 125, 153}},
{{153, 152, 151}},
{{152, 153, 154}},
{{154, 142, 152}},
{{142, 154, 145}},
{{129, 153, 125}},
{{153, 129, 155}},
{{155, 154, 153}},
{{154, 155, 156}},
{{156, 145, 154}},

{{145, 156, 148}},
{{158, 161, 157}},
{{161, 158, 162}},
{{159, 162, 158}},
{{162, 159, 163}},
{{160, 163, 159}},
{{163, 160, 164}},
{{162, 165, 161}},
{{165, 162, 166}},
{{163, 166, 162}},
{{166, 163, 167}},
{{164, 167, 163}},
{{167, 164, 168}},
{{166, 169, 165}},
{{169, 166, 170}},
{{167, 170, 166}},
{{170, 167, 171}},
{{168, 171, 167}},
{{171, 168, 172}},
{{173, 164, 160}},
{{164, 173, 176}},
{{174, 176, 173}},
{{176, 174, 177}},
{{175, 177, 174}},
{{177, 175, 178}},
{{176, 168, 164}},
{{168, 176, 179}},
{{177, 179, 176}},
{{179, 177, 180}},
{{178, 180, 177}},
{{180, 178, 181}},
{{179, 172, 168}},
{{172, 179, 182}},
{{180, 182, 179}},
{{182, 180, 183}},
{{181, 183, 180}},
{{183, 181, 184}},
{{188, 186, 185}},
{{186, 188, 189}},

{{189, 187, 186}},
{{187, 189, 190}},
{{190, 175, 187}},
{{175, 190, 178}},
{{191, 189, 188}},
{{189, 191, 192}},
{{192, 190, 189}},
{{190, 192, 193}},
{{193, 178, 190}},
{{178, 193, 181}},
{{194, 192, 191}},
{{192, 194, 195}},
{{195, 193, 192}},
{{193, 195, 196}},
{{196, 181, 193}},
{{181, 196, 184}},
{{161, 197, 157}},
{{197, 161, 199}},
{{199, 198, 197}},
{{198, 199, 200}},
{{200, 185, 198}},
{{185, 200, 188}},
{{165, 199, 161}},
{{199, 165, 201}},
{{201, 200, 199}},
{{200, 201, 202}},
{{202, 188, 200}},
{{188, 202, 191}},
{{169, 201, 165}},
{{201, 169, 203}},
{{203, 202, 201}},
{{202, 203, 204}},
{{204, 191, 202}},
{{191, 204, 194}},
{{170, 205, 169}},
{{205, 170, 206}},
{{171, 206, 170}},
{{206, 171, 207}},
{{172, 207, 171}},

{{207, 172, 208}},
{{206, 209, 205}},
{{209, 206, 210}},
{{207, 210, 206}},
{{210, 207, 211}},
{{208, 211, 207}},
{{211, 208, 212}},
{{210, 213, 209}},
{{213, 210, 214}},
{{211, 214, 210}},
{{214, 211, 215}},
{{212, 215, 211}},
{{215, 212, 216}},
{{182, 208, 172}},
{{208, 182, 217}},
{{183, 217, 182}},
{{217, 183, 218}},
{{184, 218, 183}},
{{218, 184, 219}},
{{217, 212, 208}},
{{212, 217, 220}},
{{218, 220, 217}},
{{220, 218, 221}},
{{219, 221, 218}},
{{221, 219, 222}},
{{220, 216, 212}},
{{216, 220, 223}},
{{221, 223, 220}},
{{223, 221, 224}},
{{222, 224, 221}},
{{224, 222, 225}},
{{226, 195, 194}},
{{195, 226, 227}},
{{227, 196, 195}},
{{196, 227, 228}},
{{228, 184, 196}},
{{184, 228, 219}},
{{229, 227, 226}},
{{227, 229, 230}},

{{230, 228, 227}},
{{228, 230, 231}},
{{231, 219, 228}},
{{219, 231, 222}},
{{232, 230, 229}},
{{230, 232, 233}},
{{233, 231, 230}},
{{231, 233, 234}},
{{234, 222, 231}},
{{222, 234, 225}},
{{205, 203, 169}},
{{203, 205, 235}},
{{235, 204, 203}},
{{204, 235, 236}},
{{236, 194, 204}},
{{194, 236, 226}},
{{209, 235, 205}},
{{235, 209, 237}},
{{237, 236, 235}},
{{236, 237, 238}},
{{238, 226, 236}},
{{226, 238, 229}},
{{213, 237, 209}},
{{237, 213, 239}},
{{239, 238, 237}},
{{238, 239, 240}},
{{240, 229, 238}},
{{229, 240, 232}},
{{214, 241, 213}},
{{241, 214, 242}},
{{215, 242, 214}},
{{242, 215, 243}},
{{216, 243, 215}},
{{243, 216, 244}},
{{242, 245, 241}},
{{245, 242, 246}},
{{243, 246, 242}},
{{246, 243, 247}},
{{244, 247, 243}},

{{247, 244, 248}},
{{246, 249, 245}},
{{249, 246, 250}},
{{247, 250, 246}},
{{250, 247, 251}},
{{248, 251, 247}},
{{251, 248, 252}},
{{223, 244, 216}},
{{244, 223, 253}},
{{224, 253, 223}},
{{253, 224, 254}},
{{225, 254, 224}},
{{254, 225, 255}},
{{253, 248, 244}},
{{248, 253, 256}},
{{254, 256, 253}},
{{256, 254, 257}},
{{255, 257, 254}},
{{257, 255, 258}},
{{256, 252, 248}},
{{252, 256, 259}},
{{257, 259, 256}},
{{259, 257, 260}},
{{258, 260, 257}},
{{260, 258, 261}},
{{262, 233, 232}},
{{233, 262, 263}},
{{263, 234, 233}},
{{234, 263, 264}},
{{264, 225, 234}},
{{225, 264, 255}},
{{265, 263, 262}},
{{263, 265, 266}},
{{266, 264, 263}},
{{264, 266, 267}},
{{267, 255, 264}},
{{255, 267, 258}},
{{268, 266, 265}},
{{266, 268, 269}},

{{269, 267, 266}},
{{267, 269, 270}},
{{270, 258, 267}},
{{258, 270, 261}},
{{241, 239, 213}},
{{239, 241, 271}},
{{271, 240, 239}},
{{240, 271, 272}},
{{272, 232, 240}},
{{232, 272, 262}},
{{245, 271, 241}},
{{271, 245, 273}},
{{273, 272, 271}},
{{272, 273, 274}},
{{274, 262, 272}},
{{262, 274, 265}},
{{249, 273, 245}},
{{273, 249, 275}},
{{275, 274, 273}},
{{274, 275, 276}},
{{276, 265, 274}},
{{265, 276, 268}},
{{250, 277, 249}},
{{277, 250, 278}},
{{251, 278, 250}},
{{278, 251, 279}},
{{252, 279, 251}},
{{279, 252, 280}},
{{278, 281, 277}},
{{281, 278, 282}},
{{279, 282, 278}},
{{282, 279, 283}},
{{280, 283, 279}},
{{283, 280, 284}},
{{282, 285, 281}},
{{283, 285, 282}},
{{284, 285, 283}},
{{259, 280, 252}},
{{280, 259, 286}},

{{260, 286, 259}},
{{286, 260, 287}},
{{261, 287, 260}},
{{287, 261, 288}},
{{286, 284, 280}},
{{284, 286, 289}},
{{287, 289, 286}},
{{289, 287, 290}},
{{288, 290, 287}},
{{290, 288, 291}},
{{289, 285, 284}},
{{290, 285, 289}},
{{291, 285, 290}},
{{292, 269, 268}},
{{269, 292, 293}},
{{293, 270, 269}},
{{270, 293, 294}},
{{294, 261, 270}},
{{261, 294, 288}},
{{295, 293, 292}},
{{293, 295, 296}},
{{296, 294, 293}},
{{294, 296, 297}},
{{297, 288, 294}},
{{288, 297, 291}},
{{295, 285, 296}},
{{296, 285, 297}},
{{297, 285, 291}},
{{277, 275, 249}},
{{275, 277, 298}},
{{298, 276, 275}},
{{276, 298, 299}},
{{299, 268, 276}},
{{268, 299, 292}},
{{281, 298, 277}},
{{298, 281, 300}},
{{300, 299, 298}},
{{299, 300, 301}},
{{301, 292, 299}},

```
    {{292, 301, 295}},
    {{281, 285, 300}},
    {{300, 285, 301}},
    {{301, 285, 295}},
};

void model_make_teapot_552(model_t *m)
{
    memset(m, 0, sizeof(*m));
    strncpy(m->name, "teapot_552", sizeof(m->name) - 1);

    int vcount = (int)(sizeof(teapot552_verts) / sizeof(teapot552_verts[0]));
    int fcount = (int)(sizeof(teapot552_faces) / sizeof(teapot552_faces[0]));

    if (vcount > MODEL_MAX_VERTS) vcount = MODEL_MAX_VERTS;
    if (fcount > MODEL_MAX_FACES) fcount = MODEL_MAX_FACES;

    memcpy(m->verts, teapot552_verts, (size_t)vcount * sizeof(vec3_t));
    memcpy(m->faces, teapot552_faces, (size_t)fcount * sizeof(face_t));
    m->num_verts = vcount;
    m->num_faces = fcount;
}


void model_make_saturn(model_t *m)
{

    model_make_icosphere(m, 1);
    strncpy(m->name, "saturn", sizeof(m->name) - 1);


    for (int i = 0; i < m->num_verts; i++) {
        m->verts[i].x *= 0.55f;
        m->verts[i].y *= 0.55f;
        m->verts[i].z *= 0.55f;
    }
    int body_vert_count = m->num_verts;
```

```
    int   major_seg = 24, minor_seg = 5;
    float R = 1.1f, r = 0.15f;
    float tilt = 27.0f * (float)M_PI / 180.0f;
    float ct = cosf(tilt), st = sinf(tilt);

    for (int i = 0; i < major_seg; i++) {
        float theta = 2.0f * (float)M_PI * i / major_seg;
        for (int j = 0; j < minor_seg; j++) {
            float phi = 2.0f * (float)M_PI * j / minor_seg;
            if (m->num_verts >= MODEL_MAX_VERTS) goto ring_done;
            float rx = (R + r * cosf(phi)) * cosf(theta);
            float ry = r * sinf(phi);
            float rz = (R + r * cosf(phi)) * sinf(theta);

            m->verts[m->num_verts++] = (vec3_t){
                rx,
                ry * ct - rz * st,
                ry * st + rz * ct
            };
        }
    }

    int ring_base = body_vert_count;
    for (int i = 0; i < major_seg; i++) {
        int ni = (i + 1) % major_seg;
        for (int j = 0; j < minor_seg; j++) {
            int nj = (j + 1) % minor_seg;
            int a = ring_base + i  * minor_seg + j;
            int b = ring_base + ni * minor_seg + j;
            int c = ring_base + ni * minor_seg + nj;
            int d = ring_base + i  * minor_seg + nj;
            if (m->num_faces + 1 >= MODEL_MAX_FACES) goto ring_done;
            m->faces[m->num_faces++] = (face_t){{ a, b, c }};
            m->faces[m->num_faces++] = (face_t){{ a, c, d }};
        }
    }
ring_done:;
}
```

```
static void add_box(model_t *m,
                    float x0, float y0, float z0,
                    float x1, float y1, float z1)
{
   int b = m->num_verts;

   m->verts[m->num_verts++] = (vec3_t){ x0, y0, z0 };
   m->verts[m->num_verts++] = (vec3_t){ x1, y0, z0 };
   m->verts[m->num_verts++] = (vec3_t){ x1, y0, z1 };
   m->verts[m->num_verts++] = (vec3_t){ x0, y0, z1 };
   m->verts[m->num_verts++] = (vec3_t){ x0, y1, z0 };
   m->verts[m->num_verts++] = (vec3_t){ x1, y1, z0 };
   m->verts[m->num_verts++] = (vec3_t){ x1, y1, z1 };
   m->verts[m->num_verts++] = (vec3_t){ x0, y1, z1 };


   m->faces[m->num_faces++] = (face_t){{ b+0, b+2, b+1 }};
   m->faces[m->num_faces++] = (face_t){{ b+0, b+3, b+2 }};

   m->faces[m->num_faces++] = (face_t){{ b+4, b+5, b+6 }};
   m->faces[m->num_faces++] = (face_t){{ b+4, b+6, b+7 }};

   m->faces[m->num_faces++] = (face_t){{ b+0, b+1, b+5 }};
   m->faces[m->num_faces++] = (face_t){{ b+0, b+5, b+4 }};

   m->faces[m->num_faces++] = (face_t){{ b+2, b+3, b+7 }};
   m->faces[m->num_faces++] = (face_t){{ b+2, b+7, b+6 }};

   m->faces[m->num_faces++] = (face_t){{ b+3, b+0, b+4 }};
   m->faces[m->num_faces++] = (face_t){{ b+3, b+4, b+7 }};

   m->faces[m->num_faces++] = (face_t){{ b+1, b+2, b+6 }};
   m->faces[m->num_faces++] = (face_t){{ b+1, b+6, b+5 }};
}

static void add_cylinder(model_t *m, float cx, float cy_bot, float cy_top,
                         float cz, float r, int seg)
{
```

```
    int bot_centre = m->num_verts;
    m->verts[m->num_verts++] = (vec3_t){ cx, cy_bot, cz };
    int top_centre = m->num_verts;
    m->verts[m->num_verts++] = (vec3_t){ cx, cy_top, cz };

    int ring_bot = m->num_verts;
    for (int i = 0; i < seg; i++) {
        float a = 2.0f * (float)M_PI * i / seg;
        m->verts[m->num_verts++] = (vec3_t){ cx + r*cosf(a), cy_bot, cz + r*sinf(a) };
    }
    int ring_top = m->num_verts;
    for (int i = 0; i < seg; i++) {
        float a = 2.0f * (float)M_PI * i / seg;
        m->verts[m->num_verts++] = (vec3_t){ cx + r*cosf(a), cy_top, cz + r*sinf(a) };
    }

    for (int i = 0; i < seg; i++) {
        int ni = (i + 1) % seg;

        m->faces[m->num_faces++] = (face_t){{ bot_centre, ring_bot+i, ring_bot+ni }};

        m->faces[m->num_faces++] = (face_t){{ top_centre, ring_top+ni, ring_top+i }};

        m->faces[m->num_faces++] = (face_t){{ ring_bot+i, ring_top+i,  ring_top+ni }};
        m->faces[m->num_faces++] = (face_t){{ ring_bot+i, ring_top+ni, ring_bot+ni }};
    }
}

void model_make_lego(model_t *m)
{
    memset(m, 0, sizeof(*m));
    strncpy(m->name, "lego", sizeof(m->name) - 1);


    float bx = 1.6f, by = 0.6f, bz = 0.8f;
    add_box(m, -bx, -by, -bz, bx, by, bz);


    int   seg    = 8;
```

```
    float stud_r  = 0.22f;
    float stud_h  = 0.22f;
    float stud_y0 = by;
    float stud_y1 = by + stud_h;

    float xs[4] = { -1.2f, -0.4f, 0.4f, 1.2f };
    float zs[2] = { -0.4f,  0.4f };
    for (int zi = 0; zi < 2; zi++)
        for (int xi = 0; xi < 4; xi++)
            add_cylinder(m, xs[xi], stud_y0, stud_y1, zs[zi], stud_r, seg);
}


void model_make_dna(model_t *m)
{
    memset(m, 0, sizeof(*m));
    strncpy(m->name, "dna", sizeof(m->name) - 1);

    int   steps    = 20;
    int   tube_seg = 4;
    float helix_r  = 0.55f;
    float tube_r   = 0.09f;
    float pitch    = 2.0f;
    float total_h  = 2.0f;
    float y_bot    = -total_h * 0.5f;


    int ring_base[2][21];


    for (int strand = 0; strand < 2; strand++) {
        float phase    = strand * (float)M_PI;
        int   prev_ring = -1;

        for (int s = 0; s <= steps; s++) {
            float t     = (float)s / steps;
            float y     = y_bot + t * total_h;
            float angle = 2.0f * (float)M_PI * t * (total_h / pitch) + phase;
```

```
float cx = helix_r * cosf(angle);
float cz = helix_r * sinf(angle);


float da   = 2.0f * (float)M_PI * (total_h / pitch) / steps;
float tx   = -helix_r * sinf(angle) * da;
float ty   =  total_h / steps;
float tz   =  helix_r * cosf(angle) * da;
float tlen = sqrtf(tx*tx + ty*ty + tz*tz);
if (tlen > 1e-6f) { tx/=tlen; ty/=tlen; tz/=tlen; }


float ux = tz, uy = 0.0f, uz = -tx;
float ulen = sqrtf(ux*ux + uz*uz);
if (ulen < 1e-6f) { ux = 1.0f; uz = 0.0f; }
else { ux/=ulen; uz/=ulen; }

float vx = ty*uz - tz*uy;
float vy = tz*ux - tx*uz;
float vz = tx*uy - ty*ux;

if (m->num_verts + tube_seg > MODEL_MAX_VERTS) goto dna_done;
ring_base[strand][s] = m->num_verts;

for (int k = 0; k < tube_seg; k++) {
   float a = 2.0f * (float)M_PI * k / tube_seg;
   float ca = cosf(a), sa = sinf(a);
   m->verts[m->num_verts++] = (vec3_t){
      cx + tube_r*(ca*ux + sa*vx),
      y  + tube_r*(ca*uy + sa*vy),
      cz + tube_r*(ca*uz + sa*vz)
   };
}

if (prev_ring >= 0) {
   int cr = ring_base[strand][s];
   int pr = prev_ring;
   for (int k = 0; k < tube_seg; k++) {
      int nk = (k + 1) % tube_seg;
```

```
            if (m->num_faces + 2 > MODEL_MAX_FACES) goto dna_done;
            m->faces[m->num_faces++] = (face_t){{ pr+k, cr+k,  cr+nk }};
            m->faces[m->num_faces++] = (face_t){{ pr+k, cr+nk, pr+nk }};
          }
        }
        prev_ring = ring_base[strand][s];
    }
}


float rung_hw = 0.06f;
for (int s = 0; s <= steps; s += 2) {

    float t      = (float)s / steps;
    float y      = y_bot + t * total_h;
    float angle0 = 2.0f * (float)M_PI * t * (total_h / pitch);
    float angle1 = angle0 + (float)M_PI;

    float x0 = helix_r * cosf(angle0), z0 = helix_r * sinf(angle0);
    float x1 = helix_r * cosf(angle1), z1 = helix_r * sinf(angle1);


    float dx = x1-x0, dz = z1-z0;
    float dlen = sqrtf(dx*dx + dz*dz);
    if (dlen < 1e-6f) continue;

    float px = -dz/dlen * rung_hw;
    float pz =  dx/dlen * rung_hw;

    if (m->num_verts + 8 > MODEL_MAX_VERTS) break;
    if (m->num_faces + 12 > MODEL_MAX_FACES) break;


    int b = m->num_verts;
    m->verts[m->num_verts++] = (vec3_t){ x0+px, y-rung_hw, z0+pz };
    m->verts[m->num_verts++] = (vec3_t){ x0-px, y-rung_hw, z0-pz };
    m->verts[m->num_verts++] = (vec3_t){ x0-px, y+rung_hw, z0-pz };
    m->verts[m->num_verts++] = (vec3_t){ x0+px, y+rung_hw, z0+pz };
    m->verts[m->num_verts++] = (vec3_t){ x1+px, y-rung_hw, z1+pz };
```

```
        m->verts[m->num_verts++] = (vec3_t){ x1-px, y-rung_hw, z1-pz };
        m->verts[m->num_verts++] = (vec3_t){ x1-px, y+rung_hw, z1-pz };
        m->verts[m->num_verts++] = (vec3_t){ x1+px, y+rung_hw, z1+pz };


        m->faces[m->num_faces++] = (face_t){{ b+0, b+1, b+2 }};
        m->faces[m->num_faces++] = (face_t){{ b+0, b+2, b+3 }};

        m->faces[m->num_faces++] = (face_t){{ b+4, b+6, b+5 }};
        m->faces[m->num_faces++] = (face_t){{ b+4, b+7, b+6 }};

        m->faces[m->num_faces++] = (face_t){{ b+3, b+2, b+6 }};
        m->faces[m->num_faces++] = (face_t){{ b+3, b+6, b+7 }};

        m->faces[m->num_faces++] = (face_t){{ b+0, b+5, b+1 }};
        m->faces[m->num_faces++] = (face_t){{ b+0, b+4, b+5 }};

        m->faces[m->num_faces++] = (face_t){{ b+0, b+3, b+7 }};
        m->faces[m->num_faces++] = (face_t){{ b+0, b+7, b+4 }};

        m->faces[m->num_faces++] = (face_t){{ b+1, b+5, b+6 }};
        m->faces[m->num_faces++] = (face_t){{ b+1, b+6, b+2 }};
    }

dna_done:;
}


void model_make_rocket(model_t *m)
{
    memset(m, 0, sizeof(*m));
    strncpy(m->name, "rocket", sizeof(m->name) - 1);

    int   seg      = 10;
    float body_r   = 0.3f;
    float body_bot = -1.0f;
    float body_top =  0.4f;
    float nose_top =  1.2f;
```

```
int ring_bot = m->num_verts;
for (int i = 0; i < seg; i++) {
   float a = 2.0f * (float)M_PI * i / seg;
   m->verts[m->num_verts++] = (vec3_t){ body_r*cosf(a), body_bot, body_r*sinf(a) };
}
int ring_top = m->num_verts;
for (int i = 0; i < seg; i++) {
   float a = 2.0f * (float)M_PI * i / seg;
   m->verts[m->num_verts++] = (vec3_t){ body_r*cosf(a), body_top, body_r*sinf(a) };
}

int bot_c = m->num_verts;
m->verts[m->num_verts++] = (vec3_t){ 0.0f, body_bot, 0.0f };

for (int i = 0; i < seg; i++) {
   int ni = (i + 1) % seg;

   m->faces[m->num_faces++] = (face_t){{ ring_bot+i, ring_top+i,  ring_top+ni }};
   m->faces[m->num_faces++] = (face_t){{ ring_bot+i, ring_top+ni, ring_bot+ni }};

   m->faces[m->num_faces++] = (face_t){{ bot_c, ring_bot+ni, ring_bot+i }};
}


int lat      = 6;
float nose_h  = nose_top - body_top;
int prev_ring = ring_top;

for (int l = 1; l <= lat; l++) {
   float t = (float)l / lat;

   float y = body_top + nose_h * (1.0f - cosf(t * (float)M_PI / 2.0f));
   float r = body_r   * cosf(t * (float)M_PI / 2.0f);

   if (l == lat) {

      int apex = m->num_verts;
      m->verts[m->num_verts++] = (vec3_t){ 0.0f, nose_top, 0.0f };
```

```
        for (int i = 0; i < seg; i++) {
            int ni = (i + 1) % seg;
            m->faces[m->num_faces++] = (face_t){{ prev_ring+i, apex, prev_ring+ni }};
        }
    } else {
        int cur_ring = m->num_verts;
        for (int i = 0; i < seg; i++) {
            float a = 2.0f * (float)M_PI * i / seg;
            m->verts[m->num_verts++] = (vec3_t){ r*cosf(a), y, r*sinf(a) };
        }
        for (int i = 0; i < seg; i++) {
            int ni = (i + 1) % seg;
            m->faces[m->num_faces++] = (face_t){{ prev_ring+i, cur_ring+i,  cur_ring+ni }};
            m->faces[m->num_faces++] = (face_t){{ prev_ring+i, cur_ring+ni, prev_ring+ni }};
        }
        prev_ring = cur_ring;
    }
}


float fin_angles[4] = { 0.0f, (float)M_PI/2, (float)M_PI, 3.0f*(float)M_PI/2 };
float fin_out  = 0.85f;
float fin_bot  = body_bot;
float fin_top  = body_bot + 0.55f;
float fin_thick = 0.04f;

for (int f = 0; f < 4; f++) {
    float ca = cosf(fin_angles[f]);
    float sa = sinf(fin_angles[f]);


    float tx = (body_r + fin_out) * ca;
    float tz = (body_r + fin_out) * sa;

    float ibx = body_r * ca, ibz = body_r * sa;

    float itx = body_r * ca, itz = body_r * sa;
```

```
        float ox = -sa * fin_thick;
        float oz =  ca * fin_thick;

        int b = m->num_verts;
        m->verts[m->num_verts++] = (vec3_t){ ibx+ox, fin_bot, ibz+oz };
        m->verts[m->num_verts++] = (vec3_t){ ibx-ox, fin_bot, ibz-oz };
        m->verts[m->num_verts++] = (vec3_t){ tx,     fin_bot, tz     };
        m->verts[m->num_verts++] = (vec3_t){ itx+ox, fin_top, itz+oz  };
        m->verts[m->num_verts++] = (vec3_t){ itx-ox, fin_top, itz-oz  };


        m->faces[m->num_faces++] = (face_t){{ b+0, b+2, b+3 }};

        m->faces[m->num_faces++] = (face_t){{ b+1, b+4, b+2 }};

        m->faces[m->num_faces++] = (face_t){{ b+3, b+4, b+0 }};
        m->faces[m->num_faces++] = (face_t){{ b+4, b+1, b+0 }};
    }
}


void model_make_minifigure(model_t *m)
{
    memset(m, 0, sizeof(*m));
    strncpy(m->name, "minifig", sizeof(m->name) - 1);


    float leg_w   = 0.18f, leg_d = 0.18f;
    float leg_bot = -1.0f, leg_top = -0.35f;
    float leg_sep = 0.21f;

    add_box(m, -leg_sep-leg_w, leg_bot, -leg_d,
            -leg_sep+leg_w, leg_top,  leg_d);

    add_box(m,  leg_sep-leg_w, leg_bot, -leg_d,
            leg_sep+leg_w, leg_top,  leg_d);


    float hip_w = 0.42f, hip_h = 0.18f, hip_d = 0.18f;
```

```
add_box(m, -hip_w, leg_top, -hip_d, hip_w, leg_top+hip_h, hip_d);


float tor_w = 0.36f, tor_d = 0.16f;
float tor_bot = leg_top + hip_h;
float tor_top = tor_bot + 0.52f;
add_box(m, -tor_w, tor_bot, -tor_d, tor_w, tor_top, tor_d);


float arm_w = 0.22f, arm_h = 0.14f, arm_d = 0.13f;
float arm_y0 = tor_top - 0.16f;
float arm_y1 = arm_y0  - arm_h;

add_box(m, -(tor_w + arm_w*2), arm_y1, -arm_d,
         -(tor_w),           arm_y0,  arm_d);

add_box(m,  tor_w,             arm_y1, -arm_d,
          tor_w + arm_w*2,   arm_y0,  arm_d);


int   hand_seg = 6;
float hand_r   = 0.09f;
float hand_cx_l = -(tor_w + arm_w*2 + hand_r);
float hand_cx_r =  (tor_w + arm_w*2 + hand_r);
float hand_cy   = (arm_y0 + arm_y1) * 0.5f;
add_cylinder(m, hand_cx_l, hand_cy - hand_r, hand_cy + hand_r,
           0.0f, hand_r, hand_seg);
add_cylinder(m, hand_cx_r, hand_cy - hand_r, hand_cy + hand_r,
           0.0f, hand_r, hand_seg);


float neck_r   = 0.10f;
float neck_bot = tor_top;
float neck_top = tor_top + 0.10f;
add_cylinder(m, 0.0f, neck_bot, neck_top, 0.0f, neck_r, 6);


float head_r   = 0.30f;
float head_bot = neck_top;
```

```
    float head_top = head_bot + 0.38f;
    add_cylinder(m, 0.0f, head_bot, head_top, 0.0f, head_r, 10);


    float stud_r   = 0.10f;
    float stud_bot = head_top;
    float stud_top = head_top + 0.08f;
    add_cylinder(m, 0.0f, stud_bot, stud_top, 0.0f, stud_r, 6);
}
```

```
================================================================
FILE: software/userspace/usbkeyboard.h
================================================================
```

```
// usbkeyboard.h

#ifndef _USBKEYBOARD_H
#define _USBKEYBOARD_H

#include <libusb-1.0/libusb.h>

#define USB_HID_KEYBOARD_PROTOCOL 1


#define USB_LCTRL  (1 << 0)
#define USB_LSHIFT (1 << 1)
#define USB_LALT   (1 << 2)
#define USB_LGUI   (1 << 3)
#define USB_RCTRL  (1 << 4)
#define USB_RSHIFT (1 << 5)
#define USB_RALT   (1 << 6)
#define USB_RGUI   (1 << 7)

struct usb_keyboard_packet {
    uint8_t modifiers;
    uint8_t reserved;
    uint8_t keycode[6];
};
```

```
#define KEY_A      0x04
#define KEY_B      0x05
#define KEY_C      0x06
#define KEY_D      0x07
#define KEY_E      0x08
#define KEY_F      0x09
#define KEY_G      0x0a
#define KEY_H      0x0b
#define KEY_I     0x0c
#define KEY_J     0x0d
#define KEY_K      0x0e
#define KEY_L      0x0f
#define KEY_M      0x10
#define KEY_N      0x11
#define KEY_O      0x12
#define KEY_P      0x13
#define KEY_Q      0x14
#define KEY_R      0x15
#define KEY_S      0x16
#define KEY_T      0x17
#define KEY_U      0x18
#define KEY_V      0x19
#define KEY_W      0x1a
#define KEY_X      0x1b
#define KEY_Y      0x1c
#define KEY_Z      0x1d

#define KEY_1      0x1e
#define KEY_2      0x1f
#define KEY_3      0x20
#define KEY_4      0x21
#define KEY_5      0x22
#define KEY_6      0x23
#define KEY_7      0x24
#define KEY_8      0x25
#define KEY_9      0x26
#define KEY_0      0x27
```

```
#define KEY_ENTER      0x28
#define KEY_ESC        0x29
#define KEY_BACKSPACE  0x2a
#define KEY_TAB        0x2b
#define KEY_SPACE      0x2c
#define KEY_MINUS      0x2d
#define KEY_EQUAL      0x2e

#define KEY_UP    0x52
#define KEY_DOWN  0x51
#define KEY_LEFT  0x50
#define KEY_RIGHT 0x4f


extern struct libusb_device_handle *openkeyboard(uint8_t *endpoint_address);

#endif
```

```
================================================================
FILE: software/userspace/usbkeyboard.c
================================================================
```

```
// usbkeyboard.c

#include "usbkeyboard.h"

#include <stdio.h>
#include <stdlib.h>


struct libusb_device_handle *openkeyboard(uint8_t *endpoint_address)
{
    libusb_device **devs;
    struct libusb_device_handle *keyboard = NULL;
    struct libusb_device_descriptor desc;
    ssize_t num_devs, d;
    uint8_t i, k;
```

```
if (libusb_init(NULL) < 0) {
   fprintf(stderr, "Error: libusb_init failed\n");
   exit(1);
}

if ((num_devs = libusb_get_device_list(NULL, &devs)) < 0) {
   fprintf(stderr, "Error: libusb_get_device_list failed\n");
   exit(1);
}

for (d = 0; d < num_devs; d++) {
   libusb_device *dev = devs[d];
   if (libusb_get_device_descriptor(dev, &desc) < 0) {
      fprintf(stderr, "Error: libusb_get_device_descriptor failed\n");
      exit(1);
   }

   if (desc.bDeviceClass == LIBUSB_CLASS_PER_INTERFACE) {
      struct libusb_config_descriptor *config;
      libusb_get_config_descriptor(dev, 0, &config);
      for (i = 0; i < config->bNumInterfaces; i++)
         for (k = 0; k < config->interface[i].num_altsetting; k++) {
            const struct libusb_interface_descriptor *inter =
               config->interface[i].altsetting + k;
            if (inter->bInterfaceClass    == LIBUSB_CLASS_HID &&
               inter->bInterfaceProtocol == USB_HID_KEYBOARD_PROTOCOL) {
               int r;
               if ((r = libusb_open(dev, &keyboard)) != 0) {
                  fprintf(stderr, "Error: libusb_open failed: %d\n", r);
                  exit(1);
               }
               if (libusb_kernel_driver_active(keyboard, i))
                  libusb_detach_kernel_driver(keyboard, i);
               libusb_set_auto_detach_kernel_driver(keyboard, i);
               if ((r = libusb_claim_interface(keyboard, i)) != 0) {
                  fprintf(stderr, "Error: libusb_claim_interface failed: %d\n", r);
                  exit(1);
               }
```

```
                *endpoint_address = inter->endpoint[0].bEndpointAddress;
                goto found;
            }
        }
    }
}

found:
    libusb_free_device_list(devs, 1);
    return keyboard;
}
```

==========

FILE: software/userspace/rasterizer_test.c

==========

```
// rasterizer_test.c

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <sys/ioctl.h>
#include "avalon_kernel.h"

int main(void)
{
    int fd;
    rasterizer_arg_t ra;
    int ret;

    printf("=== rasterizer driver validation ===\n\n");


    printf("[1] opening /dev/rasterizer ... ");
```

```
fd = open("/dev/rasterizer", O_RDWR);
if (fd < 0) {
   printf("FAIL: %s\n", strerror(errno));
   printf("  Driver probably not loaded. Run:\n");
   printf("    sudo insmod avalon_kernel.ko\n");
   printf("  Then check dmesg | tail\n");
   return 1;
}
printf("OK (fd=%d)\n", fd);


printf("[2] ioctl(RASTERIZER_STATUS) ... ");
memset(&ra, 0, sizeof(ra));
ret = ioctl(fd, RASTERIZER_STATUS, &ra);
if (ret != 0) {
   printf("FAIL: %s\n", strerror(errno));
} else {
   printf("OK\n");
   printf("    fifo_level = %u\n", ra.status.fifo_level);
   printf("    fifo_full  = %u\n", ra.status.fifo_full);
   printf("    (values are bus garbage if no SV slave is connected)\n");
}


printf("[3] ioctl(RASTERIZER_SET_CONTROL) ... ");
memset(&ra, 0, sizeof(ra));
ra.control.irq_enable    = 0;
ra.control.low_watermark = 16;
ret = ioctl(fd, RASTERIZER_SET_CONTROL, &ra);
if (ret != 0)
   printf("FAIL: %s\n", strerror(errno));
else
   printf("OK (wrote irq_enable=0, low_watermark=16)\n");


printf("[4] ioctl(RASTERIZER_GET_CONTROL) ... ");
memset(&ra, 0, sizeof(ra));
ret = ioctl(fd, RASTERIZER_GET_CONTROL, &ra);
if (ret != 0) {
```

```
    printf("FAIL: %s\n", strerror(errno));
} else {
    printf("OK\n");
    printf("    irq_enable    = %u (expect 0)\n", ra.control.irq_enable);
    printf("    low_watermark = %u (expect 16)\n", ra.control.low_watermark);
    if (ra.control.irq_enable == 0 && ra.control.low_watermark == 16)
        printf("    PASS: shadow read matches what we wrote\n");
    else
        printf("    FAIL: shadow read did not match\n");
}


printf("[5] ioctl(RASTERIZER_SUBMIT) ... ");
memset(&ra, 0, sizeof(ra));
ra.packet.a0         = 0x11111111;
ra.packet.b0         = 0x22222222;
ra.packet.c0         = 0x33333333;
ra.packet.flags_color = 0x000001E0;
ret = ioctl(fd, RASTERIZER_SUBMIT, &ra);
if (ret != 0) {
    if (errno == EAGAIN)
        printf("EAGAIN (FIFO reported full — expected if STATUS returns 0xFF)\n");
    else
        printf("FAIL: %s\n", strerror(errno));
} else {
    printf("OK (17 iowrite32s + 1 commit issued to bus)\n");
}


printf("[6] ioctl(0xDEADBEEF) ... ");
ret = ioctl(fd, 0xDEADBEEF, &ra);
if (ret == 0)
    printf("FAIL: driver accepted unknown command\n");
else
    printf("OK (rejected with %s)\n", strerror(errno));

close(fd);
printf("\n=== done ===\n");
return 0;
```

```
}
```

```
================================================================
FILE: software/userspace/bottleneck_probe.c
================================================================
```

```
// bottleneck_probe.c

#include <errno.h>
#include <fcntl.h>
#include <math.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/mman.h>
#include <time.h>
#include <unistd.h>

#include "../kernel/avalon_kernel.h"
#include "geometry.h"
#include "model.h"

#define SCREEN_W 256
#define SCREEN_H 240
#define DEVICE   "/dev/rasterizer"

typedef struct {
    long status_polls;
    long fifo_full_polls;
    long submit_eagain;
    unsigned int max_fifo_level;
} submit_stats_t;

typedef struct {
    int faces_total;
    int triangles_built;
```

```
    int triangles_submitted;
    int triangles_culled;
    long bbox_pixels;
    long bbox_cycles_est;
    double project_sec;
    double setup_sec;
    double submit_sec;
    submit_stats_t submit;
} frame_stats_t;

static screen_vertex_t g_sv[MODEL_MAX_VERTS];
static volatile uint32_t *g_regs = NULL;

static double elapsed_sec(struct timespec a, struct timespec b)
{
    return (double)(b.tv_sec - a.tv_sec) +
           (double)(b.tv_nsec - a.tv_nsec) * 1e-9;
}

static void project_vertices(const model_t *model, const mat4_t *mvp)
{
    for (int i = 0; i < model->num_verts; i++) {
        vec3_t v = model->verts[i];
        vec4_t clip = mat4_mul_vec4(mvp, (vec4_t){v.x, v.y, v.z, 1.0f});

        if (fabsf(clip.w) < 1e-6f) clip.w = 1e-6f;
        float invw = 1.0f / clip.w;
        float ndcx = clip.x * invw;
        float ndcy = clip.y * invw;
        float ndcz = clip.z * invw;

        g_sv[i].sx = (ndcx + 1.0f) * 0.5f * SCREEN_W;
        g_sv[i].sy = (1.0f - ndcy) * 0.5f * SCREEN_H;

        float sz = (ndcz + 1.0f) * 0.5f;
        if (sz < 0.0f) sz = 0.0f;
        if (sz > 1.0f) sz = 1.0f;
        g_sv[i].sz = sz;
    }
```

```
}

static inline void mmio_barrier(void)
{
#if defined(__arm__) || defined(__aarch64__)
   __asm__ volatile ("dsb sy" ::: "memory");
#else
   __sync_synchronize();
#endif
}

static uint32_t read_status_mmio(submit_stats_t *stats, long *status_polls)
{
   uint32_t status = g_regs[RAST_STATUS_OFFSET / 4];
   unsigned int level = status & RAST_STATUS_LEVEL_MASK;

   if (stats && level > stats->max_fifo_level)
      stats->max_fifo_level = level;
   if (stats)
      stats->status_polls++;
   if (status_polls)
      (*status_polls)++;

   return status;
}

static int submit_triangle(int fd, const triangle_packet_t *pkt,
                           submit_stats_t *stats)
{
   if (g_regs) {
      const uint32_t *w = (const uint32_t *)pkt;
      uint32_t status;


      status = read_status_mmio(stats, NULL);
      while (status & RAST_STATUS_FULL_BIT) {
         stats->fifo_full_polls++;
         status = read_status_mmio(stats, NULL);
      }
```

```
        for (int i = 0; i < RAST_PACKET_NUM_WORDS; i++)
            g_regs[i] = w[i];


        mmio_barrier();
        g_regs[RAST_COMMIT_OFFSET / 4] = 1;
        return 0;
    }

    rasterizer_arg_t ra;
    memset(&ra, 0, sizeof(ra));
    ra.packet = *pkt;
    while (ioctl(fd, RASTERIZER_SUBMIT, &ra) != 0) {
        if (errno != EAGAIN) return -1;
        stats->submit_eagain++;
    }
    return 0;
}

static int issue_present(int fd)
{
    if (g_regs) {
        mmio_barrier();
        g_regs[RAST_PRESENT_OFFSET / 4] = 1;
        return 0;
    }

    return ioctl(fd, RASTERIZER_PRESENT);
}

static int wait_present(int fd, long *status_polls)
{
    rasterizer_arg_t ra;

    if (g_regs) {
        uint32_t status;

        do {
```

```
            status = read_status_mmio(NULL, status_polls);
        } while (!(status & RAST_STATUS_SWAP_BUSY_BIT));

        for (;;) {
            status = read_status_mmio(NULL, status_polls);
            if (!(status & RAST_STATUS_SWAP_BUSY_BIT))
                return 0;
        }
    } else {
        do {
            memset(&ra, 0, sizeof(ra));
            if (ioctl(fd, RASTERIZER_STATUS, &ra) < 0)
                return -1;
            (*status_polls)++;
        } while (!ra.status.swap_busy);

        for (;;) {
            memset(&ra, 0, sizeof(ra));
            if (ioctl(fd, RASTERIZER_STATUS, &ra) < 0)
                return -1;
            (*status_polls)++;
            if (!ra.status.swap_busy)
                return 0;
        }
    }
}

static int render_frame_profile(int fd, int dry_run, const model_t *model,
                                float rot_x, float rot_y, float rot_z,
                                float cam_dist, frame_stats_t *stats)
{
    struct timespec t0, t1;
    float aspect = (float)SCREEN_W / (float)SCREEN_H;

    mat4_t proj = perspective(60.0f, aspect, 0.1f, 100.0f);
    mat4_t view = translation(0.0f, 0.0f, -cam_dist);
    mat4_t rz = rotation_z(rot_z);
    mat4_t ry = rotation_y(rot_y);
    mat4_t rx = rotation_x(rot_x);
```

```
mat4_t ryx = mat4_mul(&ry, &rx);
mat4_t model_mat = mat4_mul(&rz, &ryx);
mat4_t vp = mat4_mul(&proj, &view);
mat4_t mvp = mat4_mul(&vp, &model_mat);

vec3_t light_dir = vec3_normalize((vec3_t){0.4f, 0.7f, 0.5f});
float base_r = 0.2f, base_g = 0.7f, base_b = 1.0f;

memset(stats, 0, sizeof(*stats));
stats->faces_total = model->num_faces;

clock_gettime(CLOCK_MONOTONIC, &t0);
project_vertices(model, &mvp);
clock_gettime(CLOCK_MONOTONIC, &t1);
stats->project_sec += elapsed_sec(t0, t1);

for (int i = 0; i < model->num_faces; i++) {
   const face_t *f = &model->faces[i];

   vec3_t p0 = model->verts[f->v[0]];
   vec3_t p1 = model->verts[f->v[1]];
   vec3_t p2 = model->verts[f->v[2]];
   vec3_t face_n = vec3_normalize(
      vec3_cross(vec3_sub(p1, p0), vec3_sub(p2, p0)));
   vec4_t tn = mat4_mul_vec4(&model_mat,
                        (vec4_t){face_n.x, face_n.y, face_n.z, 0.0f});
   vec3_t wn = vec3_normalize((vec3_t){tn.x, tn.y, tn.z});
   uint8_t color = shade_face(wn, light_dir, base_r, base_g, base_b);

   triangle_packet_t pkt;
   clock_gettime(CLOCK_MONOTONIC, &t0);
   int ok = setup_triangle(&g_sv[f->v[0]], &g_sv[f->v[1]],
                        &g_sv[f->v[2]], color, &pkt);
   clock_gettime(CLOCK_MONOTONIC, &t1);
   stats->setup_sec += elapsed_sec(t0, t1);

   if (ok < 0) {
      stats->triangles_culled++;
      continue;
```

```
        }

        int xmin = pkt.bbox_packed & 0xFF;
        int xmax = (pkt.bbox_packed >> 8) & 0xFF;
        int ymin = (pkt.bbox_packed >> 16) & 0xFF;
        int ymax = (pkt.bbox_packed >> 24) & 0xFF;
        int width = xmax - xmin + 1;
        int height = ymax - ymin + 1;
        int column_banks = (width + 15) / 16;

        stats->triangles_built++;
        if (width > 0 && height > 0) {
            stats->bbox_pixels += (long)width * height;
            stats->bbox_cycles_est += (long)column_banks * height;
        }

        if (!dry_run) {
            clock_gettime(CLOCK_MONOTONIC, &t0);
            if (submit_triangle(fd, &pkt, &stats->submit) < 0)
                return -1;
            clock_gettime(CLOCK_MONOTONIC, &t1);
            stats->submit_sec += elapsed_sec(t0, t1);
        }
        stats->triangles_submitted++;
    }

    return 0;
}

static void make_builtin_model(model_t *m, int idx)
{
    switch (idx) {
    case 0: model_make_cube(m); break;
    case 1: model_make_icosphere(m, 1); break;
    case 2: model_make_icosphere(m, 2); break;
    case 3: model_make_torus(m, 12, 8, 0.7f, 0.3f); break;
    case 4: model_make_teapot(m); break;
    case 5: model_make_saturn(m); break;
    case 6: model_make_lego(m); break;
```

```
    case 7: model_make_dna(m); break;
    case 8: model_make_teapot_552(m); break;
    case 9: model_make_minifigure(m); break;
    default: model_make_icosphere(m, 2); break;
    }
}

static void usage(const char *argv0)
{
    fprintf(stderr,
            "usage: %s [--dry-run] [--frames N] [--model N] [objfile]\n"
            "  --dry-run   run ARM setup math only; do not open /dev/rasterizer\n"
            "  --frames N  number of frames to measure, default 60\n"
            "  --model N   built-in model index 0..9, default 2\n",
            argv0);
}

int main(int argc, char **argv)
{
    int dry_run = 0;
    int frames = 60;
    int model_idx = 2;
    const char *obj_path = NULL;

    for (int i = 1; i < argc; i++) {
        if (strcmp(argv[i], "--dry-run") == 0) {
            dry_run = 1;
        } else if (strcmp(argv[i], "--frames") == 0 && i + 1 < argc) {
            frames = atoi(argv[++i]);
        } else if (strcmp(argv[i], "--model") == 0 && i + 1 < argc) {
            model_idx = atoi(argv[++i]);
        } else if (argv[i][0] == '-') {
            usage(argv[0]);
            return 2;
        } else {
            obj_path = argv[i];
        }
    }
```

```
if (frames <= 0) frames = 1;

model_t model;
if (obj_path) {
   if (model_load_obj(&model, obj_path) < 0) {
      fprintf(stderr, "could not load OBJ %s\n", obj_path);
      return 1;
   }
} else {
   make_builtin_model(&model, model_idx);
}

int fd = -1;
if (!dry_run) {
   fd = open(DEVICE, O_RDWR);
   if (fd < 0) {
      fprintf(stderr, "cannot open %s: %s\n", DEVICE, strerror(errno));
      fprintf(stderr, "try --dry-run for software-only profiling\n");
      return 1;
   }
   void *mapped = mmap(NULL, 4096, PROT_READ | PROT_WRITE,
                       MAP_SHARED, fd, 0);
   if (mapped != MAP_FAILED) {
      g_regs = (volatile uint32_t *)mapped;
      fprintf(stderr, "mmap OK: direct MMIO submission enabled\n");
   } else {
      g_regs = NULL;
      fprintf(stderr, "mmap failed (%s); using ioctl fallback\n",
              strerror(errno));
   }
}

printf("frame,model,faces,built,culled,render_ms,project_ms,setup_ms,submit_ms,"
       "present_ms,fps,fifo_max,fifo_full_polls,eagain,status_polls,"
       "bbox_avg_px,hw_cycle_est\n");

for (int frame = 0; frame < frames; frame++) {
   struct timespec t0, t1, tp0, tp1;
   frame_stats_t stats;
```

```
long present_polls = 0;
double present_ms = 0.0;

float rot_x = 25.0f;
float rot_y = 45.0f + (float)frame * 0.5f;
float rot_z = 0.0f;
float cam_dist = 4.0f;

clock_gettime(CLOCK_MONOTONIC, &t0);
if (render_frame_profile(fd, dry_run, &model, rot_x, rot_y, rot_z,
                         cam_dist, &stats) < 0) {
   fprintf(stderr, "render failed on frame %d: %s\n",
           frame, strerror(errno));
   if (fd >= 0) close(fd);
   return 1;
}
clock_gettime(CLOCK_MONOTONIC, &t1);

if (!dry_run) {
   clock_gettime(CLOCK_MONOTONIC, &tp0);
   if (issue_present(fd) < 0 ||
       wait_present(fd, &present_polls) < 0) {
       fprintf(stderr, "present failed on frame %d: %s\n",
               frame, strerror(errno));
       close(fd);
       return 1;
   }
   clock_gettime(CLOCK_MONOTONIC, &tp1);
   present_ms = elapsed_sec(tp0, tp1) * 1000.0;
}

double render_ms = elapsed_sec(t0, t1) * 1000.0;
double project_ms = stats.project_sec * 1000.0;
double setup_ms = stats.setup_sec * 1000.0;
double submit_ms = stats.submit_sec * 1000.0;
double fps = render_ms + present_ms > 0.0 ?
             1000.0 / (render_ms + present_ms) : 0.0;
double bbox_avg = stats.triangles_built ?
                  (double)stats.bbox_pixels / stats.triangles_built :
```

```
                0.0;

        printf("%d,%s,%d,%d,%d,%.3f,%.3f,%.3f,%.3f,%.3f,%.2f,%u,%ld,%ld,%ld,%.1f,%ld\n",
               frame, model.name, stats.faces_total, stats.triangles_built,
               stats.triangles_culled, render_ms, project_ms, setup_ms, submit_ms,
               present_ms, fps, stats.submit.max_fifo_level,
               stats.submit.fifo_full_polls, stats.submit.submit_eagain,
               stats.submit.status_polls + present_polls, bbox_avg,
               stats.bbox_cycles_est);
        fflush(stdout);
    }

    if (fd >= 0) {
        if (g_regs) munmap((void *)g_regs, 4096);
        close(fd);
    }
    return 0;
}
```

==========

FILE: software/userspace/main.c

==========

```
// main.c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <math.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <time.h>
#include <sys/ioctl.h>
#include <sys/mman.h>
#include <libusb-1.0/libusb.h>
```

```
#include "../kernel/avalon_kernel.h"
#include "usbkeyboard.h"
#include "geometry.h"
#include "model.h"


#define SCREEN_W    256
#define SCREEN_H    240
#define DEVICE      "/dev/rasterizer"


static volatile uint32_t *g_regs = NULL;

#define STATUS_POLL_PERIOD 32

static inline void mmio_barrier(void)
{
#if defined(__arm__) || defined(__aarch64__)
   __asm__ volatile ("dsb sy" ::: "memory");
#else
   __sync_synchronize();
#endif
}

static int submit_triangle(int fd, const triangle_packet_t *pkt)
{
   if (g_regs) {
      static unsigned int tri_counter = 0;
      const uint32_t *w = (const uint32_t *)pkt;

      if ((tri_counter++ & (STATUS_POLL_PERIOD - 1)) == 0) {
         while (g_regs[RAST_STATUS_OFFSET / 4] & RAST_STATUS_FULL_BIT)
            ;
      }

      for (int i = 0; i < RAST_PACKET_NUM_WORDS; i++)
         g_regs[i] = w[i];

      mmio_barrier();
```

```
        g_regs[RAST_COMMIT_OFFSET / 4] = 1;
        return 0;
    }

    rasterizer_arg_t ra;
    memset(&ra, 0, sizeof(ra));
    ra.packet = *pkt;
    while (ioctl(fd, RASTERIZER_SUBMIT, &ra) != 0) {
        if (errno != EAGAIN) return -1;
    }
    return 0;
}


static struct libusb_device_handle *g_keyboard = NULL;
static uint8_t                     g_endpoint  = 0;

static struct usb_keyboard_packet g_pkt_cur;
static struct usb_keyboard_packet g_pkt_prev;

#define SCAN_TABLE_SIZE 256
static int g_keys[SCAN_TABLE_SIZE];
static int g_key_pressed[SCAN_TABLE_SIZE];

static int keyboard_init(void)
{
    g_keyboard = openkeyboard(&g_endpoint);
    if (!g_keyboard) {
        fprintf(stderr, "No USB keyboard found\n");
        return -1;
    }
    memset(&g_pkt_cur,  0, sizeof(g_pkt_cur));
    memset(&g_pkt_prev, 0, sizeof(g_pkt_prev));
    return 0;
}

static void keyboard_close(void)
{
    if (g_keyboard) {
```

```
        libusb_release_interface(g_keyboard, 0);
        libusb_close(g_keyboard);
        g_keyboard = NULL;
    }
    libusb_exit(NULL);
}

static void poll_keys(void)
{
    int transferred = 0;

    g_pkt_prev = g_pkt_cur;


    int rc = libusb_interrupt_transfer(g_keyboard, g_endpoint,
                                (unsigned char *)&g_pkt_cur,
                                sizeof(g_pkt_cur),
                                &transferred, 1);
    if (rc != 0 || transferred != (int)sizeof(g_pkt_cur))
        g_pkt_cur = g_pkt_prev;

    memset(g_keys,        0, sizeof(g_keys));
    memset(g_key_pressed, 0, sizeof(g_key_pressed));

    for (int i = 0; i < 6; i++) {
        uint8_t kc = g_pkt_cur.keycode[i];
        if (kc == 0) continue;

        g_keys[kc] = 1;


        int was_held = 0;
        for (int j = 0; j < 6; j++) {
            if (g_pkt_prev.keycode[j] == kc) { was_held = 1; break; }
        }
        if (!was_held)
            g_key_pressed[kc] = 1;
    }
}
```

```
static screen_vertex_t g_sv[MODEL_MAX_VERTS];

static void project_vertices(const model_t *model, const mat4_t *mvp)
{
   for (int i = 0; i < model->num_verts; i++) {
      vec3_t v    = model->verts[i];
      vec4_t clip = mat4_mul_vec4(mvp, (vec4_t){ v.x, v.y, v.z, 1.0f });

      if (fabsf(clip.w) < 1e-6f) clip.w = 1e-6f;
      float invw = 1.0f / clip.w;
      float ndcx =  clip.x * invw;
      float ndcy =  clip.y * invw;
      float ndcz =  clip.z * invw;

      g_sv[i].sx = (ndcx + 1.0f) * 0.5f * SCREEN_W;
      g_sv[i].sy = (1.0f - ndcy) * 0.5f * SCREEN_H;


      float sz = (ndcz + 1.0f) * 0.5f;
      if (sz < 0.0f) sz = 0.0f;
      if (sz > 1.0f) sz = 1.0f;
      g_sv[i].sz = sz;
   }
}


static int render_frame(int fd, const model_t *model,
                        float rot_x, float rot_y, float rot_z,
                        float cam_dist)
{
   float aspect = (float)SCREEN_W / (float)SCREEN_H;

   mat4_t proj      = perspective(60.0f, aspect, 0.1f, 100.0f);
   mat4_t view      = translation(0.0f, 0.0f, -cam_dist);
   mat4_t rz        = rotation_z(rot_z);
   mat4_t ry        = rotation_y(rot_y);
   mat4_t rx        = rotation_x(rot_x);
```

```
    mat4_t ryx       = mat4_mul(&ry, &rx);
    mat4_t model_mat = mat4_mul(&rz, &ryx);
    mat4_t vp        = mat4_mul(&proj, &view);
    mat4_t mvp       = mat4_mul(&vp, &model_mat);

    vec3_t light_dir = vec3_normalize((vec3_t){ 0.4f, 0.7f, 0.5f });
    float  base_r = 0.2f, base_g = 0.7f, base_b = 1.0f;

    project_vertices(model, &mvp);

    int submitted = 0;
    for (int i = 0; i < model->num_faces; i++) {
        const face_t *f = &model->faces[i];



        vec3_t p0 = model->verts[f->v[0]];
        vec3_t p1 = model->verts[f->v[1]];
        vec3_t p2 = model->verts[f->v[2]];
        vec3_t face_n = vec3_normalize(
                        vec3_cross(vec3_sub(p1, p0), vec3_sub(p2, p0)));

        vec4_t tn = mat4_mul_vec4(&model_mat,
                    (vec4_t){ face_n.x, face_n.y, face_n.z, 0.0f });
        vec3_t wn    = vec3_normalize((vec3_t){ tn.x, tn.y, tn.z });
        uint8_t color = shade_face(wn, light_dir, base_r, base_g, base_b);

        triangle_packet_t pkt;
        if (setup_triangle(&g_sv[f->v[0]], &g_sv[f->v[1]], &g_sv[f->v[2]],
                           color, &pkt) < 0)
            continue;

        if (submit_triangle(fd, &pkt) < 0) {
            fprintf(stderr, "submit_triangle failed: %s\n", strerror(errno));
            return -1;
        }
        submitted++;
    }
    return submitted;
}
```

```
static int load_named_obj(model_t *out, const char *stem,
                          char *loaded_path, size_t loaded_path_sz)
{
   char candidates[4][128];
   snprintf(candidates[0], sizeof(candidates[0]), "models/%s", stem);
   snprintf(candidates[1], sizeof(candidates[1]), "models/%s.obj", stem);
   snprintf(candidates[2], sizeof(candidates[2]), "%s", stem);
   snprintf(candidates[3], sizeof(candidates[3]), "%s.obj", stem);

   for (int i = 0; i < 4; i++) {
      if (model_load_obj(out, candidates[i]) == 0) {
         if (loaded_path && loaded_path_sz > 0) {
            snprintf(loaded_path, loaded_path_sz, "%s", candidates[i]);
         }
         return 0;
      }
   }

   if (loaded_path && loaded_path_sz > 0) loaded_path[0] = '\0';
   return -1;
}


int main(int argc, char *argv[])
{


   enum {
      MODEL_CUBE = 0,
      MODEL_SPHERE_LO,
      MODEL_SPHERE_MED,
      MODEL_SPHERE_HI,
      MODEL_SPHERE_ULTRA,
      MODEL_SPHERE_MAX,
      MODEL_TORUS,
      MODEL_SATURN,
      MODEL_LEGO,
```

```
    MODEL_DNA,
    MODEL_ARGV_OBJ,
    MODEL_OBJ1,
    MODEL_OBJ2,
    MODEL_OBJ3,
    MODEL_THINKER,
    MODEL_ALMA_MATER,
    MODEL_ALMA_MATER_COMP,
    MODEL_LION,
    MODEL_CROWN,
    MODEL_CROWN_80K,
    MODEL_CROWN_100K,
    MODEL_COUNT
};

static model_t models[MODEL_COUNT];
int num_models = 10;
int obj1_loaded = 0, obj2_loaded = 0, obj3_loaded = 0;
int thinker_loaded = 0, alma_loaded = 0, alma_comp_loaded = 0, lion_loaded = 0;
int crown_loaded = 0, crown_80k_loaded = 0, crown_100k_loaded = 0;
char obj1_path[128], obj2_path[128], obj3_path[128];
char thinker_path[128], alma_path[128], alma_comp_path[128], lion_path[128];
char crown_path[128], crown_80k_path[128], crown_100k_path[128];

model_make_cube(&models[MODEL_CUBE]);
model_make_icosphere(&models[MODEL_SPHERE_LO],    1);
model_make_icosphere(&models[MODEL_SPHERE_MED],   2);
model_make_icosphere(&models[MODEL_SPHERE_HI],    3);
model_make_icosphere(&models[MODEL_SPHERE_ULTRA], 4);
model_make_icosphere(&models[MODEL_SPHERE_MAX],   5);
model_make_torus(&models[MODEL_TORUS], 12, 8, 0.7f, 0.3f);
model_make_saturn(&models[MODEL_SATURN]);
model_make_lego(&models[MODEL_LEGO]);
model_make_dna(&models[MODEL_DNA]);

if (argc >= 2) {
    if (model_load_obj(&models[MODEL_ARGV_OBJ], argv[1]) == 0) {
        printf("Loaded %s: %d verts, %d faces\n",
               argv[1],
```

```
            models[MODEL_ARGV_OBJ].num_verts,
            models[MODEL_ARGV_OBJ].num_faces);
        num_models = 10;
    } else {
        fprintf(stderr, "Warning: could not load %s\n", argv[1]);
    }
}


if (load_named_obj(&models[MODEL_OBJ1], "obj1", obj1_path, sizeof(obj1_path)) == 0) {
    obj1_loaded = 1;
    strncpy(models[MODEL_OBJ1].name, "obj1", sizeof(models[MODEL_OBJ1].name) - 1);
    models[MODEL_OBJ1].name[sizeof(models[MODEL_OBJ1].name) - 1] = '\0';
} else {
    model_make_cube(&models[MODEL_OBJ1]);
    strncpy(models[MODEL_OBJ1].name, "obj1_missing", sizeof(models[MODEL_OBJ1].name) - 1);
}
if (load_named_obj(&models[MODEL_OBJ2], "obj2", obj2_path, sizeof(obj2_path)) == 0) {
    obj2_loaded = 1;
    strncpy(models[MODEL_OBJ2].name, "obj2", sizeof(models[MODEL_OBJ2].name) - 1);
    models[MODEL_OBJ2].name[sizeof(models[MODEL_OBJ2].name) - 1] = '\0';
} else {
    model_make_cube(&models[MODEL_OBJ2]);
    strncpy(models[MODEL_OBJ2].name, "obj2_missing", sizeof(models[MODEL_OBJ2].name) - 1);
}
if (load_named_obj(&models[MODEL_OBJ3], "obj3", obj3_path, sizeof(obj3_path)) == 0) {
    obj3_loaded = 1;
    strncpy(models[MODEL_OBJ3].name, "obj3", sizeof(models[MODEL_OBJ3].name) - 1);
    models[MODEL_OBJ3].name[sizeof(models[MODEL_OBJ3].name) - 1] = '\0';
} else {
    model_make_cube(&models[MODEL_OBJ3]);
    strncpy(models[MODEL_OBJ3].name, "obj3_missing", sizeof(models[MODEL_OBJ3].name) - 1);
}
if (load_named_obj(&models[MODEL_THINKER], "Thinker", thinker_path, sizeof(thinker_path)) == 0) {
    thinker_loaded = 1;
    strncpy(models[MODEL_THINKER].name, "Thinker", sizeof(models[MODEL_THINKER].name) - 1);
    models[MODEL_THINKER].name[sizeof(models[MODEL_THINKER].name) - 1] = '\0';
} else {
    model_make_cube(&models[MODEL_THINKER]);
```

```
1);     strncpy(models[MODEL_THINKER].name, "Thinker_missing", sizeof(models[MODEL_THINKER].name) -
    }
    if (load_named_obj(&models[MODEL_ALMA_MATER], "alma_mater", alma_path, sizeof(alma_path)) == 0) {
        alma_loaded = 1;
        strncpy(models[MODEL_ALMA_MATER].name, "alma_mater",
sizeof(models[MODEL_ALMA_MATER].name) - 1);
        models[MODEL_ALMA_MATER].name[sizeof(models[MODEL_ALMA_MATER].name) - 1] = '\0';
    } else {
        model_make_cube(&models[MODEL_ALMA_MATER]);
        strncpy(models[MODEL_ALMA_MATER].name, "alma_mater_missing",
sizeof(models[MODEL_ALMA_MATER].name) - 1);
    }
    if (load_named_obj(&models[MODEL_ALMA_MATER_COMP], "alma_mater_compressed", alma_comp_path,
sizeof(alma_comp_path)) == 0) {
        alma_comp_loaded = 1;
        strncpy(models[MODEL_ALMA_MATER_COMP].name, "alma_mater_comp",
sizeof(models[MODEL_ALMA_MATER_COMP].name) - 1);
        models[MODEL_ALMA_MATER_COMP].name[sizeof(models[MODEL_ALMA_MATER_COMP].name) -
1] = '\0';
    } else {
        model_make_cube(&models[MODEL_ALMA_MATER_COMP]);
        strncpy(models[MODEL_ALMA_MATER_COMP].name, "alma_comp_missing",
sizeof(models[MODEL_ALMA_MATER_COMP].name) - 1);
    }
    if (load_named_obj(&models[MODEL_LION], "wooden_lion_sculpture_derivative", lion_path,
 sizeof(lion_path)) == 0) {
        lion_loaded = 1;
        strncpy(models[MODEL_LION].name, "lion", sizeof(models[MODEL_LION].name) - 1);
        models[MODEL_LION].name[sizeof(models[MODEL_LION].name) - 1] = '\0';
    } else {
        model_make_cube(&models[MODEL_LION]);
        strncpy(models[MODEL_LION].name, "lion_missing", sizeof(models[MODEL_LION].name) - 1);
    }
    if (load_named_obj(&models[MODEL_CROWN], "ColumbiaCrown", crown_path, sizeof(crown_path)) == 0) {
        crown_loaded = 1;
        strncpy(models[MODEL_CROWN].name, "Crown", sizeof(models[MODEL_CROWN].name) - 1);
        models[MODEL_CROWN].name[sizeof(models[MODEL_CROWN].name) - 1] = '\0';
    } else {
        model_make_cube(&models[MODEL_CROWN]);
```

```
        strncpy(models[MODEL_CROWN].name, "Crown_missing", sizeof(models[MODEL_CROWN].name) - 1);
    }
    if (load_named_obj(&models[MODEL_CROWN_80K], "ColumbiaCrown_80K", crown_80k_path,
sizeof(crown_80k_path)) == 0) {
        crown_80k_loaded = 1;
        strncpy(models[MODEL_CROWN_80K].name, "Crown_80K",
sizeof(models[MODEL_CROWN_80K].name) - 1);
        models[MODEL_CROWN_80K].name[sizeof(models[MODEL_CROWN_80K].name) - 1] = '\0';
    } else {
        model_make_cube(&models[MODEL_CROWN_80K]);
        strncpy(models[MODEL_CROWN_80K].name, "Crown_80K_missing",
sizeof(models[MODEL_CROWN_80K].name) - 1);
    }
    if (load_named_obj(&models[MODEL_CROWN_100K], "ColumbiaCrown_100K", crown_100k_path,
sizeof(crown_100k_path)) == 0) {
        crown_100k_loaded = 1;
        strncpy(models[MODEL_CROWN_100K].name, "Crown_100K",
sizeof(models[MODEL_CROWN_100K].name) - 1);
        models[MODEL_CROWN_100K].name[sizeof(models[MODEL_CROWN_100K].name) - 1] = '\0';
    } else {
        model_make_cube(&models[MODEL_CROWN_100K]);
        strncpy(models[MODEL_CROWN_100K].name, "Crown_100K_missing",
sizeof(models[MODEL_CROWN_100K].name) - 1);
    }

    int fd = open(DEVICE, O_RDWR);
    if (fd < 0) {
        fprintf(stderr, "Cannot open %s: %s\n", DEVICE, strerror(errno));
        fprintf(stderr, "Is the kernel module loaded? "
                        "sudo insmod avalon_kernel.ko\n");
        return 1;
    }


    void *mapped = mmap(NULL, 4096, PROT_READ | PROT_WRITE,
                        MAP_SHARED, fd, 0);
    if (mapped != MAP_FAILED) {
        g_regs = (volatile uint32_t *)mapped;
        printf("mmap OK: direct MMIO submission enabled\n");
```

```
} else {
    g_regs = NULL;
    fprintf(stderr, "mmap failed (%s); using ioctl fallback\n",
            strerror(errno));
}


if (keyboard_init() < 0) {
    close(fd);
    return 1;
}


printf("\n=== CSEE 4840 Hardware Rasterizer Demo ===\n");
printf("Controls (USB keyboard on DE1-SoC):\n");
printf("  w/s     - tilt up/down\n");
printf("  a/d     - rotate left/right\n");
printf("  +/-     - zoom in/out\n");
printf("  1       - cube\n");
printf("  2       - sphere lo   (80 tris)\n");
printf("  3       - sphere med  (320 tris)\n");
printf("  4       - sphere hi   (1280 tris)\n");
printf("  5       - sphere max  (5120 tris)\n");
printf("  0       - sphere stress (20480 tris)\n");
printf("  6       - torus\n");
printf("  7       - saturn\n");
printf("  8       - lego brick\n");
printf("  9       - dna helix\n");
printf("  b       - obj1\n");
printf("  m       - obj2\n");
printf("  n       - obj3\n");
printf("  t       - Thinker       (%s)\n", thinker_loaded  ? thinker_path   : "NOT FOUND");
printf("  f       - alma mater    (%s)\n", alma_loaded      ? alma_path       : "NOT FOUND");
printf("  c       - alma (compressed) (%s)\n", alma_comp_loaded ? alma_comp_path : "NOT FOUND");
printf("  l       - wooden lion   (%s)\n", lion_loaded      ? lion_path       : "NOT FOUND");
printf("  e       - Columbia Crown (%s)\n",      crown_loaded     ? crown_path      : "NOT FOUND");
printf("  g       - Crown 80K     (%s)\n",       crown_80k_loaded ? crown_80k_path  : "NOT FOUND");
printf("  h       - Crown 100K    (%s)\n",       crown_100k_loaded? crown_100k_path : "NOT FOUND");
if (num_models > 10) printf("  (OBJ)   - %s\n", models[MODEL_ARGV_OBJ].name);
```

```
printf("OBJ key mapping:\n");
printf("  b -> obj1: %s\n", obj1_loaded ? obj1_path : "NOT FOUND");
printf("  m -> obj2: %s\n", obj2_loaded ? obj2_path : "NOT FOUND");
printf("  n -> obj3: %s\n", obj3_loaded ? obj3_path : "NOT FOUND");
printf("  r       - reset rotation\n");
printf("  SPACE   - toggle auto-rotate\n");
printf("  ESC/q   - quit\n");
printf("==========================================\n\n");


int   current_model = 2;
float rot_x      = 25.0f;
float rot_y      = 45.0f;
float rot_z      =  0.0f;
float cam_dist   =  4.0f;
float rot_speed  =  2.0f;
int   auto_rotate = 0;
int   running    = 1;
long  frame      = 0;
float fps        = 0.0f;

struct timespec t_prev, t_now;
clock_gettime(CLOCK_MONOTONIC, &t_prev);

while (running) {


   poll_keys();


   if (g_keys[KEY_W]) rot_x -= rot_speed;
   if (g_keys[KEY_S]) rot_x += rot_speed;
   if (g_keys[KEY_A]) rot_y -= rot_speed;
   if (g_keys[KEY_D]) rot_y += rot_speed;
   if (g_keys[KEY_EQUAL]) {
      cam_dist -= 0.3f;
      if (cam_dist < 1.5f) cam_dist = 1.5f;
   }
   if (g_keys[KEY_MINUS]) {
```

```
    cam_dist += 0.3f;
    if (cam_dist > 15.0f) cam_dist = 15.0f;
}


if (g_key_pressed[KEY_ESC] || g_key_pressed[KEY_Q]) running = 0;
if (g_key_pressed[KEY_1]) current_model = MODEL_CUBE;
if (g_key_pressed[KEY_2]) current_model = MODEL_SPHERE_LO;
if (g_key_pressed[KEY_3]) current_model = MODEL_SPHERE_MED;
if (g_key_pressed[KEY_4]) current_model = MODEL_SPHERE_HI;
if (g_key_pressed[KEY_5]) current_model = MODEL_SPHERE_ULTRA;
if (g_key_pressed[KEY_0]) current_model = MODEL_SPHERE_MAX;
if (g_key_pressed[KEY_6]) current_model = MODEL_TORUS;
if (g_key_pressed[KEY_7]) current_model = MODEL_SATURN;
if (g_key_pressed[KEY_8]) current_model = MODEL_LEGO;
if (g_key_pressed[KEY_9]) current_model = MODEL_DNA;
if (g_key_pressed[KEY_B]) {
    if (obj1_loaded) current_model = MODEL_OBJ1;
    else fprintf(stderr, "\nobj1 not found. Place it at software/models/obj1.obj\n");
}
if (g_key_pressed[KEY_M]) {
    if (obj2_loaded) current_model = MODEL_OBJ2;
    else fprintf(stderr, "\nobj2 not found. Place it at software/models/obj2.obj\n");
}
if (g_key_pressed[KEY_N]) {
    if (obj3_loaded) current_model = MODEL_OBJ3;
    else fprintf(stderr, "\nobj3 not found. Place it at software/models/obj3.obj\n");
}
if (g_key_pressed[KEY_T]) {
    if (thinker_loaded) current_model = MODEL_THINKER;
    else fprintf(stderr, "\nThinker.obj not found in software/models/\n");
}
if (g_key_pressed[KEY_F]) {
    if (alma_loaded) current_model = MODEL_ALMA_MATER;
    else fprintf(stderr, "\nalma_mater.obj not found in software/models/\n");
}
if (g_key_pressed[KEY_C]) {
    if (alma_comp_loaded) current_model = MODEL_ALMA_MATER_COMP;
    else fprintf(stderr, "\nalma_mater_compressed.obj not found in software/models/\n");
```

```
}
if (g_key_pressed[KEY_L]) {
   if (lion_loaded) current_model = MODEL_LION;
   else fprintf(stderr, "\nwooden_lion_sculpture_derivative.obj not found in software/models/\n");
}
if (g_key_pressed[KEY_E]) {
   if (crown_loaded) current_model = MODEL_CROWN;
   else fprintf(stderr, "\nColumbiaCrown.obj not found in software/models/\n");
}
if (g_key_pressed[KEY_G]) {
   if (crown_80k_loaded) current_model = MODEL_CROWN_80K;
   else fprintf(stderr, "\nColumbiaCrown_80K.obj not found in software/models/\n");
}
if (g_key_pressed[KEY_H]) {
   if (crown_100k_loaded) current_model = MODEL_CROWN_100K;
   else fprintf(stderr, "\nColumbiaCrown_100K.obj not found in software/models/\n");
}
if (g_key_pressed[KEY_R]) {
   rot_x = 25.0f; rot_y = 45.0f; rot_z = 0.0f; cam_dist = 4.0f;
}
if (g_key_pressed[KEY_SPACE]) auto_rotate = !auto_rotate;

if (auto_rotate) rot_y += 0.5f;


int n = render_frame(fd, &models[current_model],
                     rot_x, rot_y, rot_z, cam_dist);
if (n < 0) {
   fprintf(stderr, "render_frame error, aborting\n");
   break;
}

if (ioctl(fd, RASTERIZER_PRESENT) < 0) {
   fprintf(stderr, "present failed: %s\n", strerror(errno));
   break;
}


rasterizer_arg_t ra;
```

```
int present_failed = 0;


do {
   memset(&ra, 0, sizeof(ra));
   if (ioctl(fd, RASTERIZER_STATUS, &ra) < 0) {
      fprintf(stderr, "status poll (assert) failed: %s\n",
            strerror(errno));
      present_failed = 1;
      break;
   }
} while (!ra.status.swap_busy);


while (!present_failed) {
   memset(&ra, 0, sizeof(ra));
   if (ioctl(fd, RASTERIZER_STATUS, &ra) < 0) {
      fprintf(stderr, "status poll (deassert) failed: %s\n",
            strerror(errno));
      present_failed = 1;
      break;
   }
   if (!ra.status.swap_busy) break;

}

if (present_failed) { running = 0; break; }


clock_gettime(CLOCK_MONOTONIC, &t_now);
float dt = (t_now.tv_sec  - t_prev.tv_sec) +
         (t_now.tv_nsec - t_prev.tv_nsec) * 1e-9f;
t_prev = t_now;
if (dt > 1e-6f) fps = 1.0f / dt;


frame++;
printf("\rFrame %ld | %s (%d tris) | FPS: %5.1f | "
      "rx=%.1f ry=%.1f dist=%.1f    ",
```

```
            frame, models[current_model].name,
            models[current_model].num_faces,
            fps, rot_x, rot_y, cam_dist);
        fflush(stdout);
    }

    printf("\nDone. %ld frames rendered.\n", frame);
    keyboard_close();
    if (g_regs) munmap((void *)g_regs, 4096);
    close(fd);
    return 0;
}
```